Heat Documentation

Release 25.1.0.dev52

Heat Developers

Feb 12, 2026

CONTENTS

1 Heats purpose and vision **3**

2 Operating Heat **5**

2.1 Installing Heat . . . 5

2.1.1 Orchestration service overview . . . 5

2.1.2 Install and configure . . . 5

Install and configure for Red Hat Enterprise Linux and CentOS . . . 6

Install and configure for Ubuntu . . . 13

Install and configure for Debian . . . 20

2.1.3 Verify operation . . . 28

2.1.4 Launch an instance . . . 28

Create a template . . . 29

Create a stack . . . 29

2.1.5 Next steps . . . 31

2.2 Running Heat API services in HTTP Server . . . 31

2.2.1 mod-wsgi . . . 32

2.2.2 uwsgi . . . 32

mod_proxy_uwsgi . . . 33

2.3 Configuring Heat . . . 33

2.3.1 Configuration options for the Orchestration service . . . 33

DEFAULT . . . 33

auth_password . . . 54

cache . . . 54

clients . . . 62

clients_aodh . . . 63

clients_barbican . . . 64

clients_cinder . . . 65

clients_designate . . . 66

clients_glance . . . 67

clients_heat . . . 68

clients_keystone . . . 69

clients_magnum . . . 70

clients_manila . . . 70

clients_mistral . . . 71

clients_neutron . . . 72

clients_nova . . . 73

clients_octavia . . . 74

clients_swift . . . 75

clients_trove . . . 76

clients_vitrage . . . 76
clients_zaqar . . . 77
constraint_validation_cache . . . 78
cors . . . 79
database . . . 80
ec2authtoken . . . 84
healthcheck . . . 92
keystone_authtoken . . . 93
noauth . . . 101
oslo_messaging_kafka . . . 101
oslo_messaging_notifications . . . 103
oslo_messaging_rabbit . . . 104
oslo_middleware . . . 112
oslo_policy . . . 113
oslo_reports . . . 115
oslo_versionedobjects . . . 116
paste_deploy . . . 116
profiler . . . 116
profiler_jaeger . . . 119
profiler_otlp . . . 120
resource_finder_cache . . . 120
revision . . . 120
service_extension_cache . . . 120
trustee . . . 121
volumes . . . 124
yaql . . . 125
2.3.2 Heat Configuration Sample . . . 125
2.3.3 Orchestration log files . . . 125
2.3.4 Heat Sample Policy . . . 125
2.4 Administering Heat . . . 143
2.4.1 Introduction . . . 143
2.4.2 Orchestration authorization model . . . 144
Password authorization . . . 144
OpenStack Identity trusts authorization . . . 144
Authorization model configuration . . . 145
2.4.3 Stack domain users . . . 146
Stack domain users configuration . . . 146
Usage workflow . . . 147
2.5 Scaling a Deployment . . . 148
2.5.1 Assumptions . . . 148
2.5.2 Architecture . . . 148
Basic Architecture . . . 148
Load Balancing . . . 148
Target Architecture . . . 149
2.5.3 Deploying Multiple APIs . . . 149
2.5.4 Deploying Multiple Engines . . . 149
2.5.5 Deploying the Proxy . . . 150
2.5.6 Sample . . . 151
Architecture . . . 151
Running the API and Engine Services . . . 151
Setting Up HAProxy . . . 152

2.6 Upgrades Guideline . . . 153
2.6.1 Plan to upgrade . . . 153
2.6.2 Cold Upgrades . . . 153
2.6.3 Rolling Upgrades . . . 153
Prerequisites . . . 154
Procedure . . . 154
2.6.4 References . . . 155
2.7 Man pages for services and utilities . . . 155
2.7.1 Heat services . . . 155
heat-engine . . . 155
heat-api . . . 156
heat-api-cfn . . . 156
2.7.2 Heat utilities . . . 157
heat-manage . . . 157
heat-db-setup . . . 158
heat-keystone-setup-domain . . . 159
heat-status . . . 160

3 Using Heat **163**
3.1 Creating your first stack . . . 163
3.1.1 Confirming you can access a Heat endpoint . . . 163
3.1.2 Preparing to create a stack . . . 163
3.1.3 Launching a stack . . . 164
List stacks . . . 164
List stack events . . . 164
Describe the wordpress stack . . . 164
Verify instance creation . . . 165
Delete the instance when done . . . 165
3.2 Glossary . . . 165
3.3 Working with Templates . . . 167
3.3.1 Template Guide . . . 167
Heat Orchestration Template (HOT) Guide . . . 167
Writing a hello world HOT template . . . 170
Guideline for features . . . 174
Heat Orchestration Template (HOT) specification . . . 177
Instances . . . 212
Software configuration . . . 220
Environments . . . 234
Template composition . . . 238
OpenStack Resource Types . . . 241
CloudFormation Compatible Resource Types . . . 568
Unsupported Heat Resource Types . . . 606
Contributed Heat Resource Types . . . 621
CloudFormation Compatible Functions . . . 622
3.3.2 Example Templates . . . 629
Example HOT Templates . . . 629
Example CFN Templates . . . 630
3.4 Using the Heat Service . . . 631

4 Developing Heat **633**
4.1 Heat Developer Guidelines . . . 633
4.1.1 Heat and DevStack . . . 633

Configure DevStack to enable heat . . . 633
Configure DevStack to enable ceilometer and aodh (if using alarms) . . . 634
Configure DevStack to enable OSprofiler . . . 634
Create a stack . . . 635
4.1.2 Blueprints and Specs . . . 635
Spec from existing stories . . . 635
4.1.3 Heat architecture . . . 635
Detailed description . . . 635
Heat services . . . 635
4.1.4 Heat Resource Plug-in Development Guide . . . 636
Resource Plug-in Life Cycle . . . 636
Configuring the Engine . . . 648
Testing . . . 648
Putting It All Together . . . 648
Resource Contributions . . . 648
4.1.5 Heat Stack Lifecycle Scheduler Hints . . . 648
Enabling the scheduler hints . . . 648
The hints . . . 649
Purpose . . . 649
4.1.6 Guru Meditation Reports . . . 649
Generating a GMR . . . 649
Structure of a GMR . . . 649
Adding support for GMRs to new executable . . . 649
Extending the GMR . . . 650
4.1.7 Heat Support Status usage Guide . . . 650
Support Status option and its parameters . . . 650
Life cycle of resource, property, attribute . . . 651
Using Support Status during code writing . . . 652
Translating mechanism for hidden properties . . . 653
4.1.8 Using Rally on Heat gates . . . 654
How to run Rally for particular patch . . . 654
Examples of using Rally . . . 655
create_stack_and_show_output_old . . . 657
create_stack_and_show_output_new . . . 657
create_stack_and_list_output_old . . . 657
create_stack_and_list_output_new . . . 657
4.2 Source Code Index . . . 658
4.2.1 heat . . . 658
heat package . . . 658

5 For Contributors **897**
5.1 Heat Contributor Guidelines . . . 897
5.1.1 So You Want to Contribute . . . 897
Communication . . . 897
Contacting the Core Team . . . 897
New Feature Planning . . . 897
Task Tracking . . . 897
Reporting a Bug . . . 897
Getting Your Patch Merged . . . 897
Project Team Lead Duties . . . 898

6 Indices and tables **899**

Heat is a service to orchestrate composite cloud applications using a declarative template format through an OpenStack-native REST API.

CHAPTER

ONE

HEATS PURPOSE AND VISION

- Heat provides a template based orchestration for describing a cloud application by executing appropriate *OpenStack* API calls to generate running cloud applications.
- A Heat template describes the infrastructure for a cloud application in text files which are readable and writable by humans, and can be managed by version control tools.
- Templates specify the relationships between resources (e.g. this volume is connected to this server). This enables Heat to call out to the OpenStack APIs to create all of your infrastructure in the correct order to completely launch your application.
- The software integrates other components of OpenStack. The templates allow creation of most OpenStack resource types (such as instances, floating ips, volumes, security groups, users, etc), as well as some more advanced functionality such as instance high availability, instance autoscaling, and nested stacks.
- Heat primarily manages infrastructure, but the templates integrate well with software configuration management tools such as Puppet and Ansible.
- Operators can customise the capabilities of Heat by installing plugins.

This documentation offers information aimed at end-users, operators and developers of Heat.

CHAPTER

TWO

OPERATING HEAT

2.1 Installing Heat

2.1.1 Orchestration service overview

The Orchestration service provides a template-based orchestration for describing a cloud application by running OpenStack API calls to generate running cloud applications. The software integrates other core components of OpenStack into a one-file template system. The templates allow you to create most OpenStack resource types such as instances, floating IPs, volumes, security groups, and users. It also provides advanced functionality such as instance high availability, instance auto-scaling, and nested stacks. This enables OpenStack core projects to receive a larger user base.

The service allows deployers to integrate with the Orchestration service directly or through custom plugins.

The Orchestration service consists of the following components:

`heat` command-line client
A CLI that communicates with the `heat-api` to run AWS CloudFormation APIs. End developers can directly use the Orchestration REST API.

`heat-api` component
An OpenStack-native REST API that processes API requests by sending them to the `heat-engine` over Remote Procedure Call (RPC).

`heat-api-cfn` component
An AWS Query API that is compatible with AWS CloudFormation. It processes API requests by sending them to the `heat-engine` over RPC.

`heat-engine`
Orchestrates the launching of templates and provides events back to the API consumer.

2.1.2 Install and configure

This section describes how to install and configure the Orchestration service, code-named heat, on the controller node.

This section assumes that you already have a working OpenStack environment with at least the following components installed: Compute, Image Service, Identity.

Note that installation and configuration vary by distribution.

Install and configure for Red Hat Enterprise Linux and CentOS

This section describes how to install and configure the Orchestration service for Red Hat Enterprise Linux 7 and CentOS 7.

Prerequisites

Before you install and configure Orchestration, you must create a database, service credentials, and API endpoints. Orchestration also requires additional information in the Identity service.

1. To create the database, complete these steps:
   - Use the database access client to connect to the database server as the `root` user:

     ```
     $ mysql -u root -p
     ```

   - Create the `heat` database:

     ```
     CREATE DATABASE heat;
     ```

   - Grant proper access to the `heat` database:

     ```
     GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost' \
       IDENTIFIED BY 'HEAT_DBPASS';
     GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%' \
       IDENTIFIED BY 'HEAT_DBPASS';
     ```

     Replace `HEAT_DBPASS` with a suitable password.

   - Exit the database access client.
2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:
   - Create the `heat` user:

     ```
     $ openstack user create --domain default --password-prompt heat
     User Password:
     Repeat User Password:
     +-----------+----------------------------------+
     | Field     | Value                            |
     +-----------+----------------------------------+
     | domain_id | e0353a670a9e496da891347c589539e9 |
     | enabled   | True                             |
     | id        | ca2e175b851943349be29a328cc5e360 |
     | name      | heat                             |
     +-----------+----------------------------------+
     ```

   - Add the `admin` role to the `heat` user:

     ```
     $ openstack role add --project service --user heat admin
     ```

> **Note**
>
> If installing OpenStack manually following the Keystone install guide, the name of the services project is `service` as given above. However, traditional methods of installing RDO (such as PackStack and TripleO) use `services` as the name of the service project. If you installed RDO using a Puppet-based method, substitute `services` as the project name.

> **Note**
>
> This command provides no output.

- Create the `heat` and `heat-cfn` service entities:

```
$ openstack service create --name heat \
  --description "Orchestration" orchestration
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Orchestration                    |
| enabled     | True                             |
| id          | 727841c6f5df4773baa4e8a5ae7d72eb |
| name        | heat                             |
| type        | orchestration                    |
+-------------+----------------------------------+

$ openstack service create --name heat-cfn \
  --description "Orchestration"  cloudformation
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Orchestration                    |
| enabled     | True                             |
| id          | c42cede91a4e47c3b10c8aedc8d890c6 |
| name        | heat-cfn                         |
| type        | cloudformation                   |
+-------------+----------------------------------+
```

4. Create the Orchestration service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  orchestration public http://controller:8004/v1/%\(tenant_id\)s
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | 3f4dab34624e4be7b000265f25049609       |
| interface    | public                                 |
| region       | RegionOne                              |
```

(continues on next page)

(continued from previous page)

```
| region_id     | RegionOne                                |
| service_id    | 727841c6f5df4773baa4e8a5ae7d72eb         |
| service_name  | heat                                     |
| service_type  | orchestration                            |
| url           | http://controller:8004/v1/%(tenant_id)s  |
+---------------+------------------------------------------+

$ openstack endpoint create --region RegionOne \
  orchestration internal http://controller:8004/v1/%\(tenant_id\)s
+---------------+------------------------------------------+
| Field         | Value                                    |
+---------------+------------------------------------------+
| enabled       | True                                     |
| id            | 9489f78e958e45cc85570fec7e836d98         |
| interface     | internal                                 |
| region        | RegionOne                                |
| region_id     | RegionOne                                |
| service_id    | 727841c6f5df4773baa4e8a5ae7d72eb         |
| service_name  | heat                                     |
| service_type  | orchestration                            |
| url           | http://controller:8004/v1/%(tenant_id)s  |
+---------------+------------------------------------------+

$ openstack endpoint create --region RegionOne \
  orchestration admin http://controller:8004/v1/%\(tenant_id\)s
+---------------+------------------------------------------+
| Field         | Value                                    |
+---------------+------------------------------------------+
| enabled       | True                                     |
| id            | 76091559514b40c6b7b38dde790efe99         |
| interface     | admin                                    |
| region        | RegionOne                                |
| region_id     | RegionOne                                |
| service_id    | 727841c6f5df4773baa4e8a5ae7d72eb         |
| service_name  | heat                                     |
| service_type  | orchestration                            |
| url           | http://controller:8004/v1/%(tenant_id)s  |
+---------------+------------------------------------------+
```

```
$ openstack endpoint create --region RegionOne \
  cloudformation public http://controller:8000/v1
+---------------+----------------------------------+
| Field         | Value                            |
+---------------+----------------------------------+
| enabled       | True                             |
| id            | b3ea082e019c4024842bf0a80555052c |
| interface     | public                           |
| region        | RegionOne                        |
| region_id     | RegionOne                        |
```

(continues on next page)

(continued from previous page)

```
| service_id   | c42cede91a4e47c3b10c8aedc8d890c6 |
| service_name | heat-cfn                         |
| service_type | cloudformation                   |
| url          | http://controller:8000/v1        |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  cloudformation internal http://controller:8000/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 169df4368cdc435b8b115a9cb084044e |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | c42cede91a4e47c3b10c8aedc8d890c6 |
| service_name | heat-cfn                         |
| service_type | cloudformation                   |
| url          | http://controller:8000/v1        |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  cloudformation admin http://controller:8000/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 3d3edcd61eb343c1bbd629aa041ff88b |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | c42cede91a4e47c3b10c8aedc8d890c6 |
| service_name | heat-cfn                         |
| service_type | cloudformation                   |
| url          | http://controller:8000/v1        |
+--------------+----------------------------------+
```

5. Orchestration requires additional information in the Identity service to manage stacks. To add this information, complete these steps:

   - Create the `heat` domain that contains projects and users for stacks:

```
$ openstack domain create --description "Stack projects and users"␣
↪heat
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Stack projects and users         |
| enabled     | True                             |
```

(continues on next page)

(continued from previous page)

```
| id          | 0f4d1bd326f2454dacc72157ba328a47 |
| name        | heat                             |
+-------------+----------------------------------+
```

- Create the `heat_domain_admin` user to manage projects and users in the `heat` domain:

```
$ openstack user create --domain heat --password-prompt heat_domain_
↪admin
User Password:
Repeat User Password:
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | 0f4d1bd326f2454dacc72157ba328a47 |
| enabled   | True                             |
| id        | b7bd1abfbcf64478b47a0f13cd4d970a |
| name      | heat_domain_admin                |
+-----------+----------------------------------+
```

- Add the `admin` role to the `heat_domain_admin` user in the `heat` domain to enable administrative stack management privileges by the `heat_domain_admin` user:

```
$ openstack role add --domain heat --user-domain heat --user heat_
↪domain_admin admin
```

> **Note**
>
> This command provides no output.

- Create the `heat_stack_owner` role:

```
$ openstack role create heat_stack_owner
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 15e34f0c4fed4e68b3246275883c8630 |
| name      | heat_stack_owner                 |
+-----------+----------------------------------+
```

- Add the `heat_stack_owner` role to the `demo` project and user to enable stack management by the `demo` user:

```
$ openstack role add --project demo --user demo heat_stack_owner
```

> **Note**
>
> This command provides no output.

> **Note**
>
> You must add the `heat_stack_owner` role to each user that manages stacks.

- Create the `heat_stack_user` role:

```
$ openstack role create heat_stack_user
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 88849d41a55d4d1d91e4f11bffd8fc5c |
| name      | heat_stack_user                  |
+-----------+----------------------------------+
```

> **Note**
>
> The Orchestration service automatically assigns the `heat_stack_user` role to users that it creates during stack deployment. By default, this role restricts API <Application Programming Interface (API)> operations. To avoid conflicts, do not add this role to users with the `heat_stack_owner` role.

Install and configure components

> **Note**
>
> Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages:

```
# dnf install openstack-heat-api openstack-heat-api-cfn \
  openstack-heat-engine
```

2. Edit the `/etc/heat/heat.conf` file and complete the following actions:

   - In the `[database]` section, configure database access:

```
[database]
...
connection = mysql+pymysql://heat:HEAT_DBPASS@controller/heat
```

     Replace `HEAT_DBPASS` with the password you chose for the Orchestration database.

   - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

```
[DEFAULT]
...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

- In the [keystone_authtoken], [ec2authtoken], [trustee], and [clients_keystone] sections, configure Identity service access:

```
[keystone_authtoken]
...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = heat
password = HEAT_PASS

[ec2authtoken]
...
auth_url = http://controller:5000
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = heat
password = HEAT_PASS

[trustee]
...
auth_type = password
auth_url = http://controller:5000
username = heat
password = HEAT_PASS
user_domain_name = Default

[clients_keystone]
...
auth_uri = http://controller:5000
```

Replace HEAT_PASS with the password you chose for the heat user in the Identity service.

- In the [DEFAULT] section, configure the metadata and wait condition URLs:

```
[DEFAULT]
...
heat_metadata_server_url = http://controller:8000
heat_waitcondition_server_url = http://controller:8000/v1/
↪waitcondition
```

- In the [DEFAULT] section, configure the stack domain and administrative credentials:

```
[DEFAULT]
...
stack_domain_admin = heat_domain_admin
stack_domain_admin_password = HEAT_DOMAIN_PASS
stack_user_domain_name = heat
```

Replace `HEAT_DOMAIN_PASS` with the password you chose for the `heat_domain_admin` user in the Identity service.

3. Populate the Orchestration database:

```
# su -s /bin/sh -c "heat-manage db_sync" heat
```

> **Note**
>
> Ignore any deprecation messages in this output.

Finalize installation

- Start the Orchestration services and configure them to start when the system boots:

```
# systemctl enable openstack-heat-api.service \
  openstack-heat-api-cfn.service openstack-heat-engine.service
# systemctl start openstack-heat-api.service \
  openstack-heat-api-cfn.service openstack-heat-engine.service
```

Install and configure for Ubuntu

This section describes how to install and configure the Orchestration service for Ubuntu 14.04 (LTS).

Prerequisites

Before you install and configure Orchestration, you must create a database, service credentials, and API endpoints. Orchestration also requires additional information in the Identity service.

1. To create the database, complete these steps:

   - Use the database access client to connect to the database server as the `root` user:

     ```
     $ mysql -u root -p
     ```

   - Create the `heat` database:

     ```
     CREATE DATABASE heat;
     ```

   - Grant proper access to the `heat` database:

     ```
     GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost' \
       IDENTIFIED BY 'HEAT_DBPASS';
     GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%' \
       IDENTIFIED BY 'HEAT_DBPASS';
     ```

Replace HEAT_DBPASS with a suitable password.

- Exit the database access client.

2. Source the admin credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

3. To create the service credentials, complete these steps:

- Create the heat user:

```
$ openstack user create --domain default --password-prompt heat
User Password:
Repeat User Password:
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | e0353a670a9e496da891347c589539e9 |
| enabled   | True                             |
| id        | ca2e175b851943349be29a328cc5e360 |
| name      | heat                             |
+-----------+----------------------------------+
```

- Add the admin role to the heat user:

```
$ openstack role add --project service --user heat admin
```

> **Note**
>
> This command provides no output.

- Create the heat and heat-cfn service entities:

```
$ openstack service create --name heat \
  --description "Orchestration" orchestration
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Orchestration                    |
| enabled     | True                             |
| id          | 727841c6f5df4773baa4e8a5ae7d72eb |
| name        | heat                             |
| type        | orchestration                    |
+-------------+----------------------------------+

$ openstack service create --name heat-cfn \
  --description "Orchestration"  cloudformation
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Orchestration                    |
```

(continues on next page)

(continued from previous page)

```
| enabled     | True                                 |
| id          | c42cede91a4e47c3b10c8aedc8d890c6 |
| name        | heat-cfn                             |
| type        | cloudformation                       |
+-------------+--------------------------------------+
```

4. Create the Orchestration service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  orchestration public http://controller:8004/v1/%\(tenant_id\)s
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | 3f4dab34624e4be7b000265f25049609       |
| interface    | public                                 |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | 727841c6f5df4773baa4e8a5ae7d72eb       |
| service_name | heat                                   |
| service_type | orchestration                          |
| url          | http://controller:8004/v1/%(tenant_id)s |
+--------------+----------------------------------------+

$ openstack endpoint create --region RegionOne \
  orchestration internal http://controller:8004/v1/%\(tenant_id\)s
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | 9489f78e958e45cc85570fec7e836d98       |
| interface    | internal                               |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | 727841c6f5df4773baa4e8a5ae7d72eb       |
| service_name | heat                                   |
| service_type | orchestration                          |
| url          | http://controller:8004/v1/%(tenant_id)s |
+--------------+----------------------------------------+

$ openstack endpoint create --region RegionOne \
  orchestration admin http://controller:8004/v1/%\(tenant_id\)s
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | 76091559514b40c6b7b38dde790efe99       |
| interface    | admin                                  |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
```

(continues on next page)

(continued from previous page)

```
| service_id   | 727841c6f5df4773baa4e8a5ae7d72eb        |
| service_name | heat                                    |
| service_type | orchestration                           |
| url          | http://controller:8004/v1/%(tenant_id)s |
+--------------+-----------------------------------------+
```

```
$ openstack endpoint create --region RegionOne \
  cloudformation public http://controller:8000/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | b3ea082e019c4024842bf0a80555052c |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | c42cede91a4e47c3b10c8aedc8d890c6 |
| service_name | heat-cfn                         |
| service_type | cloudformation                   |
| url          | http://controller:8000/v1        |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  cloudformation internal http://controller:8000/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 169df4368cdc435b8b115a9cb084044e |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | c42cede91a4e47c3b10c8aedc8d890c6 |
| service_name | heat-cfn                         |
| service_type | cloudformation                   |
| url          | http://controller:8000/v1        |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  cloudformation admin http://controller:8000/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 3d3edcd61eb343c1bbd629aa041ff88b |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | c42cede91a4e47c3b10c8aedc8d890c6 |
```

(continues on next page)

(continued from previous page)

```
| service_name | heat-cfn                          |
| service_type | cloudformation                    |
| url          | http://controller:8000/v1         |
+--------------+-----------------------------------+
```

5. Orchestration requires additional information in the Identity service to manage stacks. To add this information, complete these steps:

   - Create the `heat` domain that contains projects and users for stacks:

```
$ openstack domain create --description "Stack projects and users"␣
↪heat
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Stack projects and users         |
| enabled     | True                             |
| id          | 0f4d1bd326f2454dacc72157ba328a47 |
| name        | heat                             |
+-------------+----------------------------------+
```

   - Create the `heat_domain_admin` user to manage projects and users in the `heat` domain:

```
$ openstack user create --domain heat --password-prompt heat_domain_
↪admin
User Password:
Repeat User Password:
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | 0f4d1bd326f2454dacc72157ba328a47 |
| enabled   | True                             |
| id        | b7bd1abfbcf64478b47a0f13cd4d970a |
| name      | heat_domain_admin                |
+-----------+----------------------------------+
```

   - Add the `admin` role to the `heat_domain_admin` user in the `heat` domain to enable administrative stack management privileges by the `heat_domain_admin` user:

```
$ openstack role add --domain heat --user-domain heat --user heat_
↪domain_admin admin
```

> **Note**
>
> This command provides no output.

   - Create the `heat_stack_owner` role:

```
$ openstack role create heat_stack_owner
+-----------+----------------------------------+
```

(continues on next page)

(continued from previous page)

```
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 15e34f0c4fed4e68b3246275883c8630 |
| name      | heat_stack_owner                 |
+-----------+----------------------------------+
```

- Add the `heat_stack_owner` role to the `demo` project and user to enable stack management by the `demo` user:

```
$ openstack role add --project demo --user demo heat_stack_owner
```

> **Note**
>
> This command provides no output.

> **Note**
>
> You must add the `heat_stack_owner` role to each user that manages stacks.

- Create the `heat_stack_user` role:

```
$ openstack role create heat_stack_user
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 88849d41a55d4d1d91e4f11bffd8fc5c |
| name      | heat_stack_user                  |
+-----------+----------------------------------+
```

> **Note**
>
> The Orchestration service automatically assigns the `heat_stack_user` role to users that it creates during stack deployment. By default, this role restricts API <Application Programming Interface (API)> operations. To avoid conflicts, do not add this role to users with the `heat_stack_owner` role.

Install and configure components

> **Note**
>
> Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages:

```
# apt-get install heat-api heat-api-cfn heat-engine
```

2. Edit the `/etc/heat/heat.conf` file and complete the following actions:

   - In the `[database]` section, configure database access:

```
[database]
...
connection = mysql+pymysql://heat:HEAT_DBPASS@controller/heat
```

     Replace `HEAT_DBPASS` with the password you chose for the Orchestration database.

   - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

```
[DEFAULT]
...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

     Replace `RABBIT_PASS` with the password you chose for the `openstack` account in `RabbitMQ`.

   - In the `[keystone_authtoken]`, `[ec2authtoken]`, `[trustee]` and `[clients_keystone]` sections, configure Identity service access:

```
[keystone_authtoken]
...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = heat
password = HEAT_PASS

[ec2authtoken]
...
auth_url = http://controller:5000
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = heat
password = HEAT_PASS

[trustee]
...
auth_type = password
auth_url = http://controller:5000
username = heat
```

(continues on next page)

(continued from previous page)

```
password = HEAT_PASS
user_domain_name = Default

[clients_keystone]
...
auth_uri = http://controller:5000
```

Replace HEAT_PASS with the password you chose for the heat user in the Identity service.

- In the [DEFAULT] section, configure the metadata and wait condition URLs:

```
[DEFAULT]
...
heat_metadata_server_url = http://controller:8000
heat_waitcondition_server_url = http://controller:8000/v1/
↪waitcondition
```

- In the [DEFAULT] section, configure the stack domain and administrative credentials:

```
[DEFAULT]
...
stack_domain_admin = heat_domain_admin
stack_domain_admin_password = HEAT_DOMAIN_PASS
stack_user_domain_name = heat
```

Replace HEAT_DOMAIN_PASS with the password you chose for the heat_domain_admin user in the Identity service.

3. Populate the Orchestration database:

```
# su -s /bin/sh -c "heat-manage db_sync" heat
```

Note

Ignore any deprecation messages in this output.

Finalize installation

1. Restart the Orchestration services:

```
# service heat-api restart
# service heat-api-cfn restart
# service heat-engine restart
```

Install and configure for Debian

This section describes how to install and configure the Orchestration service for Debian.

Prerequisites

Before you install and configure Orchestration, you must create a database, service credentials, and API endpoints. Orchestration also requires additional information in the Identity service.

1. To create the database, complete these steps:
   - Use the database access client to connect to the database server as the `root` user:

     ```
     $ mysql -u root -p
     ```

   - Create the `heat` database:

     ```
     CREATE DATABASE heat;
     ```

   - Grant proper access to the `heat` database:

     ```
     GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost' \
       IDENTIFIED BY 'HEAT_DBPASS';
     GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%' \
       IDENTIFIED BY 'HEAT_DBPASS';
     ```

     Replace `HEAT_DBPASS` with a suitable password.

   - Exit the database access client.
2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:
   - Create the `heat` user:

     ```
     $ openstack user create --domain default --password-prompt heat
     User Password:
     Repeat User Password:
     +-----------+----------------------------------+
     | Field     | Value                            |
     +-----------+----------------------------------+
     | domain_id | e0353a670a9e496da891347c589539e9 |
     | enabled   | True                             |
     | id        | ca2e175b851943349be29a328cc5e360 |
     | name      | heat                             |
     +-----------+----------------------------------+
     ```

   - Add the `admin` role to the `heat` user:

     ```
     $ openstack role add --project service --user heat admin
     ```

     > **Note**
     >
     > This command provides no output.

- Create the `heat` and `heat-cfn` service entities:

```
$ openstack service create --name heat \
  --description "Orchestration" orchestration
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Orchestration                    |
| enabled     | True                             |
| id          | 727841c6f5df4773baa4e8a5ae7d72eb |
| name        | heat                             |
| type        | orchestration                    |
+-------------+----------------------------------+

$ openstack service create --name heat-cfn \
  --description "Orchestration"  cloudformation
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Orchestration                    |
| enabled     | True                             |
| id          | c42cede91a4e47c3b10c8aedc8d890c6 |
| name        | heat-cfn                         |
| type        | cloudformation                   |
+-------------+----------------------------------+
```

4. Create the Orchestration service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  orchestration public http://controller:8004/v1/%\(tenant_id\)s
+--------------+-----------------------------------------+
| Field        | Value                                   |
+--------------+-----------------------------------------+
| enabled      | True                                    |
| id           | 3f4dab34624e4be7b000265f25049609        |
| interface    | public                                  |
| region       | RegionOne                               |
| region_id    | RegionOne                               |
| service_id   | 727841c6f5df4773baa4e8a5ae7d72eb        |
| service_name | heat                                    |
| service_type | orchestration                           |
| url          | http://controller:8004/v1/%(tenant_id)s |
+--------------+-----------------------------------------+

$ openstack endpoint create --region RegionOne \
  orchestration internal http://controller:8004/v1/%\(tenant_id\)s
+--------------+-----------------------------------------+
| Field        | Value                                   |
+--------------+-----------------------------------------+
| enabled      | True                                    |
| id           | 9489f78e958e45cc85570fec7e836d98        |
```

(continues on next page)

(continued from previous page)

```
| interface    | internal                                |
| region       | RegionOne                               |
| region_id    | RegionOne                               |
| service_id   | 727841c6f5df4773baa4e8a5ae7d72eb        |
| service_name | heat                                    |
| service_type | orchestration                           |
| url          | http://controller:8004/v1/%(tenant_id)s |
+--------------+-----------------------------------------+

$ openstack endpoint create --region RegionOne \
  orchestration admin http://controller:8004/v1/%\(tenant_id\)s
+--------------+-----------------------------------------+
| Field        | Value                                   |
+--------------+-----------------------------------------+
| enabled      | True                                    |
| id           | 76091559514b40c6b7b38dde790efe99        |
| interface    | admin                                   |
| region       | RegionOne                               |
| region_id    | RegionOne                               |
| service_id   | 727841c6f5df4773baa4e8a5ae7d72eb        |
| service_name | heat                                    |
| service_type | orchestration                           |
| url          | http://controller:8004/v1/%(tenant_id)s |
+--------------+-----------------------------------------+
```

```
$ openstack endpoint create --region RegionOne \
  cloudformation public http://controller:8000/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | b3ea082e019c4024842bf0a80555052c |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | c42cede91a4e47c3b10c8aedc8d890c6 |
| service_name | heat-cfn                         |
| service_type | cloudformation                   |
| url          | http://controller:8000/v1        |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  cloudformation internal http://controller:8000/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 169df4368cdc435b8b115a9cb084044e |
| interface    | internal                         |
```

(continues on next page)

(continued from previous page)

```
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | c42cede91a4e47c3b10c8aedc8d890c6 |
| service_name | heat-cfn                         |
| service_type | cloudformation                   |
| url          | http://controller:8000/v1        |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  cloudformation admin http://controller:8000/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 3d3edcd61eb343c1bbd629aa041ff88b |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | c42cede91a4e47c3b10c8aedc8d890c6 |
| service_name | heat-cfn                         |
| service_type | cloudformation                   |
| url          | http://controller:8000/v1        |
+--------------+----------------------------------+
```

5. Orchestration requires additional information in the Identity service to manage stacks. To add this information, complete these steps:

   - Create the `heat` domain that contains projects and users for stacks:

```
$ openstack domain create --description "Stack projects and users"␣
↪heat
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Stack projects and users         |
| enabled     | True                             |
| id          | 0f4d1bd326f2454dacc72157ba328a47 |
| name        | heat                             |
+-------------+----------------------------------+
```

   - Create the `heat_domain_admin` user to manage projects and users in the `heat` domain:

```
$ openstack user create --domain heat --password-prompt heat_domain_
↪admin
User Password:
Repeat User Password:
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | 0f4d1bd326f2454dacc72157ba328a47 |
```

(continues on next page)

(continued from previous page)

```
| enabled   | True                              |
| id        | b7bd1abfbcf64478b47a0f13cd4d970a |
| name      | heat_domain_admin                 |
+-----------+-----------------------------------+
```

- Add the `admin` role to the `heat_domain_admin` user in the `heat` domain to enable administrative stack management privileges by the `heat_domain_admin` user:

```
$ openstack role add --domain heat --user-domain heat --user heat_
↪domain_admin admin
```

> **Note**
>
> This command provides no output.

- Create the `heat_stack_owner` role:

```
$ openstack role create heat_stack_owner
+-----------+-----------------------------------+
| Field     | Value                             |
+-----------+-----------------------------------+
| domain_id | None                              |
| id        | 15e34f0c4fed4e68b3246275883c8630 |
| name      | heat_stack_owner                  |
+-----------+-----------------------------------+
```

- Add the `heat_stack_owner` role to the `demo` project and user to enable stack management by the `demo` user:

```
$ openstack role add --project demo --user demo heat_stack_owner
```

> **Note**
>
> This command provides no output.

> **Note**
>
> You must add the `heat_stack_owner` role to each user that manages stacks.

- Create the `heat_stack_user` role:

```
$ openstack role create heat_stack_user
+-----------+-----------------------------------+
| Field     | Value                             |
+-----------+-----------------------------------+
| domain_id | None                              |
| id        | 88849d41a55d4d1d91e4f11bffd8fc5c |
```

(continues on next page)

(continued from previous page)

```
| name       | heat_stack_user                  |
+------------+----------------------------------+
```

> **Note**
>
> The Orchestration service automatically assigns the `heat_stack_user` role to users that it creates during stack deployment. By default, this role restricts API <Application Programming Interface (API)> operations. To avoid conflicts, do not add this role to users with the `heat_stack_owner` role.

Install and configure components

> **Note**
>
> Default configuration files vary by distribution. You might need to add these sections and options rather than modifying existing sections and options. Also, an ellipsis (`...`) in the configuration snippets indicates potential default configuration options that you should retain.

1. Install the packages:

```
# apt-get install heat-api heat-api-cfn heat-engine
```

2. Edit the `/etc/heat/heat.conf` file and complete the following actions:

   - In the `[database]` section, configure database access:

```
[database]
...
connection = mysql+pymysql://heat:HEAT_DBPASS@controller/heat
```

     Replace `HEAT_DBPASS` with the password you chose for the Orchestration database.

   - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

```
[DEFAULT]
...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

     Replace `RABBIT_PASS` with the password you chose for the `openstack` account in `RabbitMQ`.

   - In the `[keystone_authtoken]`, `[trustee]` and `[clients_keystone]` sections, configure Identity service access:

```
[keystone_authtoken]
...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
```

(continues on next page)

(continued from previous page)

```
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = heat
password = HEAT_PASS

[trustee]
...
auth_type = password
auth_url = http://controller:5000
username = heat
password = HEAT_PASS
user_domain_name = Default

[clients_keystone]
...
auth_uri = http://controller:5000
```

Replace `HEAT_PASS` with the password you chose for the `heat` user in the Identity service.

- In the `[DEFAULT]` section, configure the metadata and wait condition URLs:

```
[DEFAULT]
...
heat_metadata_server_url = http://controller:8000
heat_waitcondition_server_url = http://controller:8000/v1/
↪waitcondition
```

- In the `[DEFAULT]` section, configure the stack domain and administrative credentials:

```
[DEFAULT]
...
stack_domain_admin = heat_domain_admin
stack_domain_admin_password = HEAT_DOMAIN_PASS
stack_user_domain_name = heat
```

Replace `HEAT_DOMAIN_PASS` with the password you chose for the `heat_domain_admin` user in the Identity service.

3. Populate the Orchestration database:

```
# su -s /bin/sh -c "heat-manage db_sync" heat
```

Note

Ignore any deprecation messages in this output.

Finalize installation

1. Restart the Orchestration services:

```
# service heat-api restart
# service heat-api-cfn restart
# service heat-engine restart
```

2.1.3 Verify operation

Verify operation of the Orchestration service.

> **Note**
>
> Perform these commands on the controller node.

1. Source the `admin` tenant credentials:

```
$ . admin-openrc
```

2. List service components to verify successful launch and registration of each process:

```
$ openstack orchestration service list
+------------+-------------+--------------------------------------+-------
↪-----+--------+----------------------------+--------+
| hostname   | binary      | engine_id                            | host ␣
↪     | topic  | updated_at                 | status |
+------------+-------------+--------------------------------------+-------
↪-----+--------+----------------------------+--------+
| controller | heat-engine | 3e85d1ab-a543-41aa-aa97-378c381fb958 |␣
↪controller | engine | 2015-10-13T14:16:06.000000 | up     |
| controller | heat-engine | 45dbdcf6-5660-4d5f-973a-c4fc819da678 |␣
↪controller | engine | 2015-10-13T14:16:06.000000 | up     |
| controller | heat-engine | 51162b63-ecb8-4c6c-98c6-993af899c4f7 |␣
↪controller | engine | 2015-10-13T14:16:06.000000 | up     |
| controller | heat-engine | 8d7edc6d-77a6-460d-bd2a-984d76954646 |␣
↪controller | engine | 2015-10-13T14:16:06.000000 | up     |
+------------+-------------+--------------------------------------+-------
↪-----+--------+----------------------------+--------+
```

> **Note**
>
> This output should indicate four `heat-engine` components (default to 4 or number of CPUs on the host, whichever is greater) on the controller node.

2.1.4 Launch an instance

In environments that include the Orchestration service, you can create a stack that launches an instance.

Create a template

The Orchestration service uses templates to describe stacks. To learn about the template language, see the *Template Guide*.

- Create the `demo-template.yml` file with the following content:

```
heat_template_version: 2015-10-15
description: Launch a basic instance with CirrOS image using the
             ``m1.tiny`` flavor, ``mykey`` key,  and one network.

parameters:
  NetID:
    type: string
    description: Network ID to use for the instance.

resources:
  server:
    type: OS::Nova::Server
    properties:
      image: cirros
      flavor: m1.tiny
      key_name: mykey
      networks:
      - network: { get_param: NetID }

outputs:
  instance_name:
    description: Name of the instance.
    value: { get_attr: [ server, name ] }
  instance_ip:
    description: IP address of the instance.
    value: { get_attr: [ server, first_address ] }
```

Create a stack

Create a stack using the `demo-template.yml` template.

1. Source the `demo` credentials to perform the following steps as a non-administrative project:

```
$ . demo-openrc
```

2. Determine available networks.

```
$ openstack network list
+--------------------------------------+-------------+--------------------
↪------------------+
| ID                                   | Name        | Subnets           ␣
↪                  |
+--------------------------------------+-------------+--------------------
↪------------------+
| 4716ddfe-6e60-40e7-b2a8-42e57bf3c31c | selfservice | 2112d5eb-f9d6-45fd-
↪906e-7cabd38b7c7c |
```

(continues on next page)

(continued from previous page)

```
| b5b6993c-ddf9-40e7-91d0-86806a42edb8 | provider    | 310911f6-acf0-4a47-
↪824e-3032916582ff |
+--------------------------------------+-------------+--------------------
↪------------------+
```

> **Note**
>
> This output may differ from your environment.

3. Set the `NET_ID` environment variable to reflect the ID of a network. For example, using the provider network:

```
$ export NET_ID=$(openstack network list | awk '/ provider / { print $2 }
↪')
```

4. Create a stack of one CirrOS instance on the provider network:

```
$ openstack stack create -t demo-template.yml --parameter "NetID=$NET_ID"␣
↪stack
+--------------------------------------+------------+--------------------
↪+---------------------+--------------+
| ID                                   | Stack Name | Stack Status      ␣
↪| Creation Time       | Updated Time |
+--------------------------------------+------------+--------------------
↪+---------------------+--------------+
| dbf46d1b-0b97-4d45-a0b3-9662a1eb6cf3 | stack      | CREATE_IN_PROGRESS␣
↪| 2015-10-13T15:27:20 | None         |
+--------------------------------------+------------+--------------------
↪+---------------------+--------------+
```

5. After a short time, verify successful creation of the stack:

```
$ openstack stack list
+--------------------------------------+------------+-----------------+---
↪------------------+--------------+
| ID                                   | Stack Name | Stack Status    |␣
↪Creation Time       | Updated Time |
+--------------------------------------+------------+-----------------+---
↪------------------+--------------+
| dbf46d1b-0b97-4d45-a0b3-9662a1eb6cf3 | stack      | CREATE_COMPLETE |␣
↪2015-10-13T15:27:20 | None         |
+--------------------------------------+------------+-----------------+---
↪------------------+--------------+
```

6. Show the name and IP address of the instance and compare with the output of the OpenStack client:

```
$ openstack stack output show --all stack
[
  {
```

(continues on next page)

(continued from previous page)

```
    "output_value": "stack-server-3nzfyfofu6d4",
    "description": "Name of the instance.",
    "output_key": "instance_name"
  },
  {
    "output_value": "10.4.31.106",
    "description": "IP address of the instance.",
    "output_key": "instance_ip"
  }
]
```

```
$ openstack server list
+--------------------------------------+---------------------------+------
↪--+----------------------------------+
| ID                                   | Name                      |␣
↪Status | Networks                       |
+--------------------------------------+---------------------------+------
↪--+----------------------------------+
| 0fc2af0c-ae79-4d22-8f36-9e860c257da5 | stack-server-3nzfyfofu6d4 |␣
↪ACTIVE | public=10.4.31.106             |
+--------------------------------------+---------------------------+------
↪--+----------------------------------+
```

7. Delete the stack.

```
$ openstack stack delete --yes stack
```

2.1.5 Next steps

Your OpenStack environment now includes the heat service.

To add more services, see the additional documentation on installing OpenStack.

To learn more about the heat service, read the *Heat documentation*.

The Orchestration service (heat) uses a *Heat Orchestration Template (HOT)* to create and manage cloud resources.

This chapter assumes a working setup of OpenStack following the OpenStack Installation Tutorial.

2.2 Running Heat API services in HTTP Server

Since the Liberty release Heat has packaged a set of wsgi script entrypoints that enables users to run api services with a real web server like Apache HTTP Server (httpd).

There are several patterns for deployment. This doc shows some common ways of deploying api services with httpd.

2.2.1 mod-wsgi

This deployment method is possible since Liberty release.

The httpd/files directory contains sample files that can be changed and copied to the appropriate location in your httpd server.

On Debian/Ubuntu systems it is:

```
/etc/apache2/sites-available/heat-api.conf
/etc/apache2/sites-available/heat-api-cfn.conf
```

On Red Hat based systems it is:

```
/etc/httpd/conf.d/uwsgi-heat-api.conf
/etc/httpd/conf.d/uwsgi-heat-api-cfn.conf
```

2.2.2 uwsgi

In this deployment method we use uwsgi as a web server bound to a random local port. Then we configure apache using mod_proxy to forward all incoming requests on the specified endpoint to that local web-server. This has the advantage of letting apache manage all inbound http connections, and uwsgi manage running the python code. It also means when we make changes to Heat api code or configuration, we dont need to restart all of apache (which may be running other services too) and just need to restart the local uwsgi daemons.

The httpd/files directory contains sample files for configuring httpd to run Heat api services under uwsgi in this configuration. To use the sample configs simply copy *uwsgi-heat-api.conf* and *uwsgi-heat-api-cfn.conf* to the appropriate location for your web server.

On Debian/Ubuntu systems it is:

```
/etc/apache2/sites-available/uwsgi-heat-api.conf
/etc/apache2/sites-available/uwsgi-heat-api-cfn.conf
```

On Red Hat based systems it is:

```
/etc/httpd/conf.d/uwsgi-heat-api.conf
/etc/httpd/conf.d/uwsgi-heat-api-cfn.conf
```

Enable mod_proxy by running `sudo a2enmod proxy`

Then on Ubuntu/Debian systems enable the site by creating a symlink from the file in `sites-available` to `sites-enabled`. (This is not required on Red Hat based systems):

```
ln -s /etc/apache2/sites-available/uwsgi-heat-api.conf /etc/apache2/sites-
↪enabled
ln -s /etc/apache2/sites-available/uwsgi-heat-api-cfn.conf /etc/apache2/sites-
↪enabled
```

Start or restart httpd to pick up the new configuration.

Now we need to configure and start the uwsgi service. Copy the following files to */etc/heat*:

```
heat-api-uwsgi.ini
heat-api-cfn-uwsgi.ini
```

Update the files to match your system configuration (for example, youll want to set the number of processes and threads).

Install uwsgi and start the heat-api server using uwsgi:

```
sudo pip install uwsgi
uwsgi --ini /etc/heat/heat-api-uwsgi.ini
uwsgi --ini /etc/heat/heat-api-cfn-uwsgi.ini
```

Note

In the sample configs some random ports are used, but this doesnt matter and is just a randomly selected number. This is not a contract on the port used for the local uwsgi daemon.

mod_proxy_uwsgi

Instead of running uwsgi as a webserver listening on a local port and then having Apache HTTP proxy all the incoming requests with mod_proxy, the normally recommended way of deploying uwsgi with Apache httpd is to use mod_proxy_uwsgi and set up a local socket file for uwsgi to listen on. Apache will send the requests using the uwsgi protocol over this local socket file.

The dsvm jobs in heat upstream gate uses this deployment method.

For more details on using mod_proxy_uwsgi see the official docs.

2.3 Configuring Heat

2.3.1 Configuration options for the Orchestration service

The following options can be set in the `/etc/heat/heat.conf` config file. A *sample configuration file* is also available.

DEFAULT

`log_options`

Type
boolean

Default
`True`

Mutable
This option can be changed without restarting.

Enables or disables logging values of all registered options when starting a service (at DEBUG level).

`graceful_shutdown_timeout`

Type
integer

Default
`60`

Mutable
This option can be changed without restarting.

Specify a timeout after which a gracefully shutdown server will exit. Zero value means endless wait.

`debug`

Type
boolean

Default
`False`

Mutable
This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

`log_config_append`

Type
string

Default
`<None>`

Mutable
This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 1: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

`log_date_format`

Type
string

Default
`%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

`log_file`

Type
string

Default
`<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 2: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type
string

Default
<None>

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 3: Deprecated Variations

Group	Name
DEFAULT	logdir

use_syslog

Type
boolean

Default
`False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type
boolean

Default
`False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type
string

Default
`LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type
boolean

Default
`False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

use_stderr

Type
boolean

Default
`False`

Log output to standard error. This option is ignored if log_config_append is set.

log_color

Type
boolean

Default
`False`

(Optional) Set the color key according to log levels. This option takes effect only when logging to stderr or stdout is used. This option is ignored if log_config_append is set.

log_rotate_interval

Type
integer

Default
`1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is set to interval.

log_rotate_interval_type

Type
string

Default
`days`

Valid Values
Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

max_logfile_count

Type
integer

Default
`30`

Maximum number of rotated log files.

`max_logfile_size_mb`

Type
integer

Default
`200`

Log file maximum size in MB. This option is ignored if log_rotation_type is not set to size.

`log_rotation_type`

Type
string

Default
`none`

Valid Values
interval, size, none

Log rotation type.

Possible values

interval
Rotate logs at predefined time intervals.

size
Rotate logs once they reach a predefined size.

none
Do not rotate log files.

`logging_context_format_string`

Type
string

Default
```
%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s
[%(global_request_id)s %(request_id)s %(user_identity)s]
%(instance)s%(message)s
```

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

`logging_default_format_string`

Type
string

Default
```
%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-]
%(instance)s%(message)s
```

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

logging_debug_format_suffix

Type
string

Default
`%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

logging_exception_prefix

Type
string

Default
`%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

logging_user_identity_format

Type
string

Default
`%(user)s %(project)s %(domain)s %(system_scope)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

default_log_levels

Type
list

Default
```
['amqp=WARN', 'boto=WARN', 'sqlalchemy=WARN', 'suds=INFO',
'oslo.messaging=INFO', 'oslo_messaging=INFO', 'iso8601=WARN',
'requests.packages.urllib3.connectionpool=WARN', 'urllib3.
connectionpool=WARN', 'websocket=WARN', 'requests.packages.
urllib3.util.retry=WARN', 'urllib3.util.retry=WARN',
'keystonemiddleware=WARN', 'routes.middleware=WARN',
'stevedore=WARN', 'taskflow=WARN', 'keystoneauth=WARN', 'oslo.
cache=INFO', 'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']
```

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

publish_errors

Type
boolean

Default
`False`

Enables or disables publication of error events.

`instance_format`

Type
string

Default
`"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

`instance_uuid_format`

Type
string

Default
`"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

`rate_limit_interval`

Type
integer

Default
`0`

Interval, number of seconds, of log rate limiting.

`rate_limit_burst`

Type
integer

Default
`0`

Maximum number of logged messages per rate_limit_interval.

`rate_limit_except_level`

Type
string

Default
`CRITICAL`

Valid Values
CRITICAL, ERROR, INFO, WARNING, DEBUG,

Log level name used by rate limiting. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

fatal_deprecations

Type
boolean

Default
`False`

Enables or disables fatal status of deprecations.

executor_thread_pool_size

Type
integer

Default
`64`

Size of executor thread pool when executor is threading or eventlet.

Table 4: Deprecated Variations

Group	Name
DEFAULT	rpc_thread_pool_size

rpc_response_timeout

Type
integer

Default
`60`

Seconds to wait for a response from a call.

transport_url

Type
string

Default
`rabbit://`

The network address and optional user credentials for connecting to the messaging backend, in URL format. The expected format is:

driver://[user:pass@]host:port[,[userN:passN@]hostN:portN]/virtual_host?query

Example: rabbit://rabbitmq:password@127.0.0.1:5672//

For full details on the fields in the URL see the documentation of oslo_messaging.TransportURL at https://docs.openstack.org/oslo.messaging/latest/reference/transport.html

control_exchange

Type
string

Default
`openstack`

The default exchange under which topics are scoped. May be overridden by an exchange name specified in the transport_url option.

rpc_ping_enabled

Type
boolean

Default
`False`

Add an endpoint to answer to ping calls. Endpoint is named oslo_rpc_server_ping

host

Type
string

Default
`<Hostname>`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Name of the engine node. This can be an opaque identifier. It is not necessarily a hostname, FQDN, or IP address.

plugin_dirs

Type
list

Default
`['/usr/lib64/heat', '/usr/lib/heat', '/usr/local/lib/heat', '/usr/local/lib64/heat']`

List of directories to search for plug-ins.

environment_dir

Type
string

Default
`/etc/heat/environment.d`

The directory to search for environment files.

template_dir

Type
string

Default
`/etc/heat/templates`

The directory to search for template files.

deferred_auth_method

Type
string

Default
`trusts`

Valid Values
password, trusts

Select deferred auth method, stored password or trusts.

> **Warning**
>
> This option is deprecated for removal since 9.0.0. Its value may be silently ignored in the future.
>
> **Reason**
> Stored password based deferred auth is broken when used with keystone v3 and is not supported.

`reauthentication_auth_method`

Type
string

Default
`''`

Valid Values
, trusts

Allow reauthentication on token expiry, such that long-running tasks may complete. Note this defeats the expiry of any provided user tokens.

`allow_trusts_redelegation`

Type
boolean

Default
`False`

Create trusts with redelegation enabled. This option is only used when reauthentication_auth_method is set to trusts. Note that enabling this option does have security implications as all trusts created by Heat will use both impersonation and redelegation enabled. Enable it only when there are other services that need to create trusts from tokens Heat uses to access them, examples are Aodh and Heat in another region when configured to use trusts too.

`trusts_delegated_roles`

Type
list

Default
`[]`

Subset of trustor roles to be delegated to heat. If left unset, all roles of a user will be delegated to heat when creating a stack.

`max_resources_per_stack`

Type
integer

Default
`1000`

Maximum resources allowed per top-level stack. -1 stands for unlimited.

`max_stacks_per_tenant`

Type
integer

Default
`512`

Maximum number of stacks any one tenant may have active at one time. -1 stands for unlimited.

`max_software_configs_per_tenant`

Type
integer

Default
`4096`

Maximum number of software configs any one tenant may have active at one time. -1 stands for unlimited.

`max_software_deployments_per_tenant`

Type
integer

Default
`4096`

Maximum number of software deployments any one tenant may have active at one time.-1 stands for unlimited.

`max_snapshots_per_stack`

Type
integer

Default
`32`

Maximum number of snapshot any one stack may have active at one time. -1 stands for unlimited.

`action_retry_limit`

Type
integer

Default
`5`

Number of times to retry to bring a resource to a non-error state. Set to 0 to disable retries.

`client_retry_limit`

Type
integer

Default
`2`

Number of times to retry when a client encounters an expected intermittent error. Set to 0 to disable retries.

`max_server_name_length`

Type
integer

Default
`53`

Maximum Value
53

Maximum length of a server name to be used in nova.

`max_interface_check_attempts`

Type
integer

Default
`10`

Minimum Value
1

Number of times to check whether an interface has been attached or detached.

`max_nova_api_microversion`

Type
string

Default
`<None>`

Maximum nova API version for client plugin. With this limitation, any nova feature supported with microversion number above max_nova_api_microversion will not be available.

`max_cinder_api_microversion`

Type
string

Default
`<None>`

Maximum cinder API version for client plugin. With this limitation, any cinder feature supported with microversion number above max_cinder_api_microversion will not be available.

`max_ironic_api_microversion`

Type
string

Default
<None>

Maximum ironic API version for client plugin. With this limitation, any ironic feature supported with microversion number above max_ironic_api_microversion will not be available.

`event_purge_batch_size`

Type
integer

Default
`200`

Minimum Value
1

Controls how many events will be pruned whenever a stacks events are purged. Set this lower to keep more events at the expense of more frequent purges.

`max_events_per_stack`

Type
integer

Default
`1000`

Rough number of maximum events that will be available per stack. Actual number of events can be a bit higher since purge checks take place randomly 200/event_purge_batch_size percent of the time. Older events are deleted when events are purged. Set to 0 for unlimited events per stack.

`stack_action_timeout`

Type
integer

Default
`3600`

Timeout in seconds for stack action (ie. create or update).

`error_wait_time`

Type
integer

Default
`240`

The amount of time in seconds after an error has occurred that tasks may continue to run before being cancelled.

`engine_life_check_timeout`

Type
integer

Default
`2`

RPC timeout for the engine liveness check that is used for stack locking.

`enable_stack_abandon`

Type
boolean

Default
`False`

Enable the preview Stack Abandon feature.

> **Warning**
>
> This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason**
> Stack abandon is not supported by convergence engine.

`enable_stack_adopt`

Type
boolean

Default
`False`

Enable the preview Stack Adopt feature.

> **Warning**
>
> This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason**
> Stack adopt is not supported by convergence engine.

`convergence_engine`

Type
boolean

Default
`True`

Enables engine with convergence architecture. All stacks with this option will be created using convergence engine.

> **Warning**
>
> This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason**
> Legacy engine has been deprecated.

observe_on_update

Type
boolean

Default
`False`

On update, enables heat to collect existing resource properties from reality and converge to updated template.

default_software_config_transport

Type
string

Default
`POLL_SERVER_CFN`

Valid Values
POLL_SERVER_CFN, POLL_SERVER_HEAT, POLL_TEMP_URL, ZAQAR_MESSAGE

Template default for how the server should receive the metadata required for software configuration.

Possible values

POLL_SERVER_CFN
Allow calls to the cfn API action DescribeStackResource authenticated with the provided keypair (requires enabled heat-api-cfn)

POLL_SERVER_HEAT
Allow calls to the Heat API resource-show using the provided keystone credentials (requires keystone v3 API, and configured stack_user_* config options)

POLL_TEMP_URL
Create and populate a Swift TempURL with metadata for polling (requires object-store endpoint which supports TempURL)

ZAQAR_MESSAGE
Create a dedicated zaqar queue and post the metadata for polling.

default_deployment_signal_transport

Type
string

Default
`CFN_SIGNAL`

Valid Values
CFN_SIGNAL, TEMP_URL_SIGNAL, HEAT_SIGNAL, ZAQAR_SIGNAL

Template default for how the server should signal to heat with the deployment output values.

Possible values

CFN_SIGNAL
Allow an HTTP POST to a CFN keypair signed URL (requires enabled heat-api-cfn)

TEMP_URL_SIGNAL
Create a Swift TempURL to be signed via HTTP PUT (requires object-store endpoint which supports TempURL)

HEAT_SIGNAL
Allow calls to the Heat API resource-signal using the provided keystone credentials

ZAQAR_SIGNAL
Create a dedicated zaqar queue to be signed using the provided keystone credentials

`default_user_data_format`

Type
string

Default
`HEAT_CFNTOOLS`

Valid Values
HEAT_CFNTOOLS, RAW, SOFTWARE_CONFIG

Template default for how the user_data should be formatted for the server.

Possible values

HEAT_CFNTOOLS
The user_data is bundled as part of the heat-cfntools cloud-init boot configuration data

RAW
The user_data is passed to Nova unmodified

SOFTWARE_CONFIG
The user_data is bundled as part of the software config data, and metadata is derived from any associated SoftwareDeployment Resources

`hidden_stack_tags`

Type
list

Default
`[]`

Stacks containing these tag names will be hidden. Multiple tags should be given in a comma-delimited list (eg. hidden_stack_tags=hide_me,me_too).

`stack_scheduler_hints`

Type
boolean

Default
`False`

When this feature is enabled, scheduler hints identifying the heat stack context of a server or volume resource are passed to the configured schedulers in nova and cinder, for creates done using heat resource types OS::Cinder::Volume, OS::Nova::Server, and AWS::EC2::Instance. heat_root_stack_id will be set to the id of the root stack of the resource, heat_stack_id will be set to the id of the resources parent stack, heat_stack_name will be set to the name of the resources parent stack, heat_path_in_stack will be set to a list of comma delimited strings of stackresourcename and stackname with list[0] being rootstackname, heat_resource_name will be set to the resources name, and heat_resource_uuid will be set to the resources orchestration id.

`encrypt_parameters_and_properties`

Type
boolean

Default
`False`

Encrypt template parameters that were marked as hidden and also all the resource properties before storing them in database.

`metadata_put_timeout`

Type
floating point

Default
`60`

Minimum Value
0

Timeout in seconds for metadata update for software deployment

`periodic_interval`

Type
integer

Default
`60`

Seconds between running periodic tasks.

`heat_metadata_server_url`

Type
URI

Default
`<None>`

URL of the Heat metadata server. NOTE: Setting this is only needed if you require instances to use a different endpoint than in the keystone catalog

heat_waitcondition_server_url

Type
URI

Default
<None>

URL of the Heat waitcondition server.

instance_connection_is_secure

Type
string

Default
0

Instance connection to CFN/CW API via https.

instance_connection_https_validate_certificates

Type
string

Default
1

Instance connection to CFN/CW API validate certs if SSL is used.

region_name_for_services

Type
string

Default
<None>

Default region name used to get services endpoints.

region_name_for_shared_services

Type
string

Default
<None>

Region name for shared services endpoints.

shared_services_types

Type
list

Default
['image', 'volumev3']

The shared services located in the other region.Needs region_name_for_shared_services option to be set for this to take effect.

heat_stack_user_role

Type
string

Default
`heat_stack_user`

Keystone role for heat template-defined users.

stack_user_domain_id

Type
string

Default
<None>

Keystone domain ID which contains heat template-defined users. If this option is set, stack_user_domain_name option will be ignored.

Table 5: Deprecated Variations

Group	Name
DEFAULT	stack_user_domain

stack_user_domain_name

Type
string

Default
<None>

Keystone domain name which contains heat template-defined users. If *stack_user_domain_id* option is set, this option is ignored.

stack_domain_admin

Type
string

Default
<None>

Keystone username, a user with roles sufficient to manage users and projects in the stack_user_domain.

stack_domain_admin_password

Type
string

Default
<None>

Keystone password for stack_domain_admin user.

`max_template_size`

Type
integer

Default
`524288`

Maximum raw byte size of any template.

`max_nested_stack_depth`

Type
integer

Default
`5`

Maximum depth allowed when using nested stacks.

`template_fetch_timeout`

Type
floating point

Default
`60`

Minimum Value
0

Timeout in seconds for template download.

`num_engine_workers`

Type
integer

Default
`<None>`

Minimum Value
1

Number of heat-engine processes to fork and run. Will default to either to 4 or number of CPUs on the host, whichever is greater.

`server_keystone_endpoint_type`

Type
string

Default
`''`

Valid Values
, public, internal, admin

If set, is used to control which authentication endpoint is used by user-controlled servers to make calls back to Heat. If unset www_authenticate_uri is used.

auth_encryption_key

Type
string

Default
`notgood but just long enough i t`

Key used to encrypt authentication info in the database. Length of this key must be 32 characters.

max_json_body_size

Type
integer

Default
`1048576`

Maximum raw byte size of JSON request body. Should be larger than max_template_size.

cloud_backend

Type
string

Default
`heat.engine.clients.OpenStackClients`

Fully qualified class name to use as a client backend.

keystone_backend

Type
string

Default
`heat.engine.clients.os.keystone.heat_keystoneclient.KsClientWrapper`

Fully qualified class name to use as a keystone backend.

default_notification_level

Type
string

Default
`INFO`

Default notification level for outgoing notifications.

default_publisher_id

Type
string

Default
`<None>`

Default publisher_id for outgoing notifications.

loadbalancer_template

Type
string

Default
<None>

Custom template for the built-in loadbalancer nested stack.

auth_password

multi_cloud

Type
boolean

Default
`False`

Allow orchestration of multiple clouds.

allowed_auth_uris

Type
list

Default
`[]`

Allowed keystone endpoints for auth_uri when multi_cloud is enabled. At least one endpoint needs to be specified.

cache

config_prefix

Type
string

Default
`cache.oslo`

Prefix for building the configuration dictionary for the cache region. This should not need to be changed unless there is another dogpile.cache region with the same configuration name.

expiration_time

Type
integer

Default
`600`

Minimum Value
1

Default TTL, in seconds, for any cached item in the dogpile.cache region. This applies to any cached method that doesnt have an explicit cache expiration time defined for it.

backend_expiration_time

Type
integer

Default
<None>

Minimum Value
1

Expiration time in cache backend to purge expired records automatically. This should be greater than expiration_time and all cache_time options

backend

Type
string

Default
`dogpile.cache.null`

Valid Values
oslo_cache.memcache_pool, oslo_cache.dict, oslo_cache.etcd3gw, dogpile.cache.pymemcache, dogpile.cache.memcached, dogpile.cache.pylibmc, dogpile.cache.bmemcached, dogpile.cache.dbm, dogpile.cache.redis, dogpile.cache.redis_sentinel, dogpile.cache.memory, dogpile.cache.memory_pickle, dogpile.cache.null

Cache backend module. For eventlet-based or environments with hundreds of threaded servers, Memcache with pooling (oslo_cache.memcache_pool) is recommended. For environments with less than 100 threaded servers, Memcached (dogpile.cache.memcached) or Redis (dogpile.cache.redis) is recommended. Test environments with a single instance of the server can use the dogpile.cache.memory backend.

backend_argument

Type
multi-valued

Default
`''`

Arguments supplied to the backend module. Specify this option once per argument to be passed to the dogpile.cache backend. Example format: <argname>:<value>.

proxies

Type
list

Default
`[]`

Proxy classes to import that will affect the way the dogpile.cache backend functions. See the dogpile.cache documentation on changing-backend-behavior.

enabled

Type
boolean

Default
`False`

Global toggle for caching.

`debug_cache_backend`

Type
boolean

Default
`False`

Extra debugging from the cache backend (cache keys, get/set/delete/etc calls). This is only really useful if you need to see the specific cache-backend get/set/delete calls with the keys/values. Typically this should be left set to false.

`memcache_servers`

Type
list

Default
`['localhost:11211']`

Memcache servers in the format of host:port. This is used by backends dependent on Memcached.If `dogpile.cache.memcached` or `oslo_cache.memcache_pool` is used and a given host refer to an IPv6 or a given domain refer to IPv6 then you should prefix the given address with the address family (`inet6`) (e.g `inet6:[::1]:11211`, `inet6:[fd12:3456:789a:1::1]:11211`, `inet6:[controller-0.internalapi]:11211`). If the address family is not given then these backends will use the default `inet` address family which corresponds to IPv4

`memcache_dead_retry`

Type
integer

Default
`300`

Number of seconds memcached server is considered dead before it is tried again. (dogpile.cache.memcache and oslo_cache.memcache_pool backends only).

`memcache_socket_timeout`

Type
floating point

Default
`1.0`

Timeout in seconds for every call to a server. (dogpile.cache.memcache and oslo_cache.memcache_pool backends only).

`memcache_pool_maxsize`

Type
integer

Default
`10`

Max total number of open connections to every memcached server. (oslo_cache.memcache_pool backend only).

`memcache_pool_unused_timeout`

Type
integer

Default
`60`

Number of seconds a connection to memcached is held unused in the pool before it is closed. (oslo_cache.memcache_pool backend only).

`memcache_pool_connection_get_timeout`

Type
integer

Default
`10`

Number of seconds that an operation will wait to get a memcache client connection.

`memcache_pool_flush_on_reconnect`

Type
boolean

Default
`False`

Global toggle if memcache will be flushed on reconnect. (oslo_cache.memcache_pool backend only).

`memcache_sasl_enabled`

Type
boolean

Default
`False`

Enable the SASL(Simple Authentication and SecurityLayer) if the SASL_enable is true, else disable.

`memcache_username`

Type
string

Default
`<None>`

the user name for the memcached which SASL enabled

memcache_password

Type
string

Default
<None>

the password for the memcached which SASL enabled

redis_server

Type
string

Default
`localhost:6379`

Redis server in the format of host:port

redis_db

Type
integer

Default
`0`

Minimum Value
0

Database id in Redis server

redis_username

Type
string

Default
<None>

the user name for redis

redis_password

Type
string

Default
<None>

the password for redis

redis_sentinels

Type
list

Default
`['localhost:26379']`

Redis sentinel servers in the format of host:port

redis_socket_timeout

Type
floating point

Default
`1.0`

Timeout in seconds for every call to a server. (dogpile.cache.redis and dogpile.cache.redis_sentinel backends only).

redis_sentinel_service_name

Type
string

Default
`mymaster`

Service name of the redis sentinel cluster.

tls_enabled

Type
boolean

Default
`False`

Global toggle for TLS usage when communicating with the caching servers. Currently supported by `dogpile.cache.bmemcache`, `dogpile.cache.pymemcache`, `oslo_cache.memcache_pool`, `dogpile.cache.redis` and `dogpile.cache.redis_sentinel`.

tls_cafile

Type
string

Default
`<None>`

Path to a file of concatenated CA certificates in PEM format necessary to establish the caching servers authenticity. If tls_enabled is False, this option is ignored.

tls_certfile

Type
string

Default
`<None>`

Path to a single file in PEM format containing the clients certificate as well as any number of CA certificates needed to establish the certificates authenticity. This file is only required when client side authentication is necessary. If tls_enabled is False, this option is ignored.

tls_keyfile

Type
string

Default
<None>

Path to a single file containing the clients private key in. Otherwise the private key will be taken from the file specified in tls_certfile. If tls_enabled is False, this option is ignored.

`tls_allowed_ciphers`

Type
string

Default
<None>

Set the available ciphers for sockets created with the TLS context. It should be a string in the OpenSSL cipher list format. If not specified, all OpenSSL enabled ciphers will be available. Currently supported by `dogpile.cache.bmemcache`, `dogpile.cache.pymemcache` and `oslo_cache.memcache_pool`.

`enable_socket_keepalive`

Type
boolean

Default
`False`

Global toggle for the socket keepalive of dogpiles pymemcache backend

`socket_keepalive_idle`

Type
integer

Default
1

Minimum Value
0

The time (in seconds) the connection needs to remain idle before TCP starts sending keepalive probes. Should be a positive integer most greater than zero.

`socket_keepalive_interval`

Type
integer

Default
1

Minimum Value
0

The time (in seconds) between individual keepalive probes. Should be a positive integer greater than zero.

`socket_keepalive_count`

Type
integer

Default
`1`

Minimum Value
0

The maximum number of keepalive probes TCP should send before dropping the connection. Should be a positive integer greater than zero.

`enable_retry_client`

Type
boolean

Default
`False`

Enable retry client mechanisms to handle failure. Those mechanisms can be used to wrap all kind of pymemcache clients. The wrapper allows you to define how many attempts to make and how long to wait between attempts.

`retry_attempts`

Type
integer

Default
`2`

Minimum Value
1

Number of times to attempt an action before failing.

`retry_delay`

Type
floating point

Default
`0`

Number of seconds to sleep between each attempt.

`hashclient_retry_attempts`

Type
integer

Default
`2`

Minimum Value
1

Amount of times a client should be tried before it is marked dead and removed from the pool in the HashClients internal mechanisms.

`hashclient_retry_timeout`

Type
floating point

Default
`1`

Time in seconds that should pass between retry attempts in the HashClients internal mechanisms.

Table 6: Deprecated Variations

Group	Name
cache	hashclient_retry_delay

`hashclient_dead_timeout`

Type
floating point

Default
`60`

Time in seconds before attempting to add a node back in the pool in the HashClients internal mechanisms.

Table 7: Deprecated Variations

Group	Name
cache	dead_timeout

`enforce_fips_mode`

Type
boolean

Default
`False`

Global toggle for enforcing the OpenSSL FIPS mode. This feature requires Python support. This is available in Python 3.9 in all environments and may have been backported to older Python versions on select environments. If the Python executable used does not support OpenSSL FIPS mode, an exception will be raised. Currently supported by `dogpile.cache.bmemcache`, `dogpile.cache.pymemcache` and `oslo_cache.memcache_pool`.

clients

`endpoint_type`

Type
string

Default
`publicURL`

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
`False`

If set, then the servers certificate will not be verified.

clients_aodh

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_barbican

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_cinder

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

http_log_debug

Type
boolean

Default
False

Allow clients debug log output.

clients_designate

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_glance

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_heat

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

url

Type
URI

Default
<None>

Optional heat url in format like http://127.0.0.1:8004/v1/%(tenant_id)s.

clients_keystone

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

auth_uri

Type
URI

Default
<None>

Unversioned keystone url in format like http://127.0.0.1:5000.

clients_magnum

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_manila

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_mistral

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_neutron

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_nova

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

http_log_debug

Type
boolean

Default
False

Allow clients debug log output.

clients_octavia

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_swift

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_trove

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_vitrage

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

clients_zaqar

endpoint_type

Type
string

Default
<None>

Type of endpoint in Identity service catalog to use for communication with the OpenStack service.

ca_file

Type
string

Default
<None>

Optional CA cert file to use in SSL connections.

cert_file

Type
string

Default
<None>

Optional PEM-formatted certificate chain file.

key_file

Type
string

Default
<None>

Optional PEM-formatted file that contains the private key.

insecure

Type
boolean

Default
<None>

If set, then the servers certificate will not be verified.

constraint_validation_cache

expiration_time

Type
integer

Default
60

TTL, in seconds, for any cached item in the dogpile.cache region used for caching of validation constraints.

caching

Type
boolean

Default
True

Toggle to enable/disable caching when Orchestration Engine validates property constraints of stack. During property validation with constraints Orchestration Engine caches requests to other OpenStack services. Please note that the global toggle for oslo.cache(enabled=True in [cache] group) must be enabled to use this feature.

cors

allowed_origin

Type
list

Default
<None>

Indicate whether this resource may be shared with the domain received in the requests origin header. Format: <protocol>://<host>[:<port>], no trailing slash. Example: https://horizon.example.com

allow_credentials

Type
boolean

Default
`True`

Indicate that the actual request can include user credentials

expose_headers

Type
list

Default
`['X-Auth-Token', 'X-Subject-Token', 'X-Service-Token', 'X-OpenStack-Request-ID']`

Indicate which headers are safe to expose to the API. Defaults to HTTP Simple Headers.

max_age

Type
integer

Default
`3600`

Maximum cache age of CORS preflight requests.

allow_methods

Type
list

Default
`['GET', 'PUT', 'POST', 'DELETE', 'PATCH']`

Indicate which methods can be used during the actual request.

allow_headers

Type
list

Default
`['X-Auth-Token', 'X-Identity-Status', 'X-Roles', 'X-Service-Catalog', 'X-User-Id', 'X-Tenant-Id', 'X-OpenStack-Request-ID']`

Indicate which header field names may be used during the actual request.

database

`sqlite_synchronous`

Type
boolean

Default
`True`

If True, SQLite uses synchronous mode.

`backend`

Type
string

Default
`sqlalchemy`

The back end to use for the database.

`connection`

Type
string

Default
`<None>`

The SQLAlchemy connection string to use to connect to the database.

`slave_connection`

Type
string

Default
`<None>`

The SQLAlchemy connection string to use to connect to the slave database.

`asyncio_connection`

Type
string

Default
`<None>`

The SQLAlchemy asyncio connection string to use to connect to the database.

asyncio_slave_connection

Type
string

Default
<None>

The SQLAlchemy asyncio connection string to use to connect to the slave database.

synchronous_reader

Type
boolean

Default
True

Whether or not to assume a reader context needs to guarantee it can read data committed by a writer assuming replication lag is present; defaults to True. When False, a reader context works the same as async_reader and will select the slave database if present. When using a galera cluster, this can be set to False only if you set mysql_wsrep_sync_wait to 1 (this will guarantee that the reader will wait until writesets are committed).Note that this may incur a performance degradation within the galera cluster. Note also that this parameter has no effect if you do not set any slave_connection.

mysql_sql_mode

Type
string

Default
TRADITIONAL

The SQL mode to be used for MySQL sessions. This option, including the default, overrides any server-set SQL mode. To use whatever SQL mode is set by the server configuration, set this to no value. Example: mysql_sql_mode=

mysql_wsrep_sync_wait

Type
integer

Default
<None>

For Galera only, configure wsrep_sync_wait causality checks on new connections. Default is None, meaning dont configure any setting.

connection_recycle_time

Type
integer

Default
3600

Connections which have been present in the connection pool longer than this number of seconds will be replaced with a new one the next time they are checked out from the pool.

`max_pool_size`

Type
integer

Default
`5`

Maximum number of SQL connections to keep open in a pool. Setting a value of 0 indicates no limit.

`max_retries`

Type
integer

Default
`10`

Maximum number of database connection retries during startup. Set to -1 to specify an infinite retry count.

`retry_interval`

Type
integer

Default
`10`

Interval between retries of opening a SQL connection.

`max_overflow`

Type
integer

Default
`50`

If set, use this value for max_overflow with SQLAlchemy.

`connection_debug`

Type
integer

Default
`0`

Minimum Value
0

Maximum Value
100

Verbosity of SQL debugging information: 0=None, 100=Everything.

`connection_trace`

Type
boolean

Default
`False`

Add Python stack traces to SQL as comment strings.

`pool_timeout`

Type
integer

Default
`<None>`

If set, use this value for pool_timeout with SQLAlchemy.

`use_db_reconnect`

Type
boolean

Default
`False`

Enable the experimental use of database reconnect on connection lost.

`db_retry_interval`

Type
integer

Default
`1`

Seconds between retries of a database transaction.

`db_inc_retry_interval`

Type
boolean

Default
`True`

If True, increases the interval between retries of a database operation up to db_max_retry_interval.

`db_max_retry_interval`

Type
integer

Default
`10`

If db_inc_retry_interval is set, the maximum seconds between retries of a database operation.

`db_max_retries`

Type
integer

Default
`20`

Maximum retries in case of connection error or deadlock error before error is raised. Set to -1 to specify an infinite retry count.

connection_parameters

Type
string

Default
''

Optional URL parameters to append onto the connection URL at connect time; specify as param1=value1&param2=value2&

ec2authtoken

auth_uri

Type
URI

Default
<None>

Authentication Endpoint URI.

> **Warning**
>
> This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason**
> Superseded by the endpoint_override option

multi_cloud

Type
boolean

Default
False

Allow orchestration of multiple clouds.

clouds

Type
list

Default
[]

A list of names of clouds when multicloud is enabled. At least one should be defined when multi_cloud is enabled. For each name there must be a section [ec2authtoken.<name>] with keystone auth settings.

allowed_auth_uris

Type
list

Default
`[]`

Allowed keystone endpoints for auth_uri when multi_cloud is enabled. At least one endpoint needs to be specified.

> **Warning**
>
> This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason**
> Superseded by the clouds option

`auth_type`

Type
unknown type

Default
`<None>`

Authentication type to load

Table 8: Deprecated Variations

Group	Name
ec2authtoken	auth_plugin

`auth_section`

Type
unknown type

Default
`<None>`

Config Section from which to load plugin specific options

`auth_url`

Type
unknown type

Default
`<None>`

Authentication URL

`system_scope`

Type
unknown type

Default
<None>

Scope for system operations

domain_id

Type
unknown type

Default
<None>

Domain ID to scope to

domain_name

Type
unknown type

Default
<None>

Domain name to scope to

project_id

Type
unknown type

Default
<None>

Project ID to scope to

project_name

Type
unknown type

Default
<None>

Project name to scope to

project_domain_id

Type
unknown type

Default
<None>

Domain ID containing project

project_domain_name

Type
unknown type

Default
<None>

Domain name containing project

trust_id

Type
unknown type

Default
<None>

ID of the trust to use as a trustee use

user_id

Type
unknown type

Default
<None>

Users user ID

username

Type
unknown type

Default
<None>

Users username

Table 9: Deprecated Variations

Group	Name
ec2authtoken	user-name
ec2authtoken	user_name

user_domain_id

Type
unknown type

Default
<None>

Users domain ID

user_domain_name

Type
unknown type

Default
<None>

Users domain name

password

Type
unknown type

Default
<None>

Users password

cafile

Type
string

Default
<None>

PEM encoded Certificate Authority to use when verifying HTTPs connections.

Table 10: Deprecated Variations

Group	Name
ec2authtoken	ca_file

certfile

Type
string

Default
<None>

PEM encoded client certificate cert file

Table 11: Deprecated Variations

Group	Name
ec2authtoken	cert_file

keyfile

Type
string

Default
<None>

PEM encoded client certificate key file

Table 12: Deprecated Variations

Group	Name
ec2authtoken	key_file

insecure

Type
boolean

Default
`False`

Verify HTTPS connections.

timeout

Type
integer

Default
`<None>`

Timeout value for http requests

collect_timing

Type
boolean

Default
`False`

Collect per-API call timing information.

split_loggers

Type
boolean

Default
`False`

Log requests to multiple loggers.

service_type

Type
string

Default
`<None>`

The default service_type for endpoint URL discovery.

service_name

Type
string

Default
`<None>`

The default service_name for endpoint URL discovery.

`valid_interfaces`

Type
list

Default
<None>

List of interfaces, in order of preference, for endpoint URL.

`region_name`

Type
string

Default
<None>

The default region_name for endpoint URL discovery.

`endpoint_override`

Type
string

Default
<None>

Always use this endpoint URL for requests for this client. NOTE: The unversioned endpoint should be specified here; to request a particular API version, use the *version*, *min-version*, and/or *max-version* options.

`version`

Type
string

Default
<None>

Minimum Major API version within a given Major API version for endpoint URL discovery. Mutually exclusive with min_version and max_version

`min_version`

Type
string

Default
<None>

The minimum major version of a given API, intended to be used as the lower bound of a range with max_version. Mutually exclusive with version. If min_version is given with no max_version it is as if max version is latest.

`max_version`

Type
string

Default
<None>

The maximum major version of a given API, intended to be used as the upper bound of a range with min_version. Mutually exclusive with version.

`connect_retries`

Type
integer

Default
<None>

The maximum number of retries that should be attempted for connection errors.

`connect_retry_delay`

Type
floating point

Default
<None>

Delay (in seconds) between two retries for connection errors. If not set, exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.

`status_code_retries`

Type
integer

Default
<None>

The maximum number of retries that should be attempted for retriable HTTP status codes.

`status_code_retry_delay`

Type
floating point

Default
<None>

Delay (in seconds) between two retries for retriable status codes. If not set, exponential retry starting with 0.5 seconds up to a maximum of 60 seconds is used.

`retriable_status_codes`

Type
list

Default
<None>

List of retriable HTTP status codes that should be retried. If not set default to [503]

`interface`

Type
string

Default
<None>

The default interface for endpoint URL discovery.

> **Warning**
>
> This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason**
> Using valid-interfaces is preferrable because it is capable of accepting a list of possible interfaces.

healthcheck

`detailed`

Type
boolean

Default
`False`

Show more detailed information as part of the response. Security note: Enabling this option may expose sensitive details about the service being monitored. Be sure to verify that it will not violate your security policies.

`backends`

Type
list

Default
`[]`

Additional backends that can perform health checks and report that information back as part of a request.

`allowed_source_ranges`

Type
list

Default
`[]`

A list of network addresses to limit source ip allowed to access healthcheck information. Any request from ip outside of these network addresses are ignored.

`ignore_proxied_requests`

Type
boolean

Default
`False`

Ignore requests with proxy headers.

disable_by_file_path

Type
string

Default
<None>

Check the presence of a file to determine if an application is running on a port. Used by DisableByFileHealthcheck plugin.

disable_by_file_paths

Type
list

Default
[]

Check the presence of a file based on a port to determine if an application is running on a port. Expects a port:path list of strings. Used by DisableByFilesPortsHealthcheck plugin.

enable_by_file_paths

Type
list

Default
[]

Check the presence of files. Used by EnableByFilesHealthcheck plugin.

keystone_authtoken

www_authenticate_uri

Type
string

Default
<None>

Complete public Identity API endpoint. This endpoint should not be an admin endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If youre using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint.

Table 13: Deprecated Variations

Group	Name
keystone_authtoken	auth_uri

auth_uri

Type
string

Default
<None>

Complete public Identity API endpoint. This endpoint should not be an admin endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If youre using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint. This option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

Warning

This option is deprecated for removal since Queens. Its value may be silently ignored in the future.

Reason
The auth_uri option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

`auth_version`

Type
string

Default
<None>

API version of the Identity API endpoint.

`interface`

Type
string

Default
`internal`

Interface to use for the Identity API endpoint. Valid values are public, internal (default) or admin.

`delay_auth_decision`

Type
boolean

Default
`False`

Do not handle authorization requests within the middleware, but delegate the authorization decision to downstream WSGI components.

`http_connect_timeout`

Type
integer

Default
<None>

Request timeout value for communicating with Identity API server.

http_request_max_retries

Type
integer

Default
3

How many times are we trying to reconnect when communicating with Identity API Server.

cache

Type
string

Default
<None>

Request environment key where the Swift cache object is stored. When auth_token middleware is deployed with a Swift cache, use this option to have the middleware share a caching backend with swift. Otherwise, use the `memcached_servers` option instead.

certfile

Type
string

Default
<None>

Required if identity server requires client certificate

keyfile

Type
string

Default
<None>

Required if identity server requires client certificate

cafile

Type
string

Default
<None>

A PEM encoded Certificate Authority to use when verifying HTTPs connections. Defaults to system CAs.

insecure

Type
boolean

Default
`False`

Verify HTTPS connections.

region_name

Type
string

Default
<None>

The region in which the identity server can be found.

memcached_servers

Type
list

Default
<None>

Optionally specify a list of memcached server(s) to use for caching. If left undefined, tokens will instead be cached in-process.

Table 14: Deprecated Variations

Group	Name
keystone_authtoken	memcache_servers

token_cache_time

Type
integer

Default
300

In order to prevent excessive effort spent validating tokens, the middleware caches previously-seen tokens for a configurable duration (in seconds). Set to -1 to disable caching completely.

memcache_security_strategy

Type
string

Default
None

Valid Values
None, MAC, ENCRYPT

(Optional) If defined, indicate whether token data should be authenticated or authenticated and encrypted. If MAC, token data is authenticated (with HMAC) in the cache. If ENCRYPT, token data is encrypted and authenticated in the cache. If the value is not one of these options or empty, auth_token will raise an exception on initialization.

memcache_secret_key

Type
string

Default
<None>

(Optional, mandatory if memcache_security_strategy is defined) This string is used for key derivation.

memcache_tls_enabled

Type
boolean

Default
False

(Optional) Global toggle for TLS usage when comunicating with the caching servers.

memcache_tls_cafile

Type
string

Default
<None>

(Optional) Path to a file of concatenated CA certificates in PEM format necessary to establish the caching servers authenticity. If tls_enabled is False, this option is ignored.

memcache_tls_certfile

Type
string

Default
<None>

(Optional) Path to a single file in PEM format containing the clients certificate as well as any number of CA certificates needed to establish the certificates authenticity. This file is only required when client side authentication is necessary. If tls_enabled is False, this option is ignored.

memcache_tls_keyfile

Type
string

Default
<None>

(Optional) Path to a single file containing the clients private key in. Otherwhise the private key will be taken from the file specified in tls_certfile. If tls_enabled is False, this option is ignored.

memcache_tls_allowed_ciphers

Type
string

Default
<None>

(Optional) Set the available ciphers for sockets created with the TLS context. It should be a string in the OpenSSL cipher list format. If not specified, all OpenSSL enabled ciphers will be available.

memcache_pool_dead_retry

Type
integer

Default
300

(Optional) Number of seconds memcached server is considered dead before it is tried again.

memcache_pool_maxsize

Type
integer

Default
10

(Optional) Maximum total number of open connections to every memcached server.

memcache_pool_socket_timeout

Type
integer

Default
3

(Optional) Socket timeout in seconds for communicating with a memcached server.

memcache_pool_unused_timeout

Type
integer

Default
60

(Optional) Number of seconds a connection to memcached is held unused in the pool before it is closed.

memcache_pool_conn_get_timeout

Type
integer

Default
10

(Optional) Number of seconds that an operation will wait to get a memcached client connection from the pool.

memcache_use_advanced_pool

Type
boolean

Default
True

(Optional) Use the advanced (eventlet safe) memcached client pool.

`include_service_catalog`

Type
boolean

Default
`True`

(Optional) Indicate whether to set the X-Service-Catalog header. If False, middleware will not ask for service catalog on token validation and will not set the X-Service-Catalog header.

`enforce_token_bind`

Type
string

Default
`permissive`

Used to control the use and type of token binding. Can be set to: disabled to not check token binding. permissive (default) to validate binding information if the bind type is of a form known to the server and ignore it if not. strict like permissive but if the bind type is unknown the token will be rejected. required any form of token binding is needed to be allowed. Finally the name of a binding method that must be present in tokens.

`service_token_roles`

Type
list

Default
`['service']`

A choice of roles that must be present in a service token. Service tokens are allowed to request that an expired token can be used and so this check should tightly control that only actual services should be sending this token. Roles here are applied as an ANY check so any role in this list must be present. For backwards compatibility reasons this currently only affects the allow_expired check.

`service_token_roles_required`

Type
boolean

Default
`False`

For backwards compatibility reasons we must let valid service tokens pass that dont pass the service_token_roles check as valid. Setting this true will become the default in a future release and should be enabled if possible.

`service_type`

Type
string

Default
`<None>`

The name or type of the service as it appears in the service catalog. This is used to validate tokens that have restricted access rules.

memcache_sasl_enabled

Type
boolean

Default
`False`

Enable the SASL(Simple Authentication and Security Layer) if the SASL_enable is true, else disable.

memcache_username

Type
string

Default
`''`

the user name for the SASL

memcache_password

Type
string

Default
`''`

the username password for SASL

auth_type

Type
unknown type

Default
`<None>`

Authentication type to load

Table 15: Deprecated Variations

Group	Name
keystone_authtoken	auth_plugin

auth_section

Type
unknown type

Default
`<None>`

Config Section from which to load plugin specific options

noauth

token_response

Type
string

Default
''

JSON file containing the content returned by the noauth middleware.

oslo_messaging_kafka

kafka_max_fetch_bytes

Type
integer

Default
1048576

Max fetch bytes of Kafka consumer

kafka_consumer_timeout

Type
floating point

Default
1.0

Default timeout(s) for Kafka consumers

consumer_group

Type
string

Default
oslo_messaging_consumer

Group id for Kafka consumer. Consumers in one group will coordinate message consumption

producer_batch_timeout

Type
floating point

Default
0.0

Upper bound on the delay for KafkaProducer batching in seconds

producer_batch_size

Type
integer

Default
16384

Size of batch for the producer async send

compression_codec

Type
string

Default
`none`

Valid Values
none, gzip, snappy, lz4, zstd

The compression codec for all data generated by the producer. If not set, compression will not be used. Note that the allowed values of this depend on the kafka version

enable_auto_commit

Type
boolean

Default
`False`

Enable asynchronous consumer commits

max_poll_records

Type
integer

Default
`500`

The maximum number of records returned in a poll call

security_protocol

Type
string

Default
`PLAINTEXT`

Valid Values
PLAINTEXT, SASL_PLAINTEXT, SSL, SASL_SSL

Protocol used to communicate with brokers

sasl_mechanism

Type
string

Default
`PLAIN`

Mechanism when security protocol is SASL

ssl_cafile

Type
string

Default
''

CA certificate PEM file used to verify the server certificate

ssl_client_cert_file

Type
string

Default
''

Client certificate PEM file used for authentication.

ssl_client_key_file

Type
string

Default
''

Client key PEM file used for authentication.

ssl_client_key_password

Type
string

Default
''

Client key password file used for authentication.

oslo_messaging_notifications

driver

Type
multi-valued

Default
''

The Drivers(s) to handle sending notifications. Possible values are messaging, messagingv2, routing, log, test, noop

transport_url

Type
string

Default
<None>

A URL representing the messaging driver to use for notifications. If not set, we fall back to the same configuration used for RPC.

topics

Type
list

Default
`['notifications']`

AMQP topic used for OpenStack notifications.

retry

Type
integer

Default
`-1`

The maximum number of attempts to re-send a notification message which failed to be delivered due to a recoverable error. 0 - No retry, -1 - indefinite

oslo_messaging_rabbit

amqp_durable_queues

Type
boolean

Default
`False`

Use durable queues in AMQP. If rabbit_quorum_queue is enabled, queues will be durable and this value will be ignored.

amqp_auto_delete

Type
boolean

Default
`False`

Auto-delete queues in AMQP.

rpc_conn_pool_size

Type
integer

Default
`30`

Minimum Value
1

Size of RPC connection pool.

conn_pool_min_size

Type
integer

Default
`2`

The pool size limit for connections expiration policy

conn_pool_ttl

Type
integer

Default
`1200`

The time-to-live in sec of idle connections in the pool

ssl

Type
boolean

Default
`False`

Connect over SSL.

ssl_version

Type
string

Default
`''`

SSL version to use (valid only if SSL enabled). Valid values are TLSv1 and SSLv23. SSLv2, SSLv3, TLSv1_1, and TLSv1_2 may be available on some distributions.

ssl_key_file

Type
string

Default
`''`

SSL key file (valid only if SSL enabled).

ssl_cert_file

Type
string

Default
`''`

SSL cert file (valid only if SSL enabled).

ssl_ca_file

Type
string

Default
`''`

SSL certification authority file (valid only if SSL enabled).

ssl_enforce_fips_mode

Type
boolean

Default
`False`

Global toggle for enforcing the OpenSSL FIPS mode. This feature requires Python support. This is available in Python 3.9 in all environments and may have been backported to older Python versions on select environments. If the Python executable used does not support OpenSSL FIPS mode, an exception will be raised.

heartbeat_in_pthread

Type
boolean

Default
`False`

(DEPRECATED) It is recommend not to use this option anymore. Run the health check heartbeat thread through a native python thread by default. If this option is equal to False then the health check heartbeat will inherit the execution model from the parent process. For example if the parent process has monkey patched the stdlib by using eventlet/greenlet then the heartbeat will be run through a green thread. This option should be set to True only for the wsgi services.

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
The option is related to Eventlet which will be removed. In addition this has never worked as expected with services using eventlet for core service framework.

kombu_reconnect_delay

Type
floating point

Default
`1.0`

Minimum Value
0.0

Maximum Value
4.5

How long to wait (in seconds) before reconnecting in response to an AMQP consumer cancel notification.

kombu_reconnect_splay

Type
floating point

Default
`0.0`

Minimum Value
0.0

Random time to wait for when reconnecting in response to an AMQP consumer cancel notification.

kombu_compression

Type
string

Default
`<None>`

EXPERIMENTAL: Possible values are: gzip, bz2. If not set compression will not be used. This option may not be available in future versions.

kombu_missing_consumer_retry_timeout

Type
integer

Default
`60`

How long to wait a missing client before abandoning to send it its replies. This value should not be longer than rpc_response_timeout.

Table 16: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_reconnect_timeout

kombu_failover_strategy

Type
string

Default
`round-robin`

Valid Values
round-robin, shuffle

Determines how the next RabbitMQ node is chosen in case the one we are currently connected to becomes unavailable. Takes effect only if more than one RabbitMQ node is provided in config.

`rabbit_login_method`

Type
string

Default
`AMQPLAIN`

Valid Values
PLAIN, AMQPLAIN, EXTERNAL, RABBIT-CR-DEMO

The RabbitMQ login method.

`rabbit_retry_interval`

Type
integer

Default
`1`

Minimum Value
1

How frequently to retry connecting with RabbitMQ.

`rabbit_retry_backoff`

Type
integer

Default
`2`

Minimum Value
0

How long to backoff for between retries when connecting to RabbitMQ.

`rabbit_interval_max`

Type
integer

Default
`30`

Minimum Value
1

Maximum interval of RabbitMQ connection retries.

`rabbit_ha_queues`

Type
boolean

Default
`False`

Try to use HA queues in RabbitMQ (x-ha-policy: all). If you change this option, you must wipe the RabbitMQ database. In RabbitMQ 3.0, queue mirroring is no longer controlled by the x-ha-policy argument when declaring a queue. If you just want to make sure that all queues (except those with auto-generated names) are mirrored across all nodes, run: rabbitmqctl set_policy HA ^(?!amq.).* {ha-mode: all}

`rabbit_quorum_queue`

Type
boolean

Default
`False`

Use quorum queues in RabbitMQ (x-queue-type: quorum). The quorum queue is a modern queue type for RabbitMQ implementing a durable, replicated FIFO queue based on the Raft consensus algorithm. It is available as of RabbitMQ 3.8.0. If set this option will conflict with the HA queues (`rabbit_ha_queues`) aka mirrored queues, in other words the HA queues should be disabled. Quorum queues are also durable by default so the amqp_durable_queues option is ignored when this option is enabled.

`rabbit_transient_quorum_queue`

Type
boolean

Default
`False`

Use quorum queues for transients queues in RabbitMQ. Enabling this option will then make sure those queues are also using quorum kind of rabbit queues, which are HA by default.

`rabbit_quorum_delivery_limit`

Type
integer

Default
`0`

Each time a message is redelivered to a consumer, a counter is incremented. Once the redelivery count exceeds the delivery limit the message gets dropped or dead-lettered (if a DLX exchange has been configured) Used only when rabbit_quorum_queue is enabled, Default 0 which means dont set a limit.

`rabbit_quorum_max_memory_length`

Type
integer

Default
`0`

By default all messages are maintained in memory if a quorum queue grows in length it can put memory pressure on a cluster. This option can limit the number of messages in the quorum queue. Used only when rabbit_quorum_queue is enabled, Default 0 which means dont set a limit.

rabbit_quorum_max_memory_bytes

Type
integer

Default
`0`

By default all messages are maintained in memory if a quorum queue grows in length it can put memory pressure on a cluster. This option can limit the number of memory bytes used by the quorum queue. Used only when rabbit_quorum_queue is enabled, Default 0 which means dont set a limit.

rabbit_transient_queues_ttl

Type
integer

Default
`1800`

Minimum Value
0

Positive integer representing duration in seconds for queue TTL (x-expires). Queues which are unused for the duration of the TTL are automatically deleted. The parameter affects only reply and fanout queues. Setting 0 as value will disable the x-expires. If doing so, make sure you have a rabbitmq policy to delete the queues or you deployment will create an infinite number of queue over time.In case rabbit_stream_fanout is set to True, this option will control data retention policy (x-max-age) for messages in the fanout queue rather then the queue duration itself. So the oldest data in the stream queue will be discarded from it once reaching TTL Setting to 0 will disable x-max-age for stream which make stream grow indefinitely filling up the diskspace

rabbit_qos_prefetch_count

Type
integer

Default
`0`

Specifies the number of messages to prefetch. Setting to zero allows unlimited messages.

heartbeat_timeout_threshold

Type
integer

Default
`60`

Number of seconds after which the Rabbit broker is considered down if heartbeats keep-alive fails (0 disables heartbeat).

heartbeat_rate

Type
integer

Default
`3`

How often times during the heartbeat_timeout_threshold we check the heartbeat.

`direct_mandatory_flag`

Type
boolean

Default
`True`

(DEPRECATED) Enable/Disable the RabbitMQ mandatory flag for direct send. The direct send is used as reply, so the MessageUndeliverable exception is raised in case the client queue does not exist.MessageUndeliverable exception will be used to loop for a timeout to lets a chance to sender to recover.This flag is deprecated and it will not be possible to deactivate this functionality anymore

Warning

This option is deprecated for removal. Its value may be silently ignored in the future.

Reason
Mandatory flag no longer deactivable.

`enable_cancel_on_failover`

Type
boolean

Default
`False`

Enable x-cancel-on-ha-failover flag so that rabbitmq server will cancel and notify consumerswhen queue is down

`use_queue_manager`

Type
boolean

Default
`False`

Should we use consistant queue names or random ones

`hostname`

Type
string

Default
`node1.example.com`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Hostname used by queue manager. Defaults to the value returned by socket.gethostname().

processname

Type
string

Default
`nova-api`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

Process name used by queue manager

rabbit_stream_fanout

Type
boolean

Default
`False`

Use stream queues in RabbitMQ (x-queue-type: stream). Streams are a new persistent and replicated data structure (queue type) in RabbitMQ which models an append-only log with non-destructive consumer semantics. It is available as of RabbitMQ 3.9.0. If set this option will replace all fanout queues with only one stream queue.

oslo_middleware

max_request_body_size

Type
integer

Default
`114688`

The maximum body size for each request, in bytes.

enable_proxy_headers_parsing

Type
boolean

Default
`False`

Whether the application is behind a proxy or not. This determines if the middleware should parse the headers or not.

http_basic_auth_user_file

Type
string

Default
`/etc/htpasswd`

HTTP basic auth password file.

oslo_policy

enforce_scope

Type
boolean

Default
`True`

This option controls whether or not to enforce scope when evaluating policies. If `True`, the scope of the token used in the request is compared to the `scope_types` of the policy being enforced. If the scopes do not match, an `InvalidScope` exception will be raised. If `False`, a message will be logged informing operators that policies are being invoked with mismatching scope.

> **Warning**
>
> This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason**
> This configuration was added temporarily to facilitate a smooth transition to the new RBAC. OpenStack will always enforce scope checks. This configuration option is deprecated and will be removed in the 2025.2 cycle.

enforce_new_defaults

Type
boolean

Default
`True`

This option controls whether or not to use old deprecated defaults when evaluating policies. If `True`, the old deprecated defaults are not going to be evaluated. This means if any existing token is allowed for old defaults but is disallowed for new defaults, it will be disallowed. It is encouraged to enable this flag along with the `enforce_scope` flag so that you can get the benefits of new defaults and `scope_type` together. If `False`, the deprecated policy check string is logically ORd with the new policy check string, allowing for a graceful upgrade experience between releases with new policies, which is the default behavior.

policy_file

Type
string

Default
`policy.yaml`

The relative or absolute path of a file that maps roles to permissions for a given service. Relative paths must be specified in relation to the configuration file setting this option.

policy_default_rule

Type
string

Default
`default`

Default rule. Enforced when a requested rule is not found.

policy_dirs

Type
multi-valued

Default
`policy.d`

Directories where policy configuration files are stored. They can be relative to any directory in the search path defined by the config_dir option, or absolute paths. The file defined by policy_file must exist for these directories to be searched. Missing or empty directories are ignored.

remote_content_type

Type
string

Default
`application/x-www-form-urlencoded`

Valid Values
application/x-www-form-urlencoded, application/json

Content Type to send and receive data for REST based policy check

remote_ssl_verify_server_crt

Type
boolean

Default
`False`

server identity verification for REST based policy check

remote_ssl_ca_crt_file

Type
string

Default
`<None>`

Absolute path to ca cert file for REST based policy check

remote_ssl_client_crt_file

Type
string

Default
`<None>`

Absolute path to client cert for REST based policy check

remote_ssl_client_key_file

Type
string

Default
<None>

Absolute path client key file REST based policy check

remote_timeout

Type
floating point

Default
60

Minimum Value
0

Timeout in seconds for REST based policy check

oslo_reports

log_dir

Type
string

Default
<None>

Path to a log directory where to create a file

file_event_handler

Type
string

Default
<None>

The path to a file to watch for changes to trigger the reports, instead of signals. Setting this option disables the signal trigger for the reports. If application is running as a WSGI application it is recommended to use this instead of signals.

file_event_handler_interval

Type
integer

Default
1

How many seconds to wait between polls when file_event_handler is set

oslo_versionedobjects

fatal_exception_format_errors

Type
boolean

Default
`False`

Make exception message format errors fatal

paste_deploy

flavor

Type
string

Default
`<None>`

The flavor to use.

api_paste_config

Type
string

Default
`api-paste.ini`

The API paste config file to use.

profiler

enabled

Type
boolean

Default
`False`

Enable the profiling for all services on this node.

Default value is False (fully disable the profiling feature).

Possible values:

- True: Enables the feature
- False: Disables the feature. The profiling cannot be started via this project operations. If the profiling is triggered by another project, this project part will be empty.

Table 17: Deprecated Variations

Group	Name
profiler	profiler_enabled

trace_sqlalchemy

Type
boolean

Default
`False`

Enable SQL requests profiling in services.

Default value is False (SQL requests wont be traced).

Possible values:

- True: Enables SQL requests profiling. Each SQL query will be part of the trace and can the be analyzed by how much time was spent for that.
- False: Disables SQL requests profiling. The spent time is only shown on a higher level of operations. Single SQL queries cannot be analyzed this way.

trace_requests

Type
boolean

Default
`False`

Enable python requests package profiling.

Supported drivers: jaeger+otlp

Default value is False.

Possible values:

- True: Enables requests profiling.
- False: Disables requests profiling.

hmac_keys

Type
string

Default
`SECRET_KEY`

Secret key(s) to use for encrypting context data for performance profiling.

This string value should have the following format: <key1>[,<key2>,<keyn>], where each key is some random string. A user who triggers the profiling via the REST API has to set one of these keys in the headers of the REST API call to include profiling results of this node for this particular project.

Both enabled flag and hmac_keys config options should be set to enable profiling. Also, to generate correct profiling information across all services at least one key needs to be consistent between OpenStack projects. This ensures it can be used from client side to generate the trace, containing information from all possible resources.

connection_string

Type
string

Default
`messaging://`

Connection string for a notifier backend.

Default value is `messaging://` which sets the notifier to oslo_messaging.

Examples of possible values:

- `messaging://` - use oslo_messaging driver for sending spans.
- `redis://127.0.0.1:6379` - use redis driver for sending spans.
- `mongodb://127.0.0.1:27017` - use mongodb driver for sending spans.
- `elasticsearch://127.0.0.1:9200` - use elasticsearch driver for sending spans.
- `jaeger://127.0.0.1:6831` - use jaeger tracing as driver for sending spans.

es_doc_type

Type
string

Default
`notification`

Document type for notification indexing in elasticsearch.

es_scroll_time

Type
string

Default
`2m`

This parameter is a time value parameter (for example: es_scroll_time=2m), indicating for how long the nodes that participate in the search will maintain relevant resources in order to continue and support it.

es_scroll_size

Type
integer

Default
`10000`

Elasticsearch splits large requests in batches. This parameter defines maximum size of each batch (for example: es_scroll_size=10000).

socket_timeout

Type
floating point

Default
`0.1`

Redissentinel provides a timeout option on the connections. This parameter defines that timeout (for example: socket_timeout=0.1).

`sentinel_service_name`

Type
string

Default
`mymaster`

Redissentinel uses a service name to identify a master redis service. This parameter defines the name (for example: `sentinal_service_name=mymaster`).

`filter_error_trace`

Type
boolean

Default
`False`

Enable filter traces that contain error/exception to a separated place.

Default value is set to False.

Possible values:

- True: Enable filter traces that contain error/exception.
- False: Disable the filter.

profiler_jaeger

`service_name_prefix`

Type
string

Default
`<None>`

Set service name prefix to Jaeger service name.

`process_tags`

Type
dict

Default
`{}`

Set process tracer tags.

profiler_otlp

service_name_prefix

Type
string

Default
<None>

Set service name prefix to OTLP exporters.

resource_finder_cache

expiration_time

Type
integer

Default
3600

TTL, in seconds, for any cached item in the dogpile.cache region used for caching of OpenStack service finder functions.

caching

Type
boolean

Default
True

Toggle to enable/disable caching when Orchestration Engine looks for other OpenStack service resources using name or id. Please note that the global toggle for oslo.cache(enabled=True in [cache] group) must be enabled to use this feature.

revision

heat_revision

Type
string

Default
unknown

Heat build revision. If you would prefer to manage your build revision separately, you can move this section to a different file and add it as another config option.

service_extension_cache

expiration_time

Type
integer

Default
3600

TTL, in seconds, for any cached item in the dogpile.cache region used for caching of service extensions.

caching

Type
boolean

Default
`True`

Toggle to enable/disable caching when Orchestration Engine retrieves extensions from other OpenStack services. Please note that the global toggle for oslo.cache(enabled=True in [cache] group) must be enabled to use this feature.

trustee

auth_type

Type
unknown type

Default
`<None>`

Authentication type to load

Table 18: Deprecated Variations

Group	Name
trustee	auth_plugin

auth_section

Type
unknown type

Default
`<None>`

Config Section from which to load plugin specific options

auth_url

Type
unknown type

Default
`<None>`

Authentication URL

system_scope

Type
unknown type

Default
`<None>`

Scope for system operations

domain_id

Type
unknown type

Default
<None>

Domain ID to scope to

domain_name

Type
unknown type

Default
<None>

Domain name to scope to

project_id

Type
unknown type

Default
<None>

Project ID to scope to

Table 19: Deprecated Variations

Group	Name
trustee	tenant-id
trustee	tenant_id

project_name

Type
unknown type

Default
<None>

Project name to scope to

Table 20: Deprecated Variations

Group	Name
trustee	tenant-name
trustee	tenant_name

project_domain_id

Type
unknown type

Default
<None>

Domain ID containing project

`project_domain_name`

Type
unknown type

Default
<None>

Domain name containing project

`trust_id`

Type
unknown type

Default
<None>

ID of the trust to use as a trustee use

`default_domain_id`

Type
unknown type

Default
<None>

Optional domain ID to use with v3 and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

`default_domain_name`

Type
unknown type

Default
<None>

Optional domain name to use with v3 API and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

`user_id`

Type
unknown type

Default
<None>

User id

username

Type
unknown type

Default
<None>

Username

Table 21: Deprecated Variations

Group	Name
trustee	user-name
trustee	user_name

user_domain_id

Type
unknown type

Default
<None>

Users domain id

user_domain_name

Type
unknown type

Default
<None>

Users domain name

password

Type
unknown type

Default
<None>

Users password

volumes

backups_enabled

Type
boolean

Default
True

Indicate if cinder-backup service is enabled. This is a temporary workaround until cinder-backup service becomes discoverable, see LP#1334856.

yaql

limit_iterators

Type
integer

Default
`200`

The maximum number of elements in collection expression can take for its evaluation.

memory_quota

Type
integer

Default
`10000`

The maximum size of memory in bytes that expression can take for its evaluation.

2.3.2 Heat Configuration Sample

The following is a sample heat configuration for adaptation and use. It is auto-generated from heat when this documentation is built, so if you are having issues with an option, please compare your version of heat with the version of this documentation.

See the online version of this documentation for the full example config file.

2.3.3 Orchestration log files

The corresponding log file of each Orchestration service is stored in the `/var/log/heat/` directory of the host on which each service runs.

Table 22: Log files used by Orchestration services

Log filename	Service that logs to the file
`heat-api.log`	Orchestration service API Service
`heat-engine.log`	Orchestration service Engine Service
`heat-manage.log`	Orchestration service events

2.3.4 Heat Sample Policy

Warning

JSON formatted policy file is deprecated since Heat 17.0.0 (Xena). This oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

The following is a sample heat policy file that has been auto-generated from default policy values in code. If youre using the default policies, then the maintenance of this file is not necessary, and it should not be copied into a deployment. Doing so will result in duplicate policy definitions. It is here to help explain which policy operations protect specific heat APIs, but it is not suggested to copy and paste into a deployment unless youre planning on providing a different policy for an operation that is not the default.

If you wish build a policy file, you can also use `tox -e genpolicy` to generate it.

The sample policy file can also be downloaded in file form.

```
# Decides what is required for the 'is_admin:True' check to succeed.
#"context_is_admin": "(role:admin and is_admin_project:True)"

# Default rule for project admin.
#"project_admin": "role:admin"

# Default rule for deny stack user.
#"deny_stack_user": "not role:heat_stack_user"

# Default rule for deny everybody.
#"deny_everybody": "!"

# Default rule for allow everybody.
#"allow_everybody": ""

# Performs non-lifecycle operations on the stack (Snapshot, Resume,
# Cancel update, or check stack resources). This is the default for
# all actions but can be overridden by more specific policies for
# individual actions.
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/actions
# Intended scope(s): project
#"actions:action": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "actions:action":"rule:deny_stack_user" has been deprecated since W
# in favor of "actions:action":"role:member and
# project_id:%(project_id)s".
# The actions API now supports system scope and default roles.

# Create stack snapshot
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/actions
# Intended scope(s): project
#"actions:snapshot": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "actions:snapshot":"rule:deny_stack_user" has been deprecated since
# W in favor of "actions:snapshot":"role:member and
# project_id:%(project_id)s".
# The actions API now supports system scope and default roles.

# Suspend a stack.
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/actions
# Intended scope(s): project
#"actions:suspend": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "actions:suspend":"rule:deny_stack_user" has been deprecated since W
```

(continues on next page)

(continued from previous page)

```
# in favor of "actions:suspend":"role:member and
# project_id:%(project_id)s".
# The actions API now supports system scope and default roles.

# Resume a suspended stack.
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/actions
# Intended scope(s): project
#"actions:resume": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "actions:resume":"rule:deny_stack_user" has been deprecated since W
# in favor of "actions:resume":"role:member and
# project_id:%(project_id)s".
# The actions API now supports system scope and default roles.

# Check stack resources.
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/actions
# Intended scope(s): project
#"actions:check": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "actions:check":"rule:deny_stack_user" has been deprecated since W
# in favor of "actions:check":"role:reader and
# project_id:%(project_id)s".
# The actions API now supports system scope and default roles.

# Cancel stack operation and roll back.
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/actions
# Intended scope(s): project
#"actions:cancel_update": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "actions:cancel_update":"rule:deny_stack_user" has been deprecated
# since W in favor of "actions:cancel_update":"role:member and
# project_id:%(project_id)s".
# The actions API now supports system scope and default roles.

# Cancel stack operation without rolling back.
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/actions
# Intended scope(s): project
#"actions:cancel_without_rollback": "role:member and project_id:%(project_id)s
↪"

# DEPRECATED
# "actions:cancel_without_rollback":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "actions:cancel_without_rollback":"role:member and
# project_id:%(project_id)s".
# The actions API now supports system scope and default roles.
```

(continues on next page)

(continued from previous page)

```
# Show build information.
# GET  /v1/{tenant_id}/build_info
# Intended scope(s): project
#"build_info:build_info": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "build_info:build_info":"rule:deny_stack_user" has been deprecated
# since W in favor of "build_info:build_info":"role:reader and
# project_id:%(project_id)s".
# The build API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:ListStacks": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "cloudformation:ListStacks":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:ListStacks":"role:reader and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:CreateStack": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "cloudformation:CreateStack":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:CreateStack":"role:member and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:DescribeStacks": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "cloudformation:DescribeStacks":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:DescribeStacks":"role:reader and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:DeleteStack": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "cloudformation:DeleteStack":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:DeleteStack":"role:member and
```

(continues on next page)

(continued from previous page)

```
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:UpdateStack": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "cloudformation:UpdateStack":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:UpdateStack":"role:member and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:CancelUpdateStack": "role:member and project_id:%(project_
↪id)s"

# DEPRECATED
# "cloudformation:CancelUpdateStack":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:CancelUpdateStack":"role:member and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:DescribeStackEvents": "role:reader and project_id:%(project_
↪id)s"

# DEPRECATED
# "cloudformation:DescribeStackEvents":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:DescribeStackEvents":"role:reader and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:ValidateTemplate": "role:reader and project_id:%(project_id)s
↪"

# DEPRECATED
# "cloudformation:ValidateTemplate":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:ValidateTemplate":"role:reader and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:GetTemplate": "role:reader and project_id:%(project_id)s"
```

(continues on next page)

(continued from previous page)

```
# DEPRECATED
# "cloudformation:GetTemplate":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:GetTemplate":"role:reader and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:EstimateTemplateCost": "role:reader and project_id:%(project_
↪id)s"

# DEPRECATED
# "cloudformation:EstimateTemplateCost":"rule:deny_stack_user" has
# been deprecated since W in favor of
# "cloudformation:EstimateTemplateCost":"role:reader and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:DescribeStackResource": "(role:reader and project_id:
↪%(project_id)s) or (role:heat_stack_user and project_id:%(project_id)s)"

# DEPRECATED
# "cloudformation:DescribeStackResource":"rule:allow_everybody" has
# been deprecated since W in favor of
# "cloudformation:DescribeStackResource":"(role:reader and
# project_id:%(project_id)s) or (role:heat_stack_user and
# project_id:%(project_id)s)".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:DescribeStackResources": "role:reader and project_id:
↪%(project_id)s"

# DEPRECATED
# "cloudformation:DescribeStackResources":"rule:deny_stack_user" has
# been deprecated since W in favor of
# "cloudformation:DescribeStackResources":"role:reader and
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# Intended scope(s): project
#"cloudformation:ListStackResources": "role:reader and project_id:%(project_
↪id)s"

# DEPRECATED
# "cloudformation:ListStackResources":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "cloudformation:ListStackResources":"role:reader and
```

(continues on next page)

(continued from previous page)

```
# project_id:%(project_id)s".
# The cloud formation API now supports system scope and default roles.

# List events.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/events
# Intended scope(s): project
#"events:index": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "events:index":"rule:deny_stack_user" has been deprecated since W in
# favor of "events:index":"role:reader and project_id:%(project_id)s".
# The events API now supports system scope and default roles.

# Show event.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/resources/{resource_
↪name}/events/{event_id}
# Intended scope(s): project
#"events:show": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "events:show":"rule:deny_stack_user" has been deprecated since W in
# favor of "events:show":"role:reader and project_id:%(project_id)s".
# The events API now supports system scope and default roles.

# List resources.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/resources
# Intended scope(s): project
#"resource:index": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "resource:index":"rule:deny_stack_user" has been deprecated since W
# in favor of "resource:index":"role:reader and
# project_id:%(project_id)s".
# The resources API now supports system scope and default roles.

# Show resource metadata.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/resources/{resource_
↪name}/metadata
# Intended scope(s): project
#"resource:metadata": "(role:reader and project_id:%(project_id)s) or␣
↪(role:heat_stack_user and project_id:%(project_id)s)"

# DEPRECATED
# "resource:metadata":"rule:allow_everybody" has been deprecated since
# W in favor of "resource:metadata":"(role:reader and
# project_id:%(project_id)s) or (role:heat_stack_user and
# project_id:%(project_id)s)".
# The resources API now supports system scope and default roles.
```

(continues on next page)

(continued from previous page)

```
# Signal resource.
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/resources/{resource_
↪name}/signal
# Intended scope(s): project
#"resource:signal": "(role:reader and project_id:%(project_id)s) or␣
↪(role:heat_stack_user and project_id:%(project_id)s)"

# DEPRECATED
# "resource:signal":"rule:allow_everybody" has been deprecated since W
# in favor of "resource:signal":"(role:reader and
# project_id:%(project_id)s) or (role:heat_stack_user and
# project_id:%(project_id)s)".
# The resources API now supports system scope and default roles.

# Mark resource as unhealthy.
# PATCH  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/resources/{resource_
↪name_or_physical_id}
# Intended scope(s): project
#"resource:mark_unhealthy": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "resource:mark_unhealthy":"rule:deny_stack_user" has been deprecated
# since W in favor of "resource:mark_unhealthy":"role:member and
# project_id:%(project_id)s".
# The resources API now supports system scope and default roles.

# Show resource.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/resources/{resource_
↪name}
# Intended scope(s): project
#"resource:show": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "resource:show":"rule:deny_stack_user" has been deprecated since W
# in favor of "resource:show":"role:reader and
# project_id:%(project_id)s".
# The resources API now supports system scope and default roles.

# Intended scope(s): project
#"resource_types:OS::Nova::Flavor": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Cinder::EncryptedVolumeType": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Cinder::VolumeType": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Cinder::Quota": "rule:project_admin"
```

(continues on next page)

(continued from previous page)

```
# Intended scope(s): project
#"resource_types:OS::Neutron::Quota": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Nova::Quota": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Octavia::Quota": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Manila::ShareType": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Neutron::ProviderNet": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Neutron::QoSPolicy": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Neutron::QoSBandwidthLimitRule": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Neutron::QoSDscpMarkingRule": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Neutron::QoSMinimumBandwidthRule": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Neutron::QoSMinimumPacketRateRule": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Neutron::Segment": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Nova::HostAggregate": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Cinder::QoSSpecs": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Cinder::QoSAssociation": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Keystone::*": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Blazar::Host": "rule:project_admin"
```

(continues on next page)

(continued from previous page)

```
# Intended scope(s): project
#"resource_types:OS::Octavia::Flavor": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Octavia::FlavorProfile": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Octavia::AvailabilityZone": "rule:project_admin"

# Intended scope(s): project
#"resource_types:OS::Octavia::AvailabilityZoneProfile": "rule:project_admin"

# Intended scope(s): project
#"service:index": "role:admin and project_id:%(project_id)s"

# DEPRECATED
# "service:index":"rule:context_is_admin" has been deprecated since W
# in favor of "service:index":"role:admin and
# project_id:%(project_id)s".
# The service API now supports system scope and default roles.

# List configs globally.
# GET  /v1/{tenant_id}/software_configs
#"software_configs:global_index": "rule:deny_everybody"

# List configs.
# GET  /v1/{tenant_id}/software_configs
# Intended scope(s): project
#"software_configs:index": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "software_configs:index":"rule:deny_stack_user" has been deprecated
# since W in favor of "software_configs:index":"role:reader and
# project_id:%(project_id)s".
# The software configuration API now support system scope and default
# roles.

# Create config.
# POST  /v1/{tenant_id}/software_configs
# Intended scope(s): project
#"software_configs:create": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "software_configs:create":"rule:deny_stack_user" has been deprecated
# since W in favor of "software_configs:create":"role:member and
# project_id:%(project_id)s".
# The software configuration API now support system scope and default
# roles.
```

(continues on next page)

(continued from previous page)

```
# Show config details.
# GET  /v1/{tenant_id}/software_configs/{config_id}
# Intended scope(s): project
#"software_configs:show": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "software_configs:show":"rule:deny_stack_user" has been deprecated
# since W in favor of "software_configs:show":"role:reader and
# project_id:%(project_id)s".
# The software configuration API now support system scope and default
# roles.

# Delete config.
# DELETE  /v1/{tenant_id}/software_configs/{config_id}
# Intended scope(s): project
#"software_configs:delete": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "software_configs:delete":"rule:deny_stack_user" has been deprecated
# since W in favor of "software_configs:delete":"role:member and
# project_id:%(project_id)s".
# The software configuration API now support system scope and default
# roles.

# List deployments.
# GET  /v1/{tenant_id}/software_deployments
# Intended scope(s): project
#"software_deployments:index": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "software_deployments:index":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "software_deployments:index":"role:reader and
# project_id:%(project_id)s".
# The software deployment API now supports system scope and default
# roles.

# Create deployment.
# POST  /v1/{tenant_id}/software_deployments
# Intended scope(s): project
#"software_deployments:create": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "software_deployments:create":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "software_deployments:create":"role:member and
# project_id:%(project_id)s".
# The software deployment API now supports system scope and default
# roles.
```

(continues on next page)

(continued from previous page)

```
# Show deployment details.
# GET  /v1/{tenant_id}/software_deployments/{deployment_id}
# Intended scope(s): project
#"software_deployments:show": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "software_deployments:show":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "software_deployments:show":"role:reader and
# project_id:%(project_id)s".
# The software deployment API now supports system scope and default
# roles.

# Update deployment.
# PUT  /v1/{tenant_id}/software_deployments/{deployment_id}
# Intended scope(s): project
#"software_deployments:update": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "software_deployments:update":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "software_deployments:update":"role:member and
# project_id:%(project_id)s".
# The software deployment API now supports system scope and default
# roles.

# Delete deployment.
# DELETE  /v1/{tenant_id}/software_deployments/{deployment_id}
# Intended scope(s): project
#"software_deployments:delete": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "software_deployments:delete":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "software_deployments:delete":"role:member and
# project_id:%(project_id)s".
# The software deployment API now supports system scope and default
# roles.

# Show server configuration metadata.
# GET  /v1/{tenant_id}/software_deployments/metadata/{server_id}
# Intended scope(s): project
#"software_deployments:metadata": "(role:reader and project_id:%(project_
↪id)s) or (role:heat_stack_user and project_id:%(project_id)s)"

# Abandon stack.
# DELETE  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/abandon
# Intended scope(s): project
```

(continues on next page)

(continued from previous page)

```
#"stacks:abandon": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:abandon":"rule:deny_stack_user" has been deprecated since W
# in favor of "stacks:abandon":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Create stack.
# POST  /v1/{tenant_id}/stacks
# Intended scope(s): project
#"stacks:create": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:create":"rule:deny_stack_user" has been deprecated since W
# in favor of "stacks:create":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Delete stack.
# DELETE  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}
# Intended scope(s): project
#"stacks:delete": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:delete":"rule:deny_stack_user" has been deprecated since W
# in favor of "stacks:delete":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# List stacks in detail.
# GET  /v1/{tenant_id}/stacks
# Intended scope(s): project
#"stacks:detail": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:detail":"rule:deny_stack_user" has been deprecated since W
# in favor of "stacks:detail":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Export stack.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/export
# Intended scope(s): project
#"stacks:export": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:export":"rule:deny_stack_user" has been deprecated since W
# in favor of "stacks:export":"role:member and
```

(continues on next page)

(continued from previous page)

```
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Generate stack template.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/template
# Intended scope(s): project
#"stacks:generate_template": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:generate_template":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "stacks:generate_template":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# List stacks globally.
# GET  /v1/{tenant_id}/stacks
#"stacks:global_index": "rule:deny_everybody"

# List stacks.
# GET  /v1/{tenant_id}/stacks
# Intended scope(s): project
#"stacks:index": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:index":"rule:deny_stack_user" has been deprecated since W in
# favor of "stacks:index":"role:reader and project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# List resource types.
# GET  /v1/{tenant_id}/resource_types
# Intended scope(s): project
#"stacks:list_resource_types": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:list_resource_types":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "stacks:list_resource_types":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# List template versions.
# GET  /v1/{tenant_id}/template_versions
# Intended scope(s): project
#"stacks:list_template_versions": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:list_template_versions":"rule:deny_stack_user" has been
# deprecated since W in favor of
```

(continues on next page)

(continued from previous page)

```
# "stacks:list_template_versions":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# List template functions.
# GET  /v1/{tenant_id}/template_versions/{template_version}/functions
# Intended scope(s): project
#"stacks:list_template_functions": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:list_template_functions":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "stacks:list_template_functions":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Find stack.
# GET  /v1/{tenant_id}/stacks/{stack_identity}
# Intended scope(s): project
#"stacks:lookup": "(role:reader and project_id:%(project_id)s) or (role:heat_
↪stack_user and project_id:%(project_id)s)"

# DEPRECATED
# "stacks:lookup":"rule:allow_everybody" has been deprecated since W
# in favor of "stacks:lookup":"(role:reader and
# project_id:%(project_id)s) or (role:heat_stack_user and
# project_id:%(project_id)s)".
# The stack API now supports system scope and default roles.

# Preview stack.
# POST  /v1/{tenant_id}/stacks/preview
# Intended scope(s): project
#"stacks:preview": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:preview":"rule:deny_stack_user" has been deprecated since W
# in favor of "stacks:preview":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Show resource type schema.
# GET  /v1/{tenant_id}/resource_types/{type_name}
# Intended scope(s): project
#"stacks:resource_schema": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:resource_schema":"rule:deny_stack_user" has been deprecated
# since W in favor of "stacks:resource_schema":"role:reader and
# project_id:%(project_id)s".
```

(continues on next page)

(continued from previous page)

```
# The stack API now supports system scope and default roles.

# Show stack.
# GET  /v1/{tenant_id}/stacks/{stack_identity}
# Intended scope(s): project
#"stacks:show": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:show":"rule:deny_stack_user" has been deprecated since W in
# favor of "stacks:show":"role:reader and project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Get stack template.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/template
# Intended scope(s): project
#"stacks:template": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:template":"rule:deny_stack_user" has been deprecated since W
# in favor of "stacks:template":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Get stack environment.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/environment
# Intended scope(s): project
#"stacks:environment": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:environment":"rule:deny_stack_user" has been deprecated
# since W in favor of "stacks:environment":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Get stack files.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/files
# Intended scope(s): project
#"stacks:files": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:files":"rule:deny_stack_user" has been deprecated since W in
# favor of "stacks:files":"role:reader and project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Update stack.
# PUT  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}
# Intended scope(s): project
#"stacks:update": "role:member and project_id:%(project_id)s"
```

(continues on next page)

(continued from previous page)

```
# DEPRECATED
# "stacks:update":"rule:deny_stack_user" has been deprecated since W
# in favor of "stacks:update":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Update stack (PATCH).
# PATCH  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}
# Intended scope(s): project
#"stacks:update_patch": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:update_patch":"rule:deny_stack_user" has been deprecated
# since W in favor of "stacks:update_patch":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Update stack (PATCH) with no changes.
# PATCH  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}
# Intended scope(s): project
#"stacks:update_no_change": "rule:stacks:update_patch"

# Preview update stack.
# PUT  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/preview
# Intended scope(s): project
#"stacks:preview_update": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:preview_update":"rule:deny_stack_user" has been deprecated
# since W in favor of "stacks:preview_update":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Preview update stack (PATCH).
# PATCH  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/preview
# Intended scope(s): project
#"stacks:preview_update_patch": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:preview_update_patch":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "stacks:preview_update_patch":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Validate template.
# POST  /v1/{tenant_id}/validate
# Intended scope(s): project
#"stacks:validate_template": "role:member and project_id:%(project_id)s"
```

(continues on next page)

(continued from previous page)

```
# DEPRECATED
# "stacks:validate_template":"rule:deny_stack_user" has been
# deprecated since W in favor of
# "stacks:validate_template":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Snapshot Stack.
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/snapshots
# Intended scope(s): project
#"stacks:snapshot": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:snapshot":"rule:deny_stack_user" has been deprecated since W
# in favor of "stacks:snapshot":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Show snapshot.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/snapshots/{snapshot_id}
# Intended scope(s): project
#"stacks:show_snapshot": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:show_snapshot":"rule:deny_stack_user" has been deprecated
# since W in favor of "stacks:show_snapshot":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Delete snapshot.
# DELETE  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/snapshots/{snapshot_
↪id}
# Intended scope(s): project
#"stacks:delete_snapshot": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:delete_snapshot":"rule:deny_stack_user" has been deprecated
# since W in favor of "stacks:delete_snapshot":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# List snapshots.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/snapshots
# Intended scope(s): project
#"stacks:list_snapshots": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:list_snapshots":"rule:deny_stack_user" has been deprecated
```

(continues on next page)

(continued from previous page)

```
# since W in favor of "stacks:list_snapshots":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Restore snapshot.
# POST  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/snapshots/{snapshot_id}
↪/restore
# Intended scope(s): project
#"stacks:restore_snapshot": "role:member and project_id:%(project_id)s"

# DEPRECATED
# "stacks:restore_snapshot":"rule:deny_stack_user" has been deprecated
# since W in favor of "stacks:restore_snapshot":"role:member and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# List outputs.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/outputs
# Intended scope(s): project
#"stacks:list_outputs": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:list_outputs":"rule:deny_stack_user" has been deprecated
# since W in favor of "stacks:list_outputs":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.

# Show outputs.
# GET  /v1/{tenant_id}/stacks/{stack_name}/{stack_id}/outputs/{output_key}
# Intended scope(s): project
#"stacks:show_output": "role:reader and project_id:%(project_id)s"

# DEPRECATED
# "stacks:show_output":"rule:deny_stack_user" has been deprecated
# since W in favor of "stacks:show_output":"role:reader and
# project_id:%(project_id)s".
# The stack API now supports system scope and default roles.
```

2.4 Administering Heat

2.4.1 Introduction

The OpenStack Orchestration service, a tool for orchestrating clouds, automatically configures and deploys resources in stacks. The deployments can be simple, such as deploying WordPress on Ubuntu with an SQL back end, or complex, such as starting a server group that auto scales by starting and stopping using real-time CPU loading information from the Telemetry service.

Orchestration stacks are defined with templates, which are non-procedural documents. Templates describe tasks in terms of resources, parameters, inputs, constraints, and dependencies. When the Orches-

tration service was originally introduced, it worked with AWS CloudFormation templates, which are in the JSON format.

The Orchestration service also runs Heat Orchestration Template (HOT) templates that are written in YAML. YAML is a terse notation that loosely follows structural conventions (colons, returns, indentation) that are similar to Python or Ruby. Therefore, it is easier to write, parse, grep, generate with tools, and maintain source-code management systems.

Orchestration can be accessed through a CLI and RESTful queries. The Orchestration service provides both an OpenStack-native REST API and a CloudFormation-compatible Query API. The Orchestration service is also integrated with the OpenStack dashboard to perform stack functions through a web interface.

For more information about using the Orchestration service through the command line, see the Heat Command-Line Interface reference.

2.4.2 Orchestration authorization model

The Orchestration authorization model defines the authorization process for requests during deferred operations. A common example is an auto-scaling group update. During the auto-scaling update operation, the Orchestration service requests resources of other components (such as servers from Compute or networks from Networking) to extend or reduce the capacity of an auto-scaling group.

The Orchestration service provides the following authorization models:

- Password authorization
- OpenStack Identity trusts authorization

Password authorization

The Orchestration service supports password authorization. Password authorization requires that a user pass a username and password to the Orchestration service. Encrypted password are stored in the database, and used for deferred operations.

Password authorization involves the following steps:

1. A user requests stack creation, by providing a token and username and password. The Dashboard or python-heatclient requests the token on the users behalf.
2. If the stack contains any resources that require deferred operations, then the orchestration engine fails its validation checks if the user did not provide a valid username/password.
3. The username/password are encrypted and stored in the Orchestration database.
4. Orchestration creates a stack.
5. Later, the Orchestration service retrieves the credentials and requests another token on behalf of the user. The token is not limited in scope and provides access to all the roles of the stack owner.

OpenStack Identity trusts authorization

A trust is an OpenStack Identity extension that enables delegation, and optionally impersonation through the OpenStack Identity service. The key terminology is *trustor* (the user delegating) and *trustee* (the user being delegated to).

To create a trust, the *trustor* (in this case, the user creating the stack in the Orchestration service) provides the OpenStack Identity service with the following information:

- The ID of the *trustee* (who you want to delegate to, in this case, the Orchestration service user).
- The roles to be delegated. Configure roles through the `heat.conf` file. Ensure the configuration contains whatever roles are required to perform the deferred operations on the users behalf. For example, launching an OpenStack Compute instance in response to an auto-scaling event.
- Whether to enable impersonation.

The OpenStack Identity service provides a *trust ID*, which is consumed by *only* the trustee to obtain a *trust scoped token*. This token is limited in scope, such that the trustee has limited access to those roles delegated. In addition, the trustee has effective impersonation of the trustor user if it was selected when creating the trust. For more information, see Identity management trusts.

Trusts authorization involves the following steps:

1. A user creates a stack through an API request (only the token is required).
2. The Orchestration service uses the token to create a trust between the stack owner (trustor) and the Orchestration service user (trustee). The service delegates a special role (or roles) as defined in the *trusts_delegated_roles* list in the Orchestration configuration file. By default, the Orchestration service sets all the roles from trustor available for trustee. Deployers might modify this list to reflect a local RBAC policy. For example, to ensure that the heat process can access only those services that are expected while impersonating a stack owner.
3. Orchestration stores the encrypted *trust ID* in the Orchestration database.
4. When a deferred operation is required, the Orchestration service retrieves the *trust ID* and requests a trust scoped token which enables the service user to impersonate the stack owner during the deferred operation. Impersonation is helpful, for example, so the service user can launch Compute instances on behalf of the stack owner in response to an auto-scaling event.

Authorization model configuration

Initially, the password authorization model was the default authorization model. Since the Kilo release, the Identity trusts authorization model is enabled for the Orchestration service by default.

To enable the password authorization model, change the following parameter in the `heat.conf` file:

```
deferred_auth_method=password
```

To enable the trusts authorization model, change the following two parameters in the `heat.conf` file.

Specify the authentication method for the deferred Orchestration actions. This parameter triggers creating *trust ID* and stores it in the Orchestration database:

```
deferred_auth_method=trusts
```

Allow reauthentication with the trust scoped token issued by using the stored *trust ID* for long running tasks:

```
reauthentication_auth_method=trusts
```

To specify the trustor roles that it delegates to trustee during authorization, specify the `trusts_delegated_roles` parameter in the `heat.conf` file. If `trusts_delegated_roles` is not defined, then all the trustor roles are delegated to trustee.

> **Note**
>
> The trustor delegated roles must be pre-configured in the OpenStack Identity service before using them in the Orchestration service.

2.4.3 Stack domain users

Stack domain users allow the Orchestration service to authorize and start the following operations within booted virtual machines:

- Provide metadata to agents inside instances. Agents poll for changes and apply the configuration that is expressed in the metadata to the instance.
- Detect when an action is complete. Typically, software configuration on a virtual machine after it is booted. Compute moves the VM state to Active as soon as it creates it, not when the Orchestration service has fully configured it.
- Provide application level status or meters from inside the instance. For example, allow auto-scaling actions to be performed in response to some measure of performance or quality of service.

The Orchestration service provides APIs that enable all of these operations, but all of those APIs require authentication. For example, credentials to access the instance that the agent is running upon. The heat-cfntools agents use signed requests, which require an ec2 key pair created through Identity. The key pair is then used to sign requests to the Orchestration CloudFormation and CloudWatch compatible APIs, which are authenticated through signature validation. Signature validation uses the Identity ec2tokens extension.

Stack domain users encapsulate all stack-defined users (users who are created as a result of data that is contained in an Orchestration template) in a separate domain. The separate domain is created specifically to contain data related to the Orchestration stacks only. A user is created, which is the *domain admin*, and Orchestration uses the *domain admin* to manage the lifecycle of the users in the stack *user domain*.

Stack domain users configuration

To configure stack domain user, the Orchestration service completes the following tasks:

1. A special OpenStack Identity service domain is created. For example, a domain that is called `heat` and the ID is set with the `stack_user_domain` option in the `heat.conf` file.
2. A user with sufficient permissions to create and delete projects and users in the `heat` domain is created.
3. The username and password for the domain admin user is set in the `heat.conf` file (`stack_domain_admin` and `stack_domain_admin_password`). This user administers *stack domain users* on behalf of stack owners, so they no longer need to be administrators themselves. The risk of this escalation path is limited because the `heat_domain_admin` is only given administrative permission for the `heat` domain.

To set up stack domain users, complete the following steps:

1. Create the domain:

   `$OS_TOKEN` refers to a token. For example, the service admin token or some other valid token for a user with sufficient roles to create users and domains. `$KS_ENDPOINT_V3` refers to the v3 OpenStack Identity endpoint (for example, `http://keystone_address:5000/v3` where *keystone_address* is the IP address or resolvable name for the Identity service).

```
$ openstack --os-token $OS_TOKEN --os-url=$KS_ENDPOINT_V3 --os-\
  identity-api-version=3 domain create heat --description "Owns \
  users and projects created by heat"
```

The domain ID is returned by this command, and is referred to as `$HEAT_DOMAIN_ID` below.

2. Create the user:

```
$ openstack --os-token $OS_TOKEN --os-url=$KS_ENDPOINT_V3 --os-\
  identity-api-version=3 user create --password $PASSWORD --domain \
  $HEAT_DOMAIN_ID heat_domain_admin --description "Manages users \
  and projects created by heat"
```

The user ID is returned by this command and is referred to as `$DOMAIN_ADMIN_ID` below.

3. Make the user a domain admin:

```
$ openstack --os-token $OS_TOKEN --os-url=$KS_ENDPOINT_V3 --os-\
  identity-api-version=3 role add --user $DOMAIN_ADMIN_ID --domain \
  $HEAT_DOMAIN_ID admin
```

Then you must add the domain ID, username and password from these steps to the `heat.conf` file:

```
stack_domain_admin_password = password
stack_domain_admin = heat_domain_admin
stack_user_domain = domain id returned from domain create above
```

Usage workflow

The following steps are run during stack creation:

1. Orchestration creates a new *stack domain project* in the `heat` domain if the stack contains any resources that require creation of a *stack domain user*.
2. For any resources that require a user, the Orchestration service creates the user in the *stack domain project*. The *stack domain project* is associated with the Orchestration stack in the Orchestration database, but is separate and unrelated (from an authentication perspective) to the stack owners project. The users who are created in the stack domain are still assigned the `heat_stack_user` role, so the API surface they can access is limited through the `policy.yaml` file. For more information, see OpenStack Identity documentation.
3. When API requests are processed, the Orchestration service performs an internal lookup, and allows stack details for a given stack to be retrieved. Details are retrieved from the database for both the stack owners project (the default API path to the stack) and the stack domain project, subject to the `policy.yaml` restrictions.

This means there are now two paths that can result in the same data being retrieved through the Orchestration API. The following example is for resource-metadata:

```
GET v1/{stack_owner_project_id}/stacks/{stack_name}/\
{stack_id}/resources/{resource_name}/metadata
```

or:

```
GET v1/{stack_domain_project_id}/stacks/{stack_name}/\
{stack_id}/resources/{resource_name}/metadata
```

The stack owner uses the former (via `openstack stack resource metadata STACK RESOURCE`), and any agents in the instance use the latter.

2.5 Scaling a Deployment

When deploying in an environment where a large number of incoming requests need to be handled, the API and engine services can be overloaded. In those scenarios, in order to increase the system performance, it can be helpful to run multiple load-balanced APIs and engines.

This guide details how to scale out the REST API, the CFN API, and the engine, also known as the *heat-api*, *heat-api-cfn*, and *heat-engine* services, respectively.

2.5.1 Assumptions

This guide, using a devstack installation of OpenStack, assumes that:

1. You have configured devstack from Single Machine Installation Guide;
2. You have set up heat on devstack, as defined at *heat and DevStack*;
3. You have installed HAProxy on the devstack server.

2.5.2 Architecture

This section shows the basic heat architecture, the load balancing mechanism used and the target scaled out architecture.

Basic Architecture

The heat architecture is as defined at *heat architecture* and shown in the diagram below, where we have a CLI that sends HTTP requests to the REST and CFN APIs, which in turn make calls using AMQP to the heat-engine:

```
                  |- [REST API] -|
[CLI] -- <HTTP> --               -- <AMQP> -- [ENGINE]
                  |- [CFN API]  -|
```

Load Balancing

As there is a need to use a load balancer mechanism between the multiple APIs and the CLI, a proxy has to be deployed.

Because the heat CLI and APIs communicate by exchanging HTTP requests and responses, a HAProxy HTTP load balancer server will be deployed between them.

This way, the proxy will take the CLIs requests to the APIs and act on their behalf. Once the proxy receives a response, it will be redirected to the caller CLI.

A round-robin distribution of messages from the AMQP queue will act as the load balancer for multiple engines. Check that your AMQP service is configured to distribute messages round-robin (RabbitMQ does this by default).

Target Architecture

A scaled out heat architecture is represented in the diagram below:

```
                              |- [REST-API] -|
                              |-    ...     -|
                              |- [REST-API] -|             |- [ENGINE] -|
[CLI] -- <HTTP> -- [PROXY] --                -- <AMQP> -- |-    ...   -|
                              |- [API-CFN]  -|             |- [ENGINE] -|
                              |-    ...     -|
                              |- [API-CFN]  -|
```

Thus, a request sent from the CLI looks like:

1. CLI contacts the proxy;
2. The HAProxy server, acting as a load balancer, redirects the call to an API instance;
3. The API server sends messages to the AMQP queue, and the engines pick up messages in round-robin fashion.

2.5.3 Deploying Multiple APIs

In order to run a heat component separately, you have to execute one of the python scripts located at the *bin* directory of your heat repository.

These scripts take as argument a configuration file. When using devstack, the configuration file is located at */etc/heat/heat.conf*. For instance, to start new REST and CFN API services, you must run:

```
python bin/heat-api --config-file=/etc/heat/heat.conf
python bin/heat-api-cfn --config-file=/etc/heat/heat.conf
```

Each API service must have a unique address to listen. Also the host value should be unique.

```
[DEFAULT]
host = {host}
```

If you wish to run multiple API processes on the same machine, you must create multiple copies of the heat.conf file, each containing a unique host string.

In addition, if you want to run some API services in different machines than the devstack server, you have to update the loopback addresses found at the *sql_connection* and *rabbit_host* properties to the devstack servers IP, which must be reachable from the remote machine.

2.5.4 Deploying Multiple Engines

All engines must be configured to use the same AMQP service. Ensure that all of the *rabbit_** and *kombu_** configuration options in the *[DEFAULT]* section of */etc/heat/heat.conf* match across each machine that will be running an engine. By using the same AMQP configuration, each engine will subscribe to the same AMQP *engine* queue and pick up work in round-robin fashion with the other engines.

One or more engines can be deployed per host. Depending on the hosts CPU architecture, it may be beneficial to deploy several engines on a single machine.

To start multiple engines on the same machine, simply start multiple *heat-engine* processes:

```
python bin/heat-engine --config-file=/etc/heat/heat.conf &
python bin/heat-engine --config-file=/etc/heat/heat.conf &
```

2.5.5 Deploying the Proxy

In order to simplify the deployment of the HAProxy server, we will replace the REST and CFN APIs deployed when installing devstack by the HAProxy server. This way, there is no need to update, on the CLI, the addresses where it should look for the APIs. In this case, when it makes a call to any API, it will find the proxy, acting on their behalf.

To deploy the HAProxy server on the devstack server, run *haproxy -f apis-proxy.conf*, where this configuration file looks like:

```
global
    daemon
    maxconn 4000

defaults
    log  global
    maxconn  8000
    option  redispatch
    retries  3
    timeout  http-request 10s
    timeout  queue 1m
    timeout  connect 10s
    timeout  client 1m
    timeout  server 1m
    timeout  check 10s

listen rest_api_proxy
    # The values required below are the original ones that were in
    # /etc/heat/heat.conf on the devstack server.
    bind {API_HOST}:{API_PORT}
    balance  source
    option  tcpka
    option  httpchk
    # The values required below are the different addresses supplied when
    # running the REST API instances.
    server SERVER_1 {HOST_1}:{PORT_1}
    ...
    server SERVER_N {HOST_N}:{PORT_N}

listen cfn_api_proxy
    # The values required below are the original ones that were in
    # /etc/heat/heat.conf on the devstack server.
    bind {API_CFN_HOST}:{API_CFN_PORT}
    balance  source
    option  tcpka
    option  httpchk
    # The values required below are the different addresses supplied when
    # running the CFN API instances.
```

(continues on next page)

(continued from previous page)

```
    server SERVER_1 {HOST_1}:{PORT_1}
    ...
    server SERVER_N {HOST_N}:{PORT_N}
```

2.5.6 Sample

This section aims to clarify some aspects of the scaling out solution, as well as to show more details of the configuration by describing a concrete sample.

Architecture

This section shows a basic OpenStack architecture and the target one that will be used for testing of the scaled-out heat services.

Basic Architecture

For this sample, consider that:

1. We have an architecture composed by 3 machines configured in a LAN, with the addresses A: 10.0.0.1; B: 10.0.0.2; and C: 10.0.0.3;
2. The OpenStack devstack installation, including the heat module, has been done in the machine A, as shown in the *Assumptions* section.

Target Architecture

At this moment, everything is running in a single devstack server. The next subsections show how to deploy a scaling out heat architecture by:

1. Running one REST and one CFN API on the machines B and C;
2. Setting up the HAProxy server on the machine A.

Running the API and Engine Services

For each machine, B and C, you must do the following steps:

1. Clone the heat repository https://opendev.org/openstack/heat, run:

::

git clone https://opendev.org/openstack/heat

2. Create a local copy of the configuration file */etc/heat/heat.conf* from the machine A;
3. Make required changes on the configuration file;
4. Enter the heat local repository and run:

```
python bin/heat-api --config-file=/etc/heat/heat.conf
python bin/heat-api-cfn --config-file=/etc/heat/heat.conf
```

5. Start as many *heat-engine* processes as you want running on that machine:

```
python bin/heat-engine --config-file=/etc/heat/heat.conf &
python bin/heat-engine --config-file=/etc/heat/heat.conf &
...
```

Changes On Configuration

The original file from A looks like:

```
[DEFAULT]
...
sql_connection = mysql+pymysql://root:admin@127.0.0.1/heat?charset=utf8
rabbit_host = localhost
host = localhost
...
```

After the changes for B, it looks like:

```
[DEFAULT]
...
sql_connection = mysql+pymysql://root:admin@10.0.0.1/heat?charset=utf8
rabbit_host = 10.0.0.1
host = 10.0.0.2
...
```

Setting Up HAProxy

On the machine A, kill the *heat-api* and *heat-api-cfn* processes by running *pkill heat-api* and *pkill heat-api-cfn*. After, run *haproxy -f apis-proxy.conf* with the following configuration:

```
 global
    daemon
    maxconn 4000

defaults
    log  global
    maxconn  8000
    option  redispatch
    retries  3
    timeout  http-request 10s
    timeout  queue 1m
    timeout  connect 10s
    timeout  client 1m
    timeout  server 1m
    timeout  check 10s

listen rest_api_proxy
    bind 10.0.0.1:8004
    balance  source
    option  tcpka
    option  httpchk
    server rest-server-1 10.0.0.2:8004
    server rest-server-2 10.0.0.3:8004

listen cfn_api_proxy
    bind 10.0.0.1:8000
```

(continues on next page)

(continued from previous page)

```
    balance  source
    option  tcpka
    option  httpchk
    server cfn-server-1 10.0.0.2:8000
    server cfn-server-2 10.0.0.3:8000
```

2.6 Upgrades Guideline

This document outlines several steps and notes for operators to reference when upgrading their heat from previous versions of OpenStack.

> **Note**
>
> This document is only tested in the case of upgrading between sequential releases.

2.6.1 Plan to upgrade

- Read and ensure you understand the release notes for the next release.
- Make a backup of your database.
- Upgrades are only supported one series at a time, or within a series.

2.6.2 Cold Upgrades

Heat already supports cold-upgrades, where the heat services have to be down during the upgrade. For time-consuming upgrades, it may be unacceptable for the services to be unavailable for a long period of time. This type of upgrade is quite simple, follow the below steps:

1. Stop all heat-api and heat-engine services.
2. Uninstall old code.
3. Install new code.
4. Update configurations.
5. Do Database sync (most time-consuming step)
6. Start all heat-api and heat-engine services.

2.6.3 Rolling Upgrades

> **Note**
>
> Rolling Upgrade is supported since Pike, which means operators can rolling upgrade Heat services from Ocata to Pike release with minimal downtime.

A rolling upgrade would provide a better experience for the users and operators of the cloud. A rolling upgrade would allow individual heat-api and heat-engine services to be upgraded one at a time, with the rest of the services still available. This upgrade would have minimal downtime. Please check spec about rolling upgrades.

Prerequisites

- Multiple Heat nodes.
- A load balancer or some other type of redirection device is being used in front of nodes that run heat-api services in such a way that a node can be dropped out of rotation. That node continues running the Heat services (heat-api or heat-engine) but is no longer having requests routed to it.

Procedure

These following steps are the process to upgrade Heat with minimal downtime:

1. Install the code for the next version of Heat either in a virtual environment or a separate control plane node, including all the python dependencies.
2. Using the newly installed heat code, run the following command to sync the database up to the most recent version. These schema change operations should have minimal or no effect on performance, and should not cause any operations to fail.

   ```
   heat-manage db_sync
   ```

3. At this point, new columns and tables may exist in the database. These DB schema changes are done in a way that both the N and N+1 release can perform operations against the same schema.
4. Create a new rabbitmq vhost for the new release and change the transport_url configuration in heat.conf file to be:

   ```
   transport_url = rabbit://<user>:<password>@<host>:5672/<new_vhost>
   ```

   for all upgrade services.
5. Stop heat-engine gracefully, Heat has supported graceful shutdown features (see the spec about rolling upgrades). Then start new heat-engine with new code (and corresponding configuration).

   Note

   Remember to do Step 4, this would ensure that the existing engines would not communicate with the new engine.

6. A heat-api service is then upgraded and started with the new rabbitmq vhost.

   Note

   The second way to do this step is switch heat-api service to use new vhost first (but remember not to shut down heat-api) and upgrade it.

7. The above process can be followed till all heat-api and heat-engine services are upgraded.

   Note

   Make sure that all heat-api services has been upgraded before you start to upgrade the last heat-engine service.

> **Warning**
>
> With the convergence architecture, whenever a resource completes the engine will send RPC messages to another (or the same) engine to start work on the next resource(s) to be processed. If the last engine is going to be shut down gracefully, it will finish what it is working on, which may post more messages to queues. It means the graceful shutdown does not wait for queues to drain. The shutdown leaves some messages unprocessed and any IN_PROGRESS stacks would get stuck without any forward progress. The operator must be careful when shutting down the last engine, make sure queues have no unprocessed messages before doing it. The operator can check the queues directly with RabbitMQs management plugin.

8. Once all services are upgraded, double check the DB and services

2.6.4 References

2.7 Man pages for services and utilities

2.7.1 Heat services

heat-engine

SYNOPSIS

```
heat-engine [options]
```

DESCRIPTION

heat-engine is the heat project server with an internal RPC api called by the heat-api server.

INVENTORY

The heat-engine does all the orchestration work and is the layer in which the resource integration is implemented.

OPTIONS

`--config-file`

Path to a config file to use. Multiple config files can be specified, with values in later files taking precedence.

`--config-dir`

Path to a config directory to pull .conf files from. This file set is sorted, so as to provide a predictable parse order if individual options are over-ridden. The set is parsed after the file(s), if any, specified via config-file, hence over-ridden options in the directory take precedence.

`--version`

Show programs version number and exit. The output could be empty if the distribution didnt specify any version information.

FILES

- /etc/heat/heat.conf

heat-api

SYNOPSIS

```
heat-api [options]
```

DESCRIPTION

heat-api provides an external REST API to the heat project.

INVENTORY

heat-api is a service that exposes an external REST based api to the heat-engine service. The communication between the heat-api and heat-engine uses message queue based RPC.

OPTIONS

--config-file

Path to a config file to use. Multiple config files can be specified, with values in later files taking precedence.

--config-dir

Path to a config directory to pull .conf files from. This file set is sorted, so as to provide a predictable parse order if individual options are over-ridden. The set is parsed after the file(s), if any, specified via config-file, hence over-ridden options in the directory take precedence.

--version

Show programs version number and exit. The output could be empty if the distribution didnt specify any version information.

FILES

- /etc/heat/heat.conf

heat-api-cfn

SYNOPSIS

```
heat-api-cfn [options]
```

DESCRIPTION

heat-api-cfn is a CloudFormation compatible API service to the heat project.

INVENTORY

heat-api-cfn is a service that exposes an external REST based api to the heat-engine service. The communication between the heat-api-cfn and heat-engine uses message queue based RPC.

OPTIONS

--config-file

Path to a config file to use. Multiple config files can be specified, with values in later files taking precedence.

--config-dir

Path to a config directory to pull .conf files from. This file set is sorted, so as to provide a predictable parse order if individual options are over-ridden. The set is parsed after the file(s), if any, specified via config-file, hence over-ridden options in the directory take precedence.

--version

Show programs version number and exit. The output could be empty if the distribution didnt specify any version information.

FILES

- /etc/heat/heat.conf

2.7.2 Heat utilities

heat-manage

SYNOPSIS

`heat-manage <action> [options]`

DESCRIPTION

heat-manage helps manage heat specific database operations.

OPTIONS

The standard pattern for executing a heat-manage command is: `heat-manage <command> [<args>]`

Run with -h to see a list of available commands: `heat-manage -h`

Commands are `db_version`, `db_sync`, `purge_deleted`, `migrate_convergence_1`, `migrate_properties_data`, and `service`. Detailed descriptions are below.

`heat-manage db_version`

Print out the db schema version.

`heat-manage db_sync`

Sync the database up to the most recent version.

`heat-manage purge_deleted [-g {days,hours,minutes,seconds}] [-p project_id] [age]`

Purge db entries marked as deleted and older than [age]. When project_id argument is provided, only entries belonging to this project will be purged.

`heat-manage migrate_properties_data`

Migrates properties data from the legacy locations in the db (resource.properties_data and event.resource_properties) to the modern location, the resource_properties_data table.

`heat-manage migrate_convergence_1 [stack_id]`

Migrates [stack_id] from non-convergence to convergence. This requires running convergence enabled heat engine(s) and cant be done when they are offline.

`heat-manage service list`

Shows details for all currently running heat-engines.

`heat-manage service clean`

Clean dead engine records.

`heat-manage --version`

Shows programs version number and exit. The output could be empty if the distribution didnt specify any version information.

FILES

The /etc/heat/heat.conf file contains global options which can be used to configure some aspects of heat-manage, for example the DB connection and logging.

BUGS

Heat bugs are managed through StoryBoard OpenStack Heat Stories

heat-db-setup

SYNOPSIS

`heat-db-setup [COMMANDS] [OPTIONS]`

DESCRIPTION

heat-db-setup is a tool which configures the local MySQL database for heat. Typically distro-specific tools would provide this functionality so please read the distro-specific documentation for configuring heat.

COMMANDS

`rpm`

Indicate the distribution is a RPM packaging based distribution.

`deb`

Indicate the distribution is a DEB packaging based distribution.

OPTIONS

`-h, --help`

Print usage information.

`-p, --password`

Specify the password for the heat MySQL user that the script will use to connect to the heat MySQL database. By default, the password heat will be used.

-r, --rootpw

Specify the root MySQL password. If the script installs the MySQL server, it will set the root password to this value instead of prompting for a password. If the MySQL server is already installed, this password will be used to connect to the database instead of having to prompt for it.

-y, --yes

In cases where the script would normally ask for confirmation before doing something, such as installing mysql-server, just assume yes. This is useful if you want to run the script non-interactively.

EXAMPLES

heat-db-setup rpm -p heat_password -r mysql_pwd -y

heat-db-setup deb -p heat_password -r mysql_pwd -y

heat-db-setup rpm

BUGS

Heat bugs are managed through StoryBoard OpenStack Heat Stories

heat-keystone-setup-domain

SYNOPSIS

```
heat-keystone-setup-domain [OPTIONS]
```

DESCRIPTION

The *heat-keystone-setup-domain* tool configures keystone by creating a stack user domain and the user credential used to manage this domain. A stack user domain can be treated as a namespace for projects, groups and users created by heat. The domain will have an admin user that manages other users, groups and projects in the domain.

This script requires admin keystone credentials to be available in the shell environment by setting *OS_USERNAME* and *OS_PASSWORD*.

After running this script, a user needs to take actions to check or modify the heat configuration file (e.g. /etc/heat/heat.conf). The tool is NOT performing these updates on behalf of the user.

Distributions may provide other tools to setup stack user domain for use with heat, so check the distro documentation first. Other tools are available to set up the stack user domain, for example *python-openstackclient*, which is preferred to this tool where it is available.

OPTIONS

-h, --help

Print usage information.

--config-dir <DIR>

Path to a config directory from which to read the `heat.conf` file(s). This file set is sorted, so as to provide a predictable parse order if individual options are over-ridden. The set is parsed after the file(s) specified via previous config-file, arguments hence over-ridden options in the directory take precedence.

--config-file <PATH>

Path to a config file to use. Multiple config files can be specified, with values in later files taking precedence. The default files used is */etc/heat/heat.conf*.

--stack-domain-admin <USERNAME>

Name of a user for Keystone to create, which has roles sufficient to manage users (i.e. stack domain users) and projects (i.e. stack domain projects) in the stack user domain.

Another way to specify the admin user name is by setting an environment variable *STACK_DOMAIN_ADMIN* before running this tool. If both command line arguments and environment variable are specified, the command line arguments take precedence.

--stack-domain-admin-password <PASSWORD>

Password for the stack-domain-admin user.

The password can be instead specified using an environment variable *STACK_DOMAIN_ADMIN_PASSWORD* before invoking this tool. If both command line arguments and environment variable are specified, the command line arguments take precedence.

--stack-user-domain-name <DOMAIN>

Name of domain to create for stack users.

The domain name can be instead specified using an environment variable *STACK_USER_DOMAIN_NAME* before invoking this tool. If both command line arguments and environment variable are specified, the command line argument take precedence.

--version

Show programs version number and exit. The output could be empty if the distribution didnt specify any version information.

EXAMPLES

heat-keystone-setup-domain

heat-keystone-setup-domain stack-user-domain-name heat_user_domain
stack-domain-admin heat_domain_admin stack-domain-admin-password verysecrete

BUGS

Heat bugs are managed through StoryBoard OpenStack Heat Stories

heat-status

Synopsis

```
heat-status <category> <command> [<args>]
```

Description

heat-status is a tool that provides routines for checking the status of a Heat deployment.

Options

The standard pattern for executing a **heat-status** command is:

```
heat-status <category> <command> [<args>]
```

Run without arguments to see a list of available command categories:

```
heat-status
```

Categories are:

- `upgrade`

Detailed descriptions are below.

You can also run with a category argument such as `upgrade` to see a list of all commands in that category:

```
heat-status upgrade
```

These sections describe the available categories and arguments for **heat-status**.

Upgrade

heat-status upgrade check

Performs a release-specific readiness check before restarting services with new code. This command expects to have complete configuration and access to databases and services.

Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

History of Checks

12.0.0 (Stein)

- Placeholder to be filled in with checks as they are added in Stein.

CHAPTER

THREE

USING HEAT

3.1 Creating your first stack

3.1.1 Confirming you can access a Heat endpoint

Before any Heat commands can be run, your cloud credentials need to be sourced:

```
$ source openrc
```

You can confirm that Heat is available with this command:

```
$ openstack stack list
```

This should return an empty line

3.1.2 Preparing to create a stack

Download and register the image:

```
$ wget https://download.fedoraproject.org/pub/fedora/linux/releases/37/Cloud/
↪x86_64/images/Fedora-Cloud-Base-37-1.7.x86_64.qcow2
$ openstack image create \
                      --disk-format=qcow2 \
                      --container-format=bare \
                      --file=Fedora-Cloud-Base-37-1.7.x86_64.qcow2 \
                      my-fedora-image
```

Your cloud will have different flavors and images available for launching instances, you can discover what is available by running:

```
$ openstack flavor list
$ openstack image list
```

To allow you to SSH into instances launched by Heat, a keypair will be generated:

```
$ openstack keypair create heat_key > heat_key.priv
$ chmod 600 heat_key.priv
```

3.1.3 Launching a stack

Now lets launch a stack, using an example template from the heat-templates repository:

```
$ openstack stack create -t https://opendev.org/openstack/heat-templates/src/
↪branch/master/hot/F20/WordPress_Native.yaml --parameter key_name=heat_key --
↪parameter image_id=my-fedora-image --parameter instance_type=m1.small␣
↪teststack
```

Which will respond:

```
+--------------------------------------+-----------+--------------------+-----
↪-----------------+
| ID                                   | Name      | Status             |␣
↪Created              |
+--------------------------------------+-----------+--------------------+-----
↪-----------------+
| 718a712a-2571-4eac-aa03-426de00ecb43 | teststack | CREATE_IN_PROGRESS |␣
↪2017-04-11T03:06:24Z |
+--------------------------------------+-----------+--------------------+-----
↪-----------------+
```

Note

Link on Heat template presented in command above should reference on RAW template. In case if it be a html page with template, Heat will return an error.

Note

You cannot rename a stack after it has been launched.

List stacks

List the stacks in your tenant:

```
$ openstack stack list
```

List stack events

List the events related to a particular stack:

```
$ openstack stack event list teststack
```

Describe the wordpress stack

Show detailed state of a stack:

```
$ openstack stack show teststack
```

Note: After a few seconds, the stack_status should change from `IN_PROGRESS` to `CREATE_COMPLETE`.

Verify instance creation

Because the software takes some time to install from the repository, it may be a few minutes before the Wordpress instance is in a running state.

Point a web browser at the location given by the `WebsiteURL` output as shown by `openstack stack output show`:

```
$ WebsiteURL=$(openstack stack output show teststack WebsiteURL -c output_
↪value -f value)
$ curl $WebsiteURL
```

Delete the instance when done

Note: The list operation will show no running stack.:

```
$ openstack stack delete teststack
$ openstack stack list
```

You can explore other heat commands by referring to the Heat command reference for the OpenStack Command-Line Interface; then read the *Template Guide* and start authoring your own templates.

3.2 Glossary

API server
HTTP REST API service for heat.

CFN
An abbreviated form of AWS CloudFormation.

constraint
Defines valid input *parameters* for a *template*.

dependency
When a *resource* must wait for another resource to finish creation before being created itself. Heat adds an implicit dependency when a resource references another resource or one of its *attributes*. An explicit dependency can also be created by the user in the template definition.

environment
Used to affect the run-time behavior of the template. Provides a way to override the default resource implementation and parameters passed to Heat. See *Environments*.

Heat Orchestration Template
A particular *template* format that is native to Heat. Heat Orchestration Templates are expressed in YAML and are not backwards-compatible with CloudFormation templates.

HOT
An acronym for *Heat Orchestration Template*.

input parameters
See *parameters*.

Metadata
May refer to *Resource Metadata*, *Nova Instance metadata*, or the *Metadata service*.

Metadata service
A Compute service that enables virtual machine instances to retrieve instance-specific data. See Nova Metadata service documentation.

multi-region
A feature of Heat that supports deployment to multiple regions.

nested resource
A *resource* instantiated as part of a *nested stack*.

nested stack
A *template* referenced by URL inside of another template. Used to reduce redundant resource definitions and group complex architectures into logical groups.

Nova Instance metadata
User-provided *key:value* pairs associated with a Compute Instance. See Instance-specific data (OpenStack Operations Guide).

OpenStack
Open source software for building private and public clouds.

orchestrate
Arrange or direct the elements of a situation to produce a desired effect.

outputs
A top-level block in a *template* that defines what data will be returned by a stack after instantiation.

parameters
A top-level block in a *template* that defines what data can be passed to customise a template when it is used to create or update a *stack*.

provider resource
A *resource* implemented by a *provider template*. The parent resources properties become the *nested stacks* parameters.

provider template
Allows user-definable *resource providers* to be specified via *nested stacks*. The nested stacks *outputs* become the parent stacks *attributes*.

resource
An element of OpenStack infrastructure instantiated from a particular *resource provider*. See also *nested resource*.

resource attribute
Data that can be obtained from a *resource*, e.g. a servers public IP or name. Usually passed to another resources *properties* or added to the stacks *outputs*.

resource group
A *resource provider* that creates one or more identically configured *resources* or *nested resources*.

Resource Metadata
A *resource property* that contains CFN-style template metadata. See AWS::CloudFormation::Init (AWS CloudFormation User Guide)

resource plugin
Python code that understands how to instantiate and manage a *resource*. See Heat Resource Plugins (OpenStack wiki).

resource property
Data utilized for the instantiation of a *resource*. Can be defined statically in a *template* or passed in as *input parameters*.

resource provider
The implementation of a particular resource type. May be a *resource plugin* or a *provider template*.

stack
A collection of instantiated *resources* that are defined in a single *template*.

stack resource
A *resource provider* that allows the management of a *nested stack* as a *resource* in a parent stack.

template
An orchestration document that details everything needed to carry out an *orchestration*.

template resource
See *provider resource*.

user data
A *resource property* that contains a user-provided data blob. User data gets passed to cloud-init to automatically configure instances at boot time. See also Nova User data documentation.

wait condition
A *resource provider* that provides a way to communicate data or events from servers back to the orchestration engine. Most commonly used to pause the creation of the *stack* while the server is being configured.

3.3 Working with Templates

3.3.1 Template Guide

Heat Orchestration Template (HOT) Guide

HOT is a template format supported by the heat, along with the other template format, i.e. the Heat CloudFormation-compatible format (CFN). This guide is targeted towards template authors and explains how to write HOT templates based on examples. A detailed specification of HOT can be found at *Heat Orchestration Template (HOT) specification*.

Status

HOT is in the process of surpassing the functionality of the CFN. This guide will be updated periodically whenever new features get implemented for HOT.

Writing a hello world HOT template

This section gives an introduction on how to write HOT templates, starting from very basic steps and then going into more and more detail by means of examples.

A most basic template

The most basic template you can think of may contain only a single resource definition using only predefined properties (along with the mandatory Heat template version tag). For example, the template below could be used to simply deploy a single compute instance.

```
heat_template_version: 2015-04-30

description: Simple template to deploy a single compute instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: my_key
      image: F18-x86_64-cfntools
      flavor: m1.small
```

Each HOT template has to include the *heat_template_version* key with a valid version of HOT, e.g. 2015-10-15 (see *Heat template version* for a list of all versions). While the *description* is optional, it is good practice to include some useful text that describes what users can do with the template. In case you want to provide a longer description that does not fit on a single line, you can provide multi-line text in YAML, for example:

```
description: >
  This is how you can provide a longer description
  of your template that goes over several lines.
```

The *resources* section is required and must contain at least one resource definition. In the example above, a compute instance is defined with fixed values for the key_name, image and flavor parameters.

Note that all those elements, i.e. a key-pair with the given name, the image and the flavor have to exist in the OpenStack environment where the template is used. Typically a template is made more easily reusable, though, by defining a set of *input parameters* instead of hard-coding such values.

Template input parameters

Input parameters defined in the *parameters* section of a HOT template (see also *Parameters section*) allow users to customize a template during deployment. For example, this allows for providing custom key-pair names or image IDs to be used for a deployment. From a template authors perspective, this helps to make a template more easily reusable by avoiding hardcoded assumptions.

Sticking to the example used above, it makes sense to allow users to provide their custom key-pairs, provide their own image, and to select a flavor for the compute instance. This can be achieved by extending the initial template as follows:

```
heat_template_version: 2015-04-30

description: Simple template to deploy a single compute instance

parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
  image_id:
    type: string
    label: Image ID
```

(continues on next page)

(continued from previous page)

```
    description: Image to be used for compute instance
  instance_type:
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: { get_param: image_id }
      flavor: { get_param: instance_type }
```

In the example above, three input parameters have been defined that have to be provided by the user upon deployment. The fixed values for the respective resource properties have been replaced by references to the corresponding input parameters by means of the *get_param* function (see also *Intrinsic functions*).

You can also define default values for input parameters which will be used in case the user does not provide the respective parameter during deployment. For example, the following definition for the *instance_type* parameter would select the m1.small flavor unless specified otherwise by the user.

```
parameters:
  instance_type:
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used
    default: m1.small
```

Another option that can be specified for a parameter is to hide its value when users request information about a stack deployed from a template. This is achieved by the *hidden* attribute and useful, for example when requesting passwords as user input:

```
parameters:
  database_password:
    type: string
    label: Database Password
    description: Password to be used for database
    hidden: true
```

Restricting user input

In some cases you might want to restrict the values of input parameters that users can supply. For example, you might know that the software running in a compute instance needs a certain amount of resources so you might want to restrict the *instance_type* parameter introduced above. Parameters in HOT templates can be restricted by adding a *constraints* section (see also *Parameter Constraints*). For example, the following would allow only three values to be provided as input for the *instance_type* parameter:

```
parameters:
  instance_type:
```

(continues on next page)

(continued from previous page)

```
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used
    constraints:
      - allowed_values: [ m1.medium, m1.large, m1.xlarge ]
        description: Value must be one of m1.medium, m1.large or m1.xlarge.
```

The *constraints* section allows for defining a list of constraints that must all be fulfilled by user input. For example, the following list of constraints could be used to clearly specify format requirements on a password to be provided by users:

```
parameters:
  database_password:
    type: string
    label: Database Password
    description: Password to be used for database
    hidden: true
    constraints:
      - length: { min: 6, max: 8 }
        description: Password length must be between 6 and 8 characters.
      - allowed_pattern: "[a-zA-Z0-9]+"
        description: Password must consist of characters and numbers only.
      - allowed_pattern: "[A-Z]+[a-zA-Z0-9]*"
        description: Password must start with an uppercase character.
```

Note that you can define multiple constraints of the same type. Especially in the case of allowed patterns this not only allows for keeping regular expressions simple and maintainable, but also for keeping error messages to be presented to users precise.

Providing template outputs

In addition to template customization through input parameters, you will typically want to provide outputs to users, which can be done in the *outputs* section of a template (see also *Outputs section*). For example, the IP address by which the instance defined in the example above can be accessed should be provided to users. Otherwise, users would have to look it up themselves. The definition for providing the IP address of the compute instance as an output is shown in the following snippet:

```
outputs:
  instance_ip:
    description: The IP address of the deployed instance
    value: { get_attr: [my_instance, first_address] }
```

Output values are typically resolved using intrinsic function such as the *get_attr* function in the example above (see also *Intrinsic functions*).

Writing a hello world HOT template

HOT is a new template format meant to replace the CloudFormation-compatible format (CFN) as the native format supported by the Orchestration module over time. This guide is targeted towards template authors and explains how to write HOT templates based on examples. A detailed specification of HOT can be found at *Heat Orchestration Template (HOT) specification*.

This section gives an introduction on how to write HOT templates, starting from very basic steps and then going into more and more detail by means of examples.

A most basic template

The most basic template you can think of contains only a single resource definition using only predefined properties. For example, the template below could be used to deploy a single compute instance:

```
heat_template_version: 2015-04-30

description: Simple template to deploy a single compute instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: my_key
      image: ubuntu-trusty-x86_64
      flavor: m1.small
```

Each HOT template must include the `heat_template_version` key with the HOT version value, for example, `2013-05-23`. Consult the *Heat template version list* for allowed values and their features.

The `description` key is optional, however it is good practice to include some useful text that describes what users can do with the template. In case you want to provide a longer description that does not fit on a single line, you can provide multi-line text in YAML, for example:

```
description: >
  This is how you can provide a longer description
  of your template that goes over several lines.
```

The `resources` section is required and must contain at least one resource definition. In the above example, a compute instance is defined with fixed values for the `key_name`, `image` and `flavor` properties.

> **Note**
>
> All the defined elements (key pair, image, flavor) have to exist in the OpenStack environment where the template is used.

Input parameters

Input parameters defined in the `parameters` section of a template allow users to customize a template during deployment. For example, this allows for providing custom key pair names or image IDs to be used for a deployment. From a template authors perspective, this helps to make a template more easily reusable by avoiding hardcoded assumptions.

The following example extends the previous template to provide parameters for the key pair, image and flavor properties of the resource:

```
heat_template_version: 2015-04-30

description: Simple template to deploy a single compute instance
```

(continues on next page)

(continued from previous page)

```
parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
  image_id:
    type: string
    label: Image ID
    description: Image to be used for compute instance
  flavor:
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: { get_param: image_id }
      flavor: { get_param: flavor }
```

Values for the three parameters must be defined by the template user during the deployment of a stack. The `get_param` intrinsic function retrieves a user-specified value for a given parameter and uses this value for the associated resource property.

For more information about intrinsic functions, see *Intrinsic functions*.

Providing default values

You can provide default values for parameters. If a user doesnt define a value for a parameter, the default value is used during the stack deployment. The following example defines a default value `m1.small` for the `flavor` property:

```
parameters:
  flavor:
    type: string
    label: Instance Type
    description: Flavor to be used
    default: m1.small
```

Note

If a template doesnt define a default value for a parameter, then the user must define the value, otherwise the stack creation will fail.

Hiding parameters values

The values that a user provides when deploying a stack are available in the stack details and can be accessed by any user in the same tenant. To hide the value of a parameter, use the `hidden` boolean attribute of the parameter:

```
parameters:
  database_password:
    type: string
    label: Database Password
    description: Password to be used for database
    hidden: true
```

Restricting user input

You can restrict the values of an input parameter to make sure that the user defines valid data for this parameter. The `constraints` property of an input parameter defines a list of constraints to apply for the parameter. The following example restricts the `flavor` parameter to a list of three possible values:

```
parameters:
  flavor:
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used
    constraints:
      - allowed_values: [ m1.medium, m1.large, m1.xlarge ]
        description: Value must be one of m1.medium, m1.large or m1.xlarge.
```

The following example defines multiple constraints for a password definition:

```
parameters:
  database_password:
    type: string
    label: Database Password
    description: Password to be used for database
    hidden: true
    constraints:
      - length: { min: 6, max: 8 }
        description: Password length must be between 6 and 8 characters.
      - allowed_pattern: "[a-zA-Z0-9]+"
        description: Password must consist of characters and numbers only.
      - allowed_pattern: "[A-Z]+[a-zA-Z0-9]*"
        description: Password must start with an uppercase character.
```

The list of supported constraints is available in the *Parameter Constraints* section.

> **Note**
>
> You can define multiple constraints of the same type. Especially in the case of allowed patterns this not only allows for keeping regular expressions simple and maintainable, but also for keeping error messages to be presented to users precise.

Template outputs

In addition to template customization through input parameters, you can provide information about the resources created during the stack deployment to the users in the `outputs` section of a template. In the following example the output section provides the IP address of the `my_instance` resource:

```
outputs:
  instance_ip:
    description: The IP address of the deployed instance
    value: { get_attr: [my_instance, first_address] }
```

> **Note**
>
> Output values are typically resolved using intrinsic function such as the `get_attr`. See *Intrinsic functions* for more information about intrinsic functions..

See *Outputs section* for more information about the `outputs` section.

Guideline for features

Here are guideline for features:

Multi-Clouds support

Start from Stein release (version 12.0.0), Heat support multi-clouds orchestration. This document means to provide guideline for how to use multi-clouds features, and whats the environment requirement.

> **Note**
>
> If you like to create a stack in multi-region environment, you dont need this feature at all. all you need to do is provide *region_name* under *context* property for *OS::Heat::Stack*. If you like to see information on how to provide SSL support for your multi-region environment, you can jump to *Use CA cert (Optional)* .

Requirements

- **Barbican service** - For better security concerns, multi-cloud orchestration feature depends on Barbican service. So you have to make sure Barbican service is ready in your environment before you use this feature.
- **Access to remote Orchestration service** - Before you run your multi-cloud template. Make sure youre able to access to remote Orchestration service with correct endpoint information, legal access right, and ability to access to the remote site KeyStone, and Orchestration service API endpoint from local site. You need to make sure local Orchestration service is able to trigger and complete necessary API calls from local site to remote site. So we can complete stack actions without facing any access error.
- **Template complete resources/functions compatibility** - In your Orchestration template, you might want to use all kind of template functions or resource types as your template version and your Orchestration service allows. But please aware that once you plan to use Orchestration services across multiple OpenStack clouds, you have to also consider the compatibility. Make sure

the template version and resource types are ready to use before you ask remote site to run it. If you accidentally provide wrong template version (which not provided in remote site), you will get error message from remote site which prevent you from actually create remote resources. But its even better if we can just find such an error earlier.

Prepare

First of all, you need to put your remote cloud credential in a Barbican secret. To build your own multi-clouds stack, you need to build a Barbican secret first with most information for remote endpoint information.

Gathering credential information

Before we start generating secret, lets talk about what credential format we need. credential is a JSON format string contains two keys `auth_type`, and `auth`. `auth_type`, and `auth` following auth plugin loader rules from Keystone. You can find plugin options and authentication plugins in keystoneauth documents.

- **auth_type** - `auth_type` is a string for plugin name. Allows value like *v3applicationcredential*, *password*, *v3oidcclientcredentials*, etc. You need to provide *available plugins <plugin-options.html#available-plugins>*.
- **auth** - auth is a dictionary contains all parameters for plugins to perform authentication. You can find all valid parameter references from available plugins or get to all class path from plugin names for more detail allowed value or trace plugin class from there.

As you can tell, all allowed authentication plugins for credentials follows plugins keystoneauth rules. So once new change in keystoneauth, it will also directly reflect credentials too. Actually we just call keystoneauth to get plugin loader for remote authentication plugins. So keep an eye on keystoneauth if youre using this feature.

Validate your credential

Now you have all your credential information ready, try to validate first if you can. You can either directly test them via config, via CLI, or via keystoneauth sessions.

build credential secret

Once youre sure its valid, we can start building the secret out. To build a secret you just have to follow the standard Barbican CLI or API to store your secret.

The local site will read this secret to perform stack actions in remote site. Lets give a quick example here: Said you have two OpenStack cloud site A and site B. If you need to control site B from site A, make sure you have a secret with site Bs access information in site A. If you also like to control site A from site B, make sure you have a secret with site As access information in site B.

```
openstack secret store -n appcred --payload '{"auth_type":
↪"v3applicationcredential", "auth": {"auth_url": "{Keystone_URL}",
↪"application_credential_id": "{ID}", "application_credential_secret": "
↪{SECRET}"}}'
```

> **Note**
>
> One common error for JSON format is to use single quote() instead of double quote () inner your JSON schema.

Create remote stacks

Now, you have a secret id generated for your Barbican secret. Use that id as input for template.

To create a remote stack, you can simply use an *OS::Heat::Stack* resource in your template.

In resource properties, provide *credential_secret_id* (Barbican secret ID from the secret we just built for credential) under *context* property.

Here is an template example for you:

```
heat_template_version: rocky

resources:
  stack_in_remote_cloud:
    type: OS::Heat::Stack
    properties:
      context:
        credential_secret_id: {$Your_Secret_ID}
      template: { get_file: "remote-app.yaml" }
```

And thats all you need to do. The rest looks the same as usual.

Local Heat will read that secret, parse the credential information out, replace current authentication plugin in context, and make remote calls.

Heat will not store your credential information anywhere. so your secret security will remains within Barbican. That means if you wish to change your credential or make sure other people cant access to it. All you need to do is to update your Barbican secret or strong the security for it. But aware of this. If you plan to switch the credential content, make sure that wont affect resources/stacks in remote site. So do such actions with super care.

Use CA cert (Optional)

For production clouds, its very important to have SSL support. Here we provide CA cert method for your SSL access. If you wish to use that, use *ca_cert* under *context* property. Which *ca_cert* is the contents of a CA Certificate file that can be used to verify a remote cloud or regions server certificate. Or you can use *insecure* (a boolean option) under *context* property if you like to use insecure mode (For security concerns, dont do it!) and you dont want to use CA cert.

Here is an example for you:

```
heat_template_version: rocky

resources:
  stack_in_remote_cloud:
    type: OS::Heat::Stack
    properties:
```

(continues on next page)

(continued from previous page)

```
    context:
      credential_secret_id: {$Your_Secret_ID}
      ca_cert: {$Contents of a CA cert}
    template: { get_file: "remote-app.yaml" }
```

Note

If insecure flag is on, ca_cert will be ignored.

Heat Orchestration Template (HOT) specification

HOT is a new template format meant to replace the Heat CloudFormation-compatible format (CFN) as the native format supported by the Heat over time. This specification explains in detail all elements of the HOT template format. An example driven guide to writing HOT templates can be found at *Heat Orchestration Template (HOT) Guide*.

Status

HOT is considered reliable, supported, and standardized as of our Icehouse (April 2014) release. The Heat core team may make improvements to the standard, which very likely would be backward compatible. The template format is also versioned. Since Juno release, Heat supports multiple different versions of the HOT specification.

Template structure

HOT templates are defined in YAML and follow the structure outlined below.

```
heat_template_version: 2016-10-14

description:
  # a description of the template

parameter_groups:
  # a declaration of input parameter groups and order

parameters:
  # declaration of input parameters

resources:
  # declaration of template resources

outputs:
  # declaration of output parameters

conditions:
  # declaration of conditions
```

heat_template_version

This key with value `2013-05-23` (or a later date) indicates that the YAML document is a HOT template of the specified version.

description
This optional key allows for giving a description of the template, or the workload that can be deployed using the template.

parameter_groups
This section allows for specifying how the input parameters should be grouped and the order to provide the parameters in. This section is optional and can be omitted when necessary.

parameters
This section allows for specifying input parameters that have to be provided when instantiating the template. The section is optional and can be omitted when no input is required.

resources
This section contains the declaration of the single resources of the template. This section with at least one resource should be defined in any HOT template, or the template would not really do anything when being instantiated.

outputs
This section allows for specifying output parameters available to users once the template has been instantiated. This section is optional and can be omitted when no output values are required.

conditions
This optional section includes statements which can be used to restrict when a resource is created or when a property is defined. They can be associated with resources and resource properties in the `resources` section, also can be associated with outputs in the `outputs` sections of a template.

Note: Support for this section is added in the Newton version.

Heat template version

The value of `heat_template_version` tells Heat not only the format of the template but also features that will be validated and supported. Beginning with the Newton release, the version can be either the date of the Heat release or the code name of the Heat release. Heat currently supports the following values for the `heat_template_version` key:

2013-05-23

The key with value `2013-05-23` indicates that the YAML document is a HOT template and it may contain features implemented until the Icehouse release. This version supports the following functions (some are back ported to this version):

```
get_attr
get_file
get_param
get_resource
list_join
resource_facade
str_replace
Fn::Base64
Fn::GetAZs
Fn::Join
Fn::MemberListToMap
Fn::Replace
Fn::ResourceFacade
```

(continues on next page)

(continued from previous page)

```
Fn::Select
Fn::Split
Ref
```

2014-10-16

The key with value `2014-10-16` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Juno release. This version removes most CFN functions that were supported in the Icehouse release, i.e. the `2013-05-23` version. So the supported functions now are:

```
get_attr
get_file
get_param
get_resource
list_join
resource_facade
str_replace
Fn::Select
```

2015-04-30

The key with value `2015-04-30` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Kilo release. This version adds the `repeat` function. So the complete list of supported functions is:

```
get_attr
get_file
get_param
get_resource
list_join
repeat
digest
resource_facade
str_replace
Fn::Select
```

2015-10-15

The key with value `2015-10-15` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Liberty release. This version removes the *Fn::Select* function, path based `get_attr`/`get_param` references should be used instead. Moreover `get_attr` since this version returns dict of all attributes for the given resource excluding *show* attribute, if theres no <attribute name> specified, e.g. `{ get_attr: [<resource name>]}`. This version also adds the str_split function and support for passing multiple lists to the existing list_join function. The complete list of supported functions is:

```
get_attr
get_file
```

(continues on next page)

(continued from previous page)

```
get_param
get_resource
list_join
repeat
digest
resource_facade
str_replace
str_split
```

2016-04-08

The key with value `2016-04-08` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Mitaka release. This version also adds the `map_merge` function which can be used to merge the contents of maps. The complete list of supported functions is:

```
digest
get_attr
get_file
get_param
get_resource
list_join
map_merge
repeat
resource_facade
str_replace
str_split
```

2016-10-14 | newton

The key with value `2016-10-14` or `newton` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Newton release. This version adds the `yaql` function which can be used for evaluation of complex expressions, the `map_replace` function that can do key/value replacements on a mapping, and the `if` function which can be used to return corresponding value based on condition evaluation. The complete list of supported functions is:

```
digest
get_attr
get_file
get_param
get_resource
list_join
map_merge
map_replace
repeat
resource_facade
str_replace
str_split
yaql
if
```

This version adds `equals` condition function which can be used to compare whether two values are equal, the `not` condition function which acts as a NOT operator, the `and` condition function which acts as an AND operator to evaluate all the specified conditions, the `or` condition function which acts as an OR operator to evaluate all the specified conditions. The complete list of supported condition functions is:

```
equals
get_param
not
and
or
```

2017-02-24 | ocata

The key with value `2017-02-24` or `ocata` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Ocata release. This version adds the `str_replace_strict` function which raises errors for missing params and the `filter` function which filters out values from lists. The complete list of supported functions is:

```
digest
filter
get_attr
get_file
get_param
get_resource
list_join
map_merge
map_replace
repeat
resource_facade
str_replace
str_replace_strict
str_split
yaql
if
```

The complete list of supported condition functions is:

```
equals
get_param
not
and
or
```

2017-09-01 | pike

The key with value `2017-09-01` or `pike` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Pike release. This version adds the `make_url` function for assembling URLs, the `list_concat` function for combining multiple lists, the `list_concat_unique` function for combining multiple lists without repeating items, the `string_replace_vstrict` function which raises errors for missing and empty params, and the `contains` function which checks whether specific value is in a sequence. The complete list of supported

functions is:

```
digest
filter
get_attr
get_file
get_param
get_resource
list_join
make_url
list_concat
list_concat_unique
contains
map_merge
map_replace
repeat
resource_facade
str_replace
str_replace_strict
str_replace_vstrict
str_split
yaql
if
```

We support yaql and contains as condition functions in this version. The complete list of supported condition functions is:

```
equals
get_param
not
and
or
yaql
contains
```

2018-03-02 | queens

The key with value `2018-03-02` or `queens` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Queens release. The complete list of supported functions is:

```
digest
filter
get_attr
get_file
get_param
get_resource
list_join
make_url
list_concat
list_concat_unique
```

(continues on next page)

(continued from previous page)

```
contains
map_merge
map_replace
repeat
resource_facade
str_replace
str_replace_strict
str_replace_vstrict
str_split
yaql
if
```

The complete list of supported condition functions is:

```
equals
get_param
not
and
or
yaql
contains
```

2018-08-31 | rocky

The key with value `2018-08-31` or `rocky` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Rocky release. The complete list of supported functions is:

```
digest
filter
get_attr
get_file
get_param
get_resource
list_join
make_url
list_concat
list_concat_unique
contains
map_merge
map_replace
repeat
resource_facade
str_replace
str_replace_strict
str_replace_vstrict
str_split
yaql
if
```

The complete list of supported condition functions is:

```
equals
get_param
not
and
or
yaql
contains
```

2021-04-16 | wallaby

The key with value `2021-04-16` or `wallaby` indicates that the YAML document is a HOT template and it may contain features added and/or removed up until the Wallaby release.

This version adds a 2-argument variant of the `if` function. When the condition is false and no third argument is supplied, the entire enclosing item (which may be e.g. a list item, a key-value pair in a dict, or a property value) will be elided. This allows for e.g. conditional definition of properties while keeping the default value when the condition is false.

The complete list of supported functions is:

```
digest
filter
get_attr
get_file
get_param
get_resource
list_join
make_url
list_concat
list_concat_unique
contains
map_merge
map_replace
repeat
resource_facade
str_replace
str_replace_strict
str_replace_vstrict
str_split
yaql
if
```

The complete list of supported condition functions is:

```
equals
get_param
not
and
or
yaql
```

(continues on next page)

(continued from previous page)

```
contains
```

Parameter groups section

The `parameter_groups` section allows for specifying how the input parameters should be grouped and the order to provide the parameters in. These groups are typically used to describe expected behavior for downstream user interfaces.

These groups are specified in a list with each group containing a list of associated parameters. The lists are used to denote the expected order of the parameters. Each parameter should be associated to a specific group only once using the parameter name to bind it to a defined parameter in the `parameters` section.

```
parameter_groups:
- label: <human-readable label of parameter group>
  description: <description of the parameter group>
  parameters:
  - <param name>
  - <param name>
```

label
: A human-readable label that defines the associated group of parameters.

description
: This attribute allows for giving a human-readable description of the parameter group.

parameters
: A list of parameters associated with this parameter group.

param name
: The name of the parameter that is defined in the associated `parameters` section.

Parameters section

The `parameters` section allows for specifying input parameters that have to be provided when instantiating the template. Such parameters are typically used to customize each deployment (e.g. by setting custom user names or passwords) or for binding to environment-specifics like certain images.

Each parameter is specified in a separated nested block with the name of the parameters defined in the first line and additional attributes such as type or default value defined as nested elements.

```
parameters:
  <param name>:
    type: <string | number | json | comma_delimited_list | boolean>
    label: <human-readable name of the parameter>
    description: <description of the parameter>
    default: <default value for parameter>
    hidden: <true | false>
    constraints:
      <parameter constraints>
    immutable: <true | false>
    tags: <list of parameter categories>
```

param name
The name of the parameter.

type
The type of the parameter. Supported types are `string`, `number`, `comma_delimited_list`, `json` and `boolean`. This attribute is required.

label
A human readable name for the parameter. This attribute is optional.

description
A human readable description for the parameter. This attribute is optional.

default
A default value for the parameter. This value is used if the user doesnt specify his own value during deployment. This attribute is optional.

hidden
Defines whether the parameters should be hidden when a user requests information about a stack created from the template. This attribute can be used to hide passwords specified as parameters.

This attribute is optional and defaults to `false`.

constraints
A list of constraints to apply. The constraints are validated by the Orchestration engine when a user deploys a stack. The stack creation fails if the parameter value doesnt comply to the constraints. This attribute is optional.

immutable
Defines whether the parameter is updatable. Stack update fails, if this is set to `true` and the parameter value is changed. This attribute is optional and defaults to `false`.

tags
A list of strings to specify the category of a parameter. This value is used to categorize a parameter so that users can group the parameters. This attribute is optional.

The table below describes all currently supported types with examples:

Type	Description	Examples
string	A literal string.	String param
number	An integer or float.	2; 0.2
comma_de	An array of literal strings that are separated by commas. The total number of strings should be one more than the total number of commas.	[one, two]; one, two; Note: one, two returns [one, two]
json	A JSON-formatted map or list.	{key: value}
boolean	Boolean type value, which can be equal t, true, on, y, yes, or 1 for true value and f, false, off, n, no, or 0 for false value.	on; n

The following example shows a minimalistic definition of two parameters

```
parameters:
  user_name:
    type: string
    label: User Name
```

(continues on next page)

(continued from previous page)

```
    description: User name to be configured for the application
  port_number:
    type: number
    label: Port Number
    description: Port number to be configured for the web server
```

Note

The description and the label are optional, but defining these attributes is good practice to provide useful information about the role of the parameter to the user.

Parameter Constraints

The `constraints` block of a parameter definition defines additional validation constraints that apply to the value of the parameter. The parameter values provided by a user are validated against the constraints at instantiation time. The constraints are defined as a list with the following syntax

```
constraints:
  - <constraint type>: <constraint definition>
    description: <constraint description>
```

constraint type
: Type of constraint to apply. The set of currently supported constraints is given below.

constraint definition
: The actual constraint, depending on the constraint type. The concrete syntax for each constraint type is given below.

description
: A description of the constraint. The text is presented to the user when the value he defines violates the constraint. If omitted, a default validation message is presented to the user. This attribute is optional.

The following example shows the definition of a string parameter with two constraints. Note that while the descriptions for each constraint are optional, it is good practice to provide concrete descriptions to present useful messages to the user at deployment time.

```
parameters:
  user_name:
    type: string
    label: User Name
    description: User name to be configured for the application
    constraints:
      - length: { min: 6, max: 8 }
        description: User name must be between 6 and 8 characters
      - allowed_pattern: "[A-Z]+[a-zA-Z0-9]*"
        description: User name must start with an uppercase character
```

Note

> While the descriptions for each constraint are optional, it is good practice to provide concrete descriptions so useful messages can be presented to the user at deployment time.

The following sections list the supported types of parameter constraints, along with the concrete syntax for each type.

length

The `length` constraint applies to parameters of type `string`, `comma_delimited_list` and `json`.

It defines a lower and upper limit for the length of the string value or list/map collection.

The syntax of the `length` constraint is

```
length: { min: <lower limit>, max: <upper limit> }
```

It is possible to define a length constraint with only a lower limit or an upper limit. However, at least one of `min` or `max` must be specified.

range

The `range` constraint applies to parameters of type `number`. It defines a lower and upper limit for the numeric value of the parameter.

The syntax of the `range` constraint is

```
range: { min: <lower limit>, max: <upper limit> }
```

It is possible to define a range constraint with only a lower limit or an upper limit. However, at least one of `min` or `max` must be specified.

The minimum and maximum boundaries are included in the range. For example, the following range constraint would allow for all numeric values between 0 and 10

```
range: { min: 0, max: 10 }
```

modulo

The `modulo` constraint applies to parameters of type `number`. The value is valid if it is a multiple of `step`, starting with `offset`.

The syntax of the `modulo` constraint is

```
modulo: { step: <step>, offset: <offset> }
```

Both `step` and `offset` must be specified.

For example, the following modulo constraint would only allow for odd numbers

```
modulo: { step: 2, offset: 1 }
```

allowed_values

The `allowed_values` constraint applies to parameters of type `string` or `number`. It specifies a set of possible values for a parameter. At deployment time, the user-provided value for the respective parameter must match one of the elements of the list.

The syntax of the `allowed_values` constraint is

```
allowed_values: [ <value>, <value>, ... ]
```

Alternatively, the following YAML list notation can be used

```
allowed_values:
  - <value>
  - <value>
  - ...
```

For example

```
parameters:
  instance_type:
    type: string
    label: Instance Type
    description: Instance type for compute instances
    constraints:
      - allowed_values:
        - m1.small
        - m1.medium
        - m1.large
```

allowed_pattern

The `allowed_pattern` constraint applies to parameters of type `string`. It specifies a regular expression against which a user-provided parameter value must evaluate at deployment.

The syntax of the `allowed_pattern` constraint is

```
allowed_pattern: <regular expression>
```

For example

```
parameters:
  user_name:
    type: string
    label: User Name
    description: User name to be configured for the application
    constraints:
      - allowed_pattern: "[A-Z]+[a-zA-Z0-9]*"
        description: User name must start with an uppercase character
```

custom_constraint

The `custom_constraint` constraint adds an extra step of validation, generally to check that the specified resource exists in the backend. Custom constraints get implemented by plug-ins and can provide any kind of advanced constraint validation logic.

The syntax of the `custom_constraint` constraint is

```
custom_constraint: <name>
```

The `name` attribute specifies the concrete type of custom constraint. It corresponds to the name under which the respective validation plugin has been registered in the Orchestration engine.

For example

```
parameters:
  key_name
    type: string
    description: SSH key pair
    constraints:
      - custom_constraint: nova.keypair
```

The following section lists the custom constraints and the plug-ins that support them.

Name	Plug-in
barbican.container	heat.engine.clients.os.barbican:ContainerConstraint
barbican.secret	heat.engine.clients.os.barbican:SecretConstraint
blazar.reservation	heat.engine.clients.os.blazar:ReservationConstraint
cinder.backup	heat.engine.clients.os.cinder:VolumeBackupConstraint
cinder.qos_specs	heat.engine.clients.os.cinder:QoSSpecsConstraint
cinder.snapshot	heat.engine.clients.os.cinder:VolumeSnapshotConstraint
cinder.volume	heat.engine.clients.os.cinder:VolumeConstraint
cinder.vtype	heat.engine.clients.os.cinder:VolumeTypeConstraint
cron_expression	heat.engine.constraint.common_constraints:CRONExpressionConstraint
designate.zone	heat.engine.clients.os.designate:DesignateZoneConstraint
dns_domain	heat.engine.constraint.common_constraints:DNSDomainConstraint
dns_name	heat.engine.constraint.common_constraints:DNSNameConstraint
expiration	heat.engine.constraint.common_constraints:ExpirationConstraint
glance.image	heat.engine.clients.os.glance:ImageConstraint
ip_addr	heat.engine.constraint.common_constraints:IPConstraint
ip_or_cidr	heat.engine.constraint.common_constraints:IPCIDRConstraint
ironic.node	heat.engine.clients.os.ironic:NodeConstraint
ironic.portgroup	heat.engine.clients.os.ironic:PortGroupConstraint
iso_8601	heat.engine.constraint.common_constraints:ISO8601Constraint
json_string	heat.engine.constraint.common_constraints:JsonStringConstraint
keystone.domain	heat.engine.clients.os.keystone.keystone_constraints:KeystoneDomainConstraint
keystone.group	heat.engine.clients.os.keystone.keystone_constraints:KeystoneGroupConstraint
keystone.project	heat.engine.clients.os.keystone.keystone_constraints:KeystoneProjectConstraint
keystone.region	heat.engine.clients.os.keystone.keystone_constraints:KeystoneRegionConstraint
keystone.role	heat.engine.clients.os.keystone.keystone_constraints:KeystoneRoleConstraint
keystone.service	heat.engine.clients.os.keystone.keystone_constraints:KeystoneServiceConstraint

continues on next page

Table 1 – continued from previous page

Name	Plug-in
keystone.user	heat.engine.clients.os.keystone.keystone_constraints:KeystoneUserConstraint
mac_addr	heat.engine.constraint.common_constraints:MACConstraint
magnum.cluster_template	heat.engine.clients.os.magnum:ClusterTemplateConstraint
manila.share_network	heat.engine.clients.os.manila:ManilaShareNetworkConstraint
manila.share_snapshot	heat.engine.clients.os.manila:ManilaShareSnapshotConstraint
manila.share_type	heat.engine.clients.os.manila:ManilaShareTypeConstraint
mistral.workflow	heat.engine.clients.os.mistral:WorkflowConstraint
net_cidr	heat.engine.constraint.common_constraints:CIDRConstraint
neutron.address_scope	heat.engine.clients.os.neutron.neutron_constraints:AddressScopeConstraint
neutron.flow_classifier	heat.engine.clients.os.neutron.neutron_constraints:FlowClassifierConstraint
neutron.lbaas.listener	heat.engine.clients.os.neutron.lbaas_constraints:ListenerConstraint
neutron.lbaas.loadbalancer	heat.engine.clients.os.neutron.lbaas_constraints:LoadbalancerConstraint
neutron.lbaas.pool	heat.engine.clients.os.neutron.lbaas_constraints:PoolConstraint
neutron.lbaas.provider	heat.engine.clients.os.neutron.lbaas_constraints:LBaasV2ProviderConstraint
neutron.network	heat.engine.clients.os.neutron.neutron_constraints:NetworkConstraint
neutron.port	heat.engine.clients.os.neutron.neutron_constraints:PortConstraint
neutron.port_pair	heat.engine.clients.os.neutron.neutron_constraints:PortPairConstraint
neutron.port_pair_group	heat.engine.clients.os.neutron.neutron_constraints:PortPairGroupConstraint
neutron.qos_policy	heat.engine.clients.os.neutron.neutron_constraints:QoSPolicyConstraint
neutron.router	heat.engine.clients.os.neutron.neutron_constraints:RouterConstraint
neutron.security_group	heat.engine.clients.os.neutron.neutron_constraints:SecurityGroupConstraint
neutron.segment	heat.engine.clients.os.openstacksdk:SegmentConstraint
neutron.subnet	heat.engine.clients.os.neutron.neutron_constraints:SubnetConstraint
neutron.subnetpool	heat.engine.clients.os.neutron.neutron_constraints:SubnetPoolConstraint
neutron.taas.tap_flow	heat.engine.clients.os.neutron.taas_constraints:TapFlowConstraint
neutron.taas.tap_service	heat.engine.clients.os.neutron.taas_constraints:TapServiceConstraint
nova.flavor	heat.engine.clients.os.nova:FlavorConstraint
nova.host	heat.engine.clients.os.nova:HostConstraint
nova.keypair	heat.engine.clients.os.nova:KeypairConstraint
nova.network	heat.engine.constraint.common_constraints:TestConstraintDelay
nova.server	heat.engine.clients.os.nova:ServerConstraint
octavia.availabilityzone	heat.engine.clients.os.octavia:AvailabilityZoneConstraint
octavia.availabilityzoneprofile	heat.engine.clients.os.octavia:AvailabilityZoneProfileConstraint
octavia.flavor	heat.engine.clients.os.octavia:FlavorConstraint
octavia.flavorprofile	heat.engine.clients.os.octavia:FlavorProfileConstraint
octavia.l7policy	heat.engine.clients.os.octavia:L7PolicyConstraint
octavia.listener	heat.engine.clients.os.octavia:ListenerConstraint
octavia.loadbalancer	heat.engine.clients.os.octavia:LoadbalancerConstraint
octavia.pool	heat.engine.clients.os.octavia:PoolConstraint
rel_dns_name	heat.engine.constraint.common_constraints:RelativeDNSNameConstraint
test_constr	heat.engine.constraint.common_constraints:TestConstraintDelay
timezone	heat.engine.constraint.common_constraints:TimezoneConstraint
trove.flavor	heat.engine.clients.os.trove:FlavorConstraint
zaqar.queue	heat.engine.clients.os.zaqar:QueueConstraint

Pseudo parameters

In addition to parameters defined by a template author, Heat also creates three parameters for every stack that allow referential access to the stacks name, stacks identifier and projects identifier. These parameters are named `OS::stack_name` for the stack name, `OS::stack_id` for the stack identifier and `OS::project_id` for the project identifier. These values are accessible via the *get_param* intrinsic function, just like user-defined parameters.

> **Note**
>
> `OS::project_id` is available since 2015.1 (Kilo).

Resources section

The `resources` section defines actual resources that make up a stack deployed from the HOT template (for instance compute instances, networks, storage volumes).

Each resource is defined as a separate block in the `resources` section with the following syntax

```
resources:
  <resource ID>:
    type: <resource type>
    properties:
      <property name>: <property value>
    metadata:
      <resource specific metadata>
    depends_on: <resource ID or list of ID>
    update_policy: <update policy>
    deletion_policy: <deletion policy>
    external_id: <external resource ID>
    condition: <condition name or expression or boolean>
```

resource ID
: A resource ID which must be unique within the `resources` section of the template.

type
: The resource type, such as `OS::Nova::Server` or `OS::Neutron::Port`. This attribute is required.

properties
: A list of resource-specific properties. The property value can be provided in place, or via a function (see *Intrinsic functions*). This section is optional.

metadata
: Resource-specific metadata. This section is optional.

depends_on
: Dependencies of the resource on one or more resources of the template. See *Resource dependencies* for details. This attribute is optional.

update_policy
: Update policy for the resource, in the form of a nested dictionary. Whether update policies are supported and what the exact semantics are depends on the type of the current resource. This attribute is optional.

deletion_policy
Deletion policy for the resource. The allowed deletion policies are `Delete`, `Retain`, and `Snapshot`. Beginning with `heat_template_version` `2016-10-14`, the lowercase equivalents `delete`, `retain`, and `snapshot` are also allowed. This attribute is optional; the default policy is to delete the physical resource when deleting a resource from the stack.

external_id
Allows for specifying the resource_id for an existing external (to the stack) resource. External resources can not depend on other resources, but we allow other resources depend on external resource. This attribute is optional. Note: when this is specified, properties will not be used for building the resource and the resource is not managed by Heat. This is not possible to update that attribute. Also resource wont be deleted by heat when stack is deleted.

condition
Condition for the resource. Which decides whether to create the resource or not. This attribute is optional.

Note: Support `condition` for resource is added in the Newton version.

Depending on the type of resource, the resource block might include more resource specific data.

All resource types that can be used in CFN templates can also be used in HOT templates, adapted to the YAML structure as outlined above.

The following example demonstrates the definition of a simple compute resource with some fixed property values

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: F18-x86_64-cfntools
```

Resource dependencies

The `depends_on` attribute of a resource defines a dependency between this resource and one or more other resources.

If a resource depends on just one other resource, the ID of the other resource is specified as string of the `depends_on` attribute, as shown in the following example

```
resources:
  server1:
    type: OS::Nova::Server
    depends_on: server2

  server2:
    type: OS::Nova::Server
```

If a resource depends on more than one other resources, the value of the `depends_on` attribute is specified as a list of resource IDs, as shown in the following example

```
resources:
  server1:
    type: OS::Nova::Server
    depends_on: [ server2, server3 ]

  server2:
    type: OS::Nova::Server

  server3:
    type: OS::Nova::Server
```

Outputs section

The `outputs` section defines output parameters that should be available to the user after a stack has been created. This would be, for example, parameters such as IP addresses of deployed instances, or URLs of web applications deployed as part of a stack.

Each output parameter is defined as a separate block within the outputs section according to the following syntax

```
outputs:
  <parameter name>:
    description: <description>
    value: <parameter value>
    condition: <condition name or expression or boolean>
```

parameter name
: The output parameter name, which must be unique within the `outputs` section of a template.

description
: A short description of the output parameter. This attribute is optional.

parameter value
: The value of the output parameter. This value is usually resolved by means of a function. See *Intrinsic functions* for details about the functions. This attribute is required.

condition
: To conditionally define an output value. None value will be shown if the condition is False. This attribute is optional.

  Note: Support `condition` for output is added in the Newton version.

The example below shows how the IP address of a compute resource can be defined as an output parameter

```
outputs:
  instance_ip:
    description: IP address of the deployed compute instance
    value: { get_attr: [my_instance, first_address] }
```

Conditions section

The `conditions` section defines one or more conditions which are evaluated based on input parameter values provided when a user creates or updates a stack. The condition can be associated with resources, resource properties and outputs. For example, based on the result of a condition, user can conditionally create resources, user can conditionally set different values of properties, and user can conditionally give outputs of a stack.

The `conditions` section is defined with the following syntax

```
conditions:
  <condition name1>: {expression1}
  <condition name2>: {expression2}
  ...
```

condition name
: The condition name, which must be unique within the `conditions` section of a template.

expression
: The expression which is expected to return True or False. Usually, the condition functions can be used as expression to define conditions:

```
equals
get_param
not
and
or
yaql
```

Note: In condition functions, you can reference a value from an input parameter, but you cannot reference resource or its attribute. We support referencing other conditions (by condition name) in condition functions. We support yaql as condition function in the Pike version.

An example of conditions section definition

```
conditions:
  cd1: True
  cd2:
    get_param: param1
  cd3:
    equals:
    - get_param: param2
    - yes
  cd4:
    not:
      equals:
      - get_param: param3
      - yes
  cd5:
    and:
    - equals:
      - get_param: env_type
      - prod
```

(continues on next page)

(continued from previous page)

```
    - not:
        equals:
        - get_param: zone
        - beijing
  cd6:
    or:
    - equals:
      - get_param: zone
      - shanghai
    - equals:
      - get_param: zone
      - beijing
  cd7:
    not: cd4
  cd8:
    and:
    - cd1
    - cd2
  cd9:
    yaql:
      expression: $.data.services.contains('heat')
      data:
        services:
          get_param: ServiceNames
  cd10:
    contains:
    - 'neutron'
    - get_param: ServiceNames
```

The example below shows how to associate condition with resources

```
parameters:
  env_type:
    default: test
    type: string
conditions:
  create_prod_res: {equals : [{get_param: env_type}, "prod"]}
resources:
  volume:
    type: OS::Cinder::Volume
    condition: create_prod_res
    properties:
      size: 1
```

The create_prod_res condition evaluates to true if the env_type parameter is equal to prod. In the above sample template, the volume resource is associated with the create_prod_res condition. Therefore, the volume resource is created only if the env_type is equal to prod.

The example below shows how to conditionally define an output

```
outputs:
  vol_size:
    value: {get_attr: [my_volume, size]}
    condition: create_prod_res
```

In the above sample template, the vol_size output is associated with the create_prod_res condition. Therefore, the vol_size output is given corresponding value only if the env_type is equal to prod, otherwise the value of the output is None.

Intrinsic functions

HOT provides a set of intrinsic functions that can be used inside templates to perform specific tasks, such as getting the value of a resource attribute at runtime. The following section describes the role and syntax of the intrinsic functions.

Note: these functions can only be used within the properties section of each resource or in the outputs section.

get_attr

The `get_attr` function references an attribute of a resource. The attribute value is resolved at runtime using the resource instance created from the respective resource definition.

Path based attribute referencing using keys or indexes requires `heat_template_version` `2014-10-16` or higher.

The syntax of the `get_attr` function is

```
get_attr:
  - <resource name>
  - <attribute name>
  - <key/index 1> (optional)
  - <key/index 2> (optional)
  - ...
```

resource name
: The resource name for which the attribute needs to be resolved.

  The resource name must exist in the `resources` section of the template.

attribute name
: The attribute name to be resolved. If the attribute returns a complex data structure such as a list or a map, then subsequent keys or indexes can be specified. These additional parameters are used to navigate the data structure to return the desired value.

The following example demonstrates how to use the `get_attr` function:

```
resources:
  my_instance:
    type: OS::Nova::Server
    # ...

outputs:
  instance_ip:
```

(continues on next page)

(continued from previous page)

```
    description: IP address of the deployed compute instance
    value: { get_attr: [my_instance, first_address] }
  instance_private_ip:
    description: Private IP address of the deployed compute instance
    value: { get_attr: [my_instance, networks, private, 0] }
```

In this example, if the `networks` attribute contained the following data:

```
{"public": ["2001:0db8:0000:0000:0000:ff00:0042:8329", "1.2.3.4"],
 "private": ["10.0.0.1"]}
```

then the value of `get_attr` function would resolve to `10.0.0.1` (first item of the `private` entry in the `networks` map).

From `heat_template_version`: 2015-10-15 <attribute_name> is optional and if <attribute_name> is not specified, `get_attr` returns dict of all attributes for the given resource excluding *show* attribute. In this case syntax would be next:

```
get_attr:
  - <resource_name>
```

get_file

The `get_file` function returns the content of a file into the template. It is generally used as a file inclusion mechanism for files containing scripts or configuration files.

The syntax of `get_file` function is

```
get_file: <content key>
```

The `content key` is used to look up the `files` dictionary that is provided in the REST API call. The Orchestration client command (`heat`) is `get_file` aware and populates the `files` dictionary with the actual content of fetched paths and URLs. The Orchestration client command supports relative paths and transforms these to the absolute URLs required by the Orchestration API.

> **Note**
>
> The `get_file` argument must be a static path or URL and not rely on intrinsic functions like `get_param`. the Orchestration client does not process intrinsic functions (they are only processed by the Orchestration engine).

The example below demonstrates the `get_file` function usage with both relative and absolute URLs

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      # general properties ...
      user_data:
        get_file: my_instance_user_data.sh
```

(continues on next page)

(continued from previous page)

```
  my_other_instance:
    type: OS::Nova::Server
    properties:
      # general properties ...
      user_data:
        get_file: http://example.com/my_other_instance_user_data.sh
```

The `files` dictionary generated by the Orchestration client during instantiation of the stack would contain the following keys:

- `file:///path/to/my_instance_user_data.sh`
- `http://example.com/my_other_instance_user_data.sh`

get_param

The `get_param` function references an input parameter of a template. It resolves to the value provided for this input parameter at runtime.

The syntax of the `get_param` function is

```
get_param:
 - <parameter name>
 - <key/index 1> (optional)
 - <key/index 2> (optional)
 - ...
```

parameter name
: The parameter name to be resolved. If the parameters returns a complex data structure such as a list or a map, then subsequent keys or indexes can be specified. These additional parameters are used to navigate the data structure to return the desired value.

The following example demonstrates the use of the `get_param` function

```
parameters:
  instance_type:
    type: string
    label: Instance Type
    description: Instance type to be used.
  server_data:
    type: json

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      flavor: { get_param: instance_type}
      metadata: { get_param: [ server_data, metadata ] }
      key_name: { get_param: [ server_data, keys, 0 ] }
```

In this example, if the `instance_type` and `server_data` parameters contained the following data:

```
{"instance_type": "m1.tiny",
{"server_data": {"metadata": {"foo": "bar"},
                 "keys": ["a_key","other_key"]}}}
```

then the value of the property `flavor` would resolve to `m1.tiny`, `metadata` would resolve to `{"foo": "bar"}` and `key_name` would resolve to `a_key`.

get_resource

The `get_resource` function references another resource within the same template. At runtime, it is resolved to reference the ID of the referenced resource, which is resource type specific. For example, a reference to a floating IP resource returns the respective IP address at runtime. The syntax of the `get_resource` function is

```
get_resource: <resource ID>
```

The resource ID of the referenced resource is given as single parameter to the `get_resource` function.

For example

```
resources:
  instance_port:
    type: OS::Neutron::Port
    properties: ...

  instance:
    type: OS::Nova::Server
    properties:
      ...
      networks:
        port: { get_resource: instance_port }
```

list_join

The `list_join` function joins a list of strings with the given delimiter.

The syntax of the `list_join` function is

```
list_join:
- <delimiter>
- <list to join>
```

For example

```
list_join: [', ', ['one', 'two', 'and three']]
```

This resolve to the string `one, two, and three`.

From HOT version `2015-10-15` you may optionally pass additional lists, which will be appended to the previous lists to join.

For example:

```
list_join: [', ', ['one', 'two'], ['three', 'four']]
```

This resolve to the string `one, two, three, four`.

From HOT version `2015-10-15` you may optionally also pass non-string list items (e.g json/map/list parameters or attributes) and they will be serialized as json before joining.

digest

The `digest` function allows for performing digest operations on a given value. This function has been introduced in the Kilo release and is usable with HOT versions later than `2015-04-30`.

The syntax of the `digest` function is

```
digest:
  - <algorithm>
  - <value>
```

algorithm
: The digest algorithm. Valid algorithms are the ones provided natively by hashlib (md5, sha1, sha224, sha256, sha384, and sha512) or any one provided by OpenSSL.

value
: The value to digest. This function will resolve to the corresponding hash of the value.

For example

```
# from a user supplied parameter
pwd_hash: { digest: ['sha512', { get_param: raw_password }] }
```

The value of the digest function would resolve to the corresponding hash of the value of `raw_password`.

repeat

The `repeat` function allows for dynamically transforming lists by iterating over the contents of one or more source lists and replacing the list elements into a template. The result of this function is a new list, where the elements are set to the template, rendered for each list item.

The syntax of the `repeat` function is

```
repeat:
  template:
    <template>
  for_each:
    <var>: <list>
```

template
: The `template` argument defines the content generated for each iteration, with placeholders for the elements that need to be replaced at runtime. This argument can be of any supported type.

for_each
: The `for_each` argument is a dictionary that defines how to generate the repetitions of the template and perform substitutions. In this dictionary the keys are the placeholder names that will be replaced in the template, and the values are the lists to iterate on. On each iteration, the function will render the template by performing substitution with elements of the given lists. If a single

key/value pair is given in this argument, the template will be rendered once for each element in the list. When more than one key/value pairs are given, the iterations will be performed on all the permutations of values between the given lists. The values in this dictionary can be given as functions such as `get_attr` or `get_param`.

The following example shows how a security group resource can be defined to include a list of ports given as a parameter

```
parameters:
  ports:
    type: comma_delimited_list
    label: ports
    default: "80,443,8080"

resources:
  security_group:
    type: OS::Neutron::SecurityGroup
    properties:
      name: web_server_security_group
      rules:
        repeat:
          for_each:
            <%port%>: { get_param: ports }
          template:
            protocol: tcp
            port_range_min: <%port%>
            port_range_max: <%port%>
```

The following example demonstrates how the use of multiple lists enables the security group to also include parameterized protocols

```
parameters:
  ports:
    type: comma_delimited_list
    label: ports
    default: "80,443,8080"
  protocols:
    type: comma_delimited_list
    label: protocols
    default: "tcp,udp"

resources:
  security_group:
    type: OS::Neutron::SecurityGroup
    properties:
      name: web_server_security_group
      rules:
        repeat:
          for_each:
            <%port%>: { get_param: ports }
            <%protocol%>: { get_param: protocols }
```

(continues on next page)

(continued from previous page)

```
          template:
            protocol: <%protocol%>
            port_range_min: <%port%>
```

Note how multiple entries in the `for_each` argument are equivalent to nested for-loops in most programming languages.

From HOT version `2016-10-14` you may also pass a map as value for the `for_each` key, in which case the list of map keys will be used as value.

From HOT version `2017-09-01` (or pike) you may specify a argument `permutations` to decide whether to iterate nested the over all the permutations of the elements in the given lists. If permutations is not specified, we set the default value to true to compatible with before behavior. The args have to be lists instead of dicts if permutations is False because keys in a dict are unordered, and the list args all have to be of the same length.

```
parameters:
  subnets:
    type: comma_delimited_list
    label: subnets
    default: "sub1, sub2"
  networks:
    type: comma_delimited_list
    label: networks
    default: "net1, net2"

resources:
  my_server:
    type: OS::Nova:Server
    properties:
      networks:
        repeat:
          for_each:
            <%sub%>: { get_param: subnets }
            <%net%>: { get_param: networks }
          template:
            subnet: <%sub%>
            network: <%net%>
          permutations: false
```

After resolved, we will get the networks of server like: [{subnet: sub1, network: net1}, {subnet: sub2, network: net2}]

resource_facade

The `resource_facade` function retrieves data in a parent provider template.

A provider template provides a custom definition of a resource, called its facade. For more information about custom templates, see *Template composition*. The syntax of the `resource_facade` function is

```
resource_facade: <data type>
```

`data type` can be one of `metadata`, `deletion_policy` or `update_policy`.

str_replace

The `str_replace` function dynamically constructs strings by providing a template string with placeholders and a list of mappings to assign values to those placeholders at runtime. The placeholders are replaced with mapping values wherever a mapping key exactly matches a placeholder.

The syntax of the `str_replace` function is

```
str_replace:
  template: <template string>
  params: <parameter mappings>
```

template
: Defines the template string that contains placeholders which will be substituted at runtime.

params
: Provides parameter mappings in the form of dictionary. Each key refers to a placeholder used in the `template` attribute. From HOT version `2015-10-15` you may optionally pass non-string parameter values (e.g json/map/list parameters or attributes) and they will be serialized as json before replacing, prior heat/HOT versions require string values.

The following example shows a simple use of the `str_replace` function in the outputs section of a template to build a URL for logging into a deployed application

```
resources:
  my_instance:
    type: OS::Nova::Server
    # general metadata and properties ...

outputs:
  Login_URL:
    description: The URL to log into the deployed application
    value:
      str_replace:
        template: http://host/MyApplication
        params:
          host: { get_attr: [ my_instance, first_address ] }
```

The following examples show the use of the `str_replace` function to build an instance initialization script

```
parameters:
  DBRootPassword:
    type: string
    label: Database Password
    description: Root password for MySQL
    hidden: true

resources:
  my_instance:
    type: OS::Nova::Server
```

(continues on next page)

(continued from previous page)

```
    properties:
      # general properties ...
      user_data:
        str_replace:
          template: |
            #!/bin/bash
            echo "Hello world"
            echo "Setting MySQL root password"
            mysqladmin -u root password $db_rootpassword
            # do more things ...
          params:
            $db_rootpassword: { get_param: DBRootPassword }
```

In the example above, one can imagine that MySQL is being configured on a compute instance and the root password is going to be set based on a user provided parameter. The script for doing this is provided as userdata to the compute instance, leveraging the `str_replace` function.

str_replace_strict

`str_replace_strict` behaves identically to the `str_replace` function, only an error is raised if any of the params are not present in the template. This may help catch typos or other issues sooner rather than later when processing a template.

str_replace_vstrict

`str_replace_vstrict` behaves identically to the `str_replace_strict` function, only an error is raised if any of the params are empty. This may help catch issues (i.e., prevent resources from being created with bogus values) sooner rather than later if it is known that all the params should be non-empty.

str_split

The `str_split` function allows for splitting a string into a list by providing an arbitrary delimiter, the opposite of `list_join`.

The syntax of the `str_split` function is as follows:

```
str_split:
  - ','
  - string,to,split
```

Or:

```
str_split: [',', 'string,to,split']
```

The result of which is:

```
['string', 'to', 'split']
```

Optionally, an index may be provided to select a specific entry from the resulting list, similar to `get_attr/get_param`:

```
str_split: [',', 'string,to,split', 0]
```

The result of which is:

```
'string'
```

Note: The index starts at zero, and any value outside the maximum (e.g the length of the list minus one) will cause an error.

map_merge

The `map_merge` function merges maps together. Values in the latter maps override any values in earlier ones. Can be very useful when composing maps that contain configuration data into a single consolidated map.

The syntax of the `map_merge` function is

```
map_merge:
- <map 1>
- <map 2>
- ...
```

For example

```
map_merge: [{'k1': 'v1', 'k2': 'v2'}, {'k1': 'v2'}]
```

This resolves to a map containing `{'k1': 'v2', 'k2': 'v2'}`.

Maps containing no items resolve to `{}`.

map_replace

The `map_replace` function does key/value replacements on an existing mapping. An input mapping is processed by iterating over all keys/values and performing a replacement if an exact match is found in either of the optional keys/values mappings.

The syntax of the `map_replace` function is

```
map_replace:
- <input map>
- keys: <map of key replacements>
  values: <map of value replacements>
```

For example

```
map_replace:
- k1: v1
  k2: v2
- keys:
    k1: K1
  values:
    v2: V2
```

This resolves to a map containing `{'K1': 'v1', 'k2': 'V2'}`.

The keys/values mappings are optional, either or both may be specified.

Note that an error is raised if a replacement defined in keys results in a collision with an existing keys in the input or output map.

Also note that while unhashable values (e.g lists) in the input map are valid, they will be ignored by the values replacement, because no key can be defined in the values mapping to define their replacement.

yaql

The `yaql` evaluates yaql expression on a given data.

The syntax of the `yaql` function is

```
yaql:
  expression: <expression>
  data: <data>
```

For example

```
parameters:
  list_param:
    type: comma_delimited_list
    default: [1, 2, 3]

outputs:
  max_elem:
    value:
      yaql:
        expression: $.data.list_param.select(int($)).max()
        data:
          list_param: {get_param: list_param}
```

max_elem output will be evaluated to 3

equals

The `equals` function compares whether two values are equal.

The syntax of the `equals` function is

```
equals: [value_1, value_2]
```

The value can be any type that you want to compare. This function returns true if the two values are equal or false if they arent.

For example

```
equals: [{get_param: env_type}, 'prod']
```

If param env_type equals to prod, this function returns true, otherwise returns false.

if

The `if` function returns the corresponding value based on the evaluation of a condition.

The syntax of the `if` function is

```
if: [condition_name, value_if_true, value_if_false]
```

For example

```
conditions:
  create_prod_res: {equals : [{get_param: env_type}, "prod"]}

resources:
  test_server:
    type: OS::Nova::Server
    properties:
      name: {if: ["create_prod_res", "s_prod", "s_test"]}
```

The name property is set to s_prod if the condition create_prod_res evaluates to true (if parameter env_type is prod), and is set to s_test if the condition create_prod_res evaluates to false (if parameter env_type isnt prod).

Note: You define all conditions in the `conditions` section of a template except for `if` conditions. You can use the `if` condition in the property values in the `resources` section and `outputs` sections of a template.

Beginning with the `wallaby` template version, the third argument is optional. If only two arguments are passed, the entire enclosing item is removed when the condition is false.

For example:

```
conditions:
  override_name: {not: {equals: [{get_param: server_name}, ""]}}

resources:
  test_server:
    type: OS::Nova::Server
    properties:
      name: {if: [override_name, {get_param: server_name}]}
```

In this example, the default name for the server (which is generated by Heat when the property value is not specified) would be used when the `server_name` parameter value is an empty string.

not

The `not` function acts as a NOT operator.

The syntax of the `not` function is

```
not: condition
```

Note: A condition can be an expression such as `equals`, `or` and `and` that evaluates to true or false, can be a boolean, and can be other condition name defined in `conditions` section of template.

Returns true for a condition that evaluates to false or returns false for a condition that evaluates to true.

For example

```
not:
  equals:
  - get_param: env_type
  - prod
```

If param env_type equals to prod, this function returns false, otherwise returns true.

Another example with boolean value definition

```
not: True
```

This function returns false.

Another example reference other condition name

```
not: my_other_condition
```

This function returns false if my_other_condition evaluates to true, otherwise returns true.

and

The `and` function acts as an AND operator to evaluate all the specified conditions.

The syntax of the `and` function is

```
and: [{condition_1}, {condition_2}, ... {condition_n}]
```

Note: A condition can be an expression such as `equals`, `or` and `not` that evaluates to true or false, can be a boolean, and can be other condition names defined in `conditions` section of template.

Returns true if all the specified conditions evaluate to true, or returns false if any one of the conditions evaluates to false.

For example

```
and:
- equals:
  - get_param: env_type
  - prod
- not:
    equals:
    - get_param: zone
    - beijing
```

If param env_type equals to prod, and param zone is not equal to beijing, this function returns true, otherwise returns false.

Another example reference with other conditions

```
and:
- other_condition_1
- other_condition_2
```

This function returns true if other_condition_1 and other_condition_2 evaluate to true both, otherwise returns false.

or

The `or` function acts as an OR operator to evaluate all the specified conditions.

The syntax of the `or` function is

```
or: [{condition_1}, {condition_2}, ... {condition_n}]
```

Note: A condition can be an expression such as `equals`, `and` and `not` that evaluates to true or false, can be a boolean, and can be other condition names defined in `conditions` section of template.

Returns true if any one of the specified conditions evaluate to true, or returns false if all of the conditions evaluates to false.

For example

```
or:
- equals:
  - get_param: env_type
  - prod
- not:
    equals:
    - get_param: zone
    - beijing
```

If param env_type equals to prod, or the param zone is not equal to beijing, this function returns true, otherwise returns false.

Another example reference other conditions

```
or:
- other_condition_1
- other_condition_2
```

This function returns true if any one of other_condition_1 or other_condition_2 evaluate to true, otherwise returns false.

filter

The `filter` function removes values from lists.

The syntax of the `filter` function is

```
filter:
  - <values>
  - <list>
```

For example

```
parameters:
  list_param:
```

(continues on next page)

(continued from previous page)

```
    type: comma_delimited_list
    default: [1, 2, 3]

outputs:
  output_list:
    value:
      filter:
        - [3]
        - {get_param: list_param}
```

output_list will be evaluated to [1, 2].

make_url

The `make_url` function builds URLs.

The syntax of the `make_url` function is

```
make_url:
  scheme: <protocol>
  username: <username>
  password: <password>
  host: <hostname or IP>
  port: <port>
  path: <path>
  query:
    <key1>: <value1>
    <key2>: <value2>
  fragment: <fragment>
```

All parameters are optional.

For example

```
outputs:
  server_url:
    value:
      make_url:
        scheme: http
        host: {get_attr: [server, networks, <network_name>, 0]}
        port: 8080
        path: /hello
        query:
          recipient: world
        fragment: greeting
```

`server_url` will be evaluated to a URL in the form:

```
http://[<server IP>]:8080/hello?recipient=world#greeting
```

list_concat

The `list_concat` function concatenates lists together.

The syntax of the `list_concat` function is

```
list_concat:
  - <list #1>
  - <list #2>
  - ...
```

For example

```
list_concat: [['v1', 'v2'], ['v3', 'v4']]
```

Will resolve to the list `['v1', 'v2', 'v3', 'v4']`.

Null values will be ignored.

list_concat_unique

The `list_concat_unique` function behaves identically to the function `list_concat`, only removes the repeating items of lists.

For example

```
list_concat_unique: [['v1', 'v2'], ['v2', 'v3']]
```

Will resolve to the list `['v1', 'v2', 'v3']`.

contains

The `contains` function checks whether the specific value is in a sequence.

The syntax of the `contains` function is

```
contains: [<value>, <sequence>]
```

This function returns true if value is in sequence or false if it isnt.

For example

```
contains: ['v1', ['v1', 'v2', 'v3']]
```

Will resolve to boolean true.

Instances

Manage instances

Create an instance

Use the *OS::Nova::Server* resource to create a Compute instance. The `flavor` property is the only mandatory one, but you need to define a boot source using one of the `image` or `block_device_mapping` properties.

You also need to define the `networks` property to indicate to which networks your instance must connect if multiple networks are available in your tenant.

The following example creates a simple instance, booted from an image, and connecting to the `private` network:

```
resources:
  instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      networks:
        - network: private
```

Connect an instance to a network

Use the `networks` property of an *OS::Nova::Server* resource to define which networks an instance should connect to. Define each network as a YAML map, containing one of the following keys:

port
: The ID of an existing Networking port. You usually create this port in the same template using an *OS::Neutron::Port* resource. You will be able to associate a floating IP to this port, and the port to your Compute instance.

network
: The name or ID of an existing network. You dont need to create an *OS::Neutron::Port* resource if you use this property. But you will not be able to use neutron floating IP association for this instance because there will be no specified port for server.

The following example demonstrates the use of the `port` and `network` properties:

```
resources:
  instance_port:
    type: OS::Neutron::Port
    properties:
      network: private
      fixed_ips:
        - subnet_id: "private-subnet"

  instance1:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      networks:
        - port: { get_resource: instance_port }

  instance2:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
```

(continues on next page)

(continued from previous page)

```
    networks:
      - network: private
```

Create and associate security groups to an instance

Use the *OS::Neutron::SecurityGroup* resource to create security groups.

Define the `security_groups` property of the *OS::Neutron::Port* resource to associate security groups to a port, then associate the port to an instance.

The following example creates a security group allowing inbound connections on ports 80 and 443 (web server) and associates this security group to an instance port:

```
resources:
  web_secgroup:
    type: OS::Neutron::SecurityGroup
    properties:
      rules:
        - protocol: tcp
          remote_ip_prefix: 0.0.0.0/0
          port_range_min: 80
          port_range_max: 80
        - protocol: tcp
          remote_ip_prefix: 0.0.0.0/0
          port_range_min: 443
          port_range_max: 443

  instance_port:
    type: OS::Neutron::Port
    properties:
      network: private
      security_groups:
        - default
        - { get_resource: web_secgroup }
      fixed_ips:
        - subnet_id: private-subnet

  instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      networks:
        - port: { get_resource: instance_port }
```

Create and associate a floating IP to an instance

Use the *OS::Neutron::FloatingIP* resource to create a floating IP, and the *OS::Neutron::FloatingIPAssociation* resource to associate the floating IP to a port:

```yaml
parameters:
  net:
    description: name of network used to launch instance.
    type: string
    default: private

resources:
  inst1:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      networks:
        - network: {get_param: net}

  floating_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: public

  association:
    type: OS::Neutron::FloatingIPAssociation
    properties:
      floatingip_id: { get_resource: floating_ip }
      port_id: {get_attr: [inst1, addresses, {get_param: net}, 0, port]}
```

You can also create an OS::Neutron::Port and associate that with the server and the floating IP. However the approach mentioned above will work better with stack updates.

```yaml
resources:
  instance_port:
    type: OS::Neutron::Port
    properties:
      network: private
      fixed_ips:
        - subnet_id: "private-subnet"

  floating_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: public

  association:
    type: OS::Neutron::FloatingIPAssociation
    properties:
      floatingip_id: { get_resource: floating_ip }
      port_id: { get_resource: instance_port }
```

Enable remote access to an instance

The `key_name` attribute of the *OS::Nova::Server* resource defines the key pair to use to enable SSH remote access:

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      key_name: my_key
```

Note

For more information about key pairs, see Configure access and security for instances.

Create a key pair

You can create new key pairs with the *OS::Nova::KeyPair* resource. Key pairs can be imported or created during the stack creation.

If the `public_key` property is not specified, the Orchestration module creates a new key pair. If the `save_private_key` property is set to `true`, the `private_key` attribute of the resource holds the private key.

The following example creates a new key pair and uses it as authentication key for an instance:

```
resources:
  my_key:
    type: OS::Nova::KeyPair
    properties:
      save_private_key: true
      name: my_key

  my_instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      key_name: { get_resource: my_key }

outputs:
  private_key:
    description: Private key
    value: { get_attr: [ my_key, private_key ] }
```

Manage networks

Create a network and a subnet

> **Note**
>
> The Networking service (neutron) must be enabled on your OpenStack deployment to create and manage networks and subnets. Networks and subnets cannot be created if your deployment uses legacy networking (nova-network).

Use the *OS::Neutron::Net* resource to create a network, and the *OS::Neutron::Subnet* resource to provide a subnet for this network:

```
resources:
  new_net:
    type: OS::Neutron::Net

  new_subnet:
    type: OS::Neutron::Subnet
    properties:
      network_id: { get_resource: new_net }
      cidr: "10.8.1.0/24"
      dns_nameservers: [ "8.8.8.8", "8.8.4.4" ]
      ip_version: 4
```

Create and manage a router

Use the *OS::Neutron::Router* resource to create a router. You can define its gateway with the `external_gateway_info` property:

```
resources:
  router1:
    type: OS::Neutron::Router
    properties:
      external_gateway_info: { network: public }
```

You can connect subnets to routers with the *OS::Neutron::RouterInterface* resource:

```
resources:
  subnet1_interface:
    type: OS::Neutron::RouterInterface
    properties:
      router_id: { get_resource: router1 }
      subnet: private-subnet
```

Complete network example

The following example creates a network stack:

- A network and an associated subnet.

- A router with an external gateway.
- An interface to the new subnet for the new router.

In this example, the `public` network is an existing shared network:

```
resources:
  internal_net:
    type: OS::Neutron::Net

  internal_subnet:
    type: OS::Neutron::Subnet
    properties:
      network_id: { get_resource: internal_net }
      cidr: "10.8.1.0/24"
      dns_nameservers: [ "8.8.8.8", "8.8.4.4" ]
      ip_version: 4

  internal_router:
    type: OS::Neutron::Router
    properties:
      external_gateway_info: { network: public }

  internal_interface:
    type: OS::Neutron::RouterInterface
    properties:
      router_id: { get_resource: internal_router }
      subnet: { get_resource: internal_subnet }
```

Manage volumes

Create a volume

Use the *OS::Cinder::Volume* resource to create a new Block Storage volume.

For example:

```
resources:
  my_new_volume:
    type: OS::Cinder::Volume
    properties:
      size: 10
```

The volumes that you create are empty by default. Use the `image` property to create a bootable volume from an existing image:

```
resources:
  my_new_bootable_volume:
    type: OS::Cinder::Volume
    properties:
      size: 10
      image: ubuntu-trusty-x86_64
```

You can also create new volumes from another volume, a volume snapshot, or a volume backup. Use the `source_volid`, `snapshot_id` or `backup_id` properties to create a new volume from an existing source.

For example, to create a new volume from a backup:

```
resources:
  another_volume:
    type: OS::Cinder::Volume
    properties:
      backup_id: 2fff50ab-1a9c-4d45-ae60-1d054d6bc868
```

In this example the `size` property is not defined because the Block Storage service uses the size of the backup to define the size of the new volume.

Attach a volume to an instance

Use the *OS::Cinder::VolumeAttachment* resource to attach a volume to an instance.

The following example creates a volume and an instance, and attaches the volume to the instance:

```
resources:
  new_volume:
    type: OS::Cinder::Volume
    properties:
      size: 1

  new_instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64

  volume_attachment:
    type: OS::Cinder::VolumeAttachment
    properties:
      volume_id: { get_resource: new_volume }
      instance_uuid: { get_resource: new_instance }
```

Boot an instance from a volume

Use the `block_device_mapping` property of the *OS::Nova::Server* resource to define a volume used to boot the instance. This property is a list of volumes to attach to the instance before its boot.

The following example creates a bootable volume from an image, and uses it to boot an instance:

```
resources:
  bootable_volume:
    type: OS::Cinder::Volume
    properties:
      size: 10
      image: ubuntu-trusty-x86_64
```

(continues on next page)

(continued from previous page)

```
instance:
  type: OS::Nova::Server
  properties:
    flavor: m1.small
    networks:
      - network: private
    block_device_mapping:
      - device_name: vda
        volume_id: { get_resource: bootable_volume }
        delete_on_termination: false
```

Software configuration

There are a variety of options to configure the software which runs on the servers in your stack. These can be broadly divided into the following:

- Custom image building
- User-data boot scripts and cloud-init
- Software deployment resources

This section will describe each of these options and provide examples for using them together in your stacks.

Image building

The first opportunity to influence what software is configured on your servers is by booting them with a custom-built image. There are a number of reasons you might want to do this, including:

- **Boot speed** - since the required software is already on the image there is no need to download and install anything at boot time.
- **Boot reliability** - software downloads can fail for a number of reasons including transient network failures and inconsistent software repositories.
- **Test verification** - custom built images can be verified in test environments before being promoted to production.
- **Configuration dependencies** - post-boot configuration may depend on agents already being installed and enabled

A number of tools are available for building custom images, including:

- diskimage-builder image building tools for OpenStack
- imagefactory builds images for a variety of operating system/cloud combinations

Examples in this guide that require custom images will use diskimage-builder.

User-data boot scripts and cloud-init

When booting a server it is possible to specify the contents of the user-data to be passed to that server. This user-data is made available either from configured config-drive or from the Metadata service

How this user-data is consumed depends on the image being booted, but the most commonly used tool for default cloud images is cloud-init.

Whether the image is using cloud-init or not, it should be possible to specify a shell script in the `user_data` property and have it be executed by the server during boot:

```
resources:

  the_server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data: |
        #!/bin/bash
        echo "Running boot script"
        # ...
```

> **Note**
>
> Debugging these scripts it is often useful to view the boot log using `nova console-log <server-id>` to view the progress of boot script execution.

Often there is a need to set variable values based on parameters or resources in the stack. This can be done with the `str_replace` intrinsic function:

```
parameters:
  foo:
    default: bar

resources:

  the_server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data:
        str_replace:
          template: |
            #!/bin/bash
            echo "Running boot script with $FOO"
            # ...
          params:
           $FOO: {get_param: foo}
```

Warning

If a stack-update is performed and there are any changes at all to the content of user_data then the server will be replaced (deleted and recreated) so that the modified boot configuration can be run on a new server.

When these scripts grow it can become difficult to maintain them inside the template, so the `get_file` intrinsic function can be used to maintain the script in a separate file:

```
parameters:
  foo:
    default: bar

resources:

  the_server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data:
        str_replace:
          template: {get_file: the_server_boot.sh}
          params:
            $FOO: {get_param: foo}
```

Note

`str_replace` can replace any strings, not just strings starting with `$`. However doing this for the above example is useful because the script file can be executed for testing by passing in environment variables.

Choosing the user_data_format

The *OS::Nova::Server* `user_data_format` property determines how the `user_data` should be formatted for the server. For the default value `HEAT_CFNTOOLS`, the `user_data` is bundled as part of the heat-cfntools cloud-init boot configuration data. While `HEAT_CFNTOOLS` is the default for `user_data_format`, it is considered legacy and `RAW` or `SOFTWARE_CONFIG` will generally be more appropriate.

For `RAW` the user_data is passed to Nova unmodified. For a cloud-init enabled image, the following are both valid `RAW` user-data:

```
resources:

  server_with_boot_script:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: RAW
      user_data: |
```

(continues on next page)

(continued from previous page)

```
        #!/bin/bash
        echo "Running boot script"
        # ...

  server_with_cloud_config:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: RAW
      user_data: |
        #cloud-config
        final_message: "The system is finally up, after $UPTIME seconds"
```

For `SOFTWARE_CONFIG` `user_data` is bundled as part of the software config data, and metadata is derived from any associated *Software deployment resources*.

Signals and wait conditions

Often it is necessary to pause further creation of stack resources until the boot configuration script has notified that it has reached a certain state. This is usually either to notify that a service is now active, or to pass out some generated data which is needed by another resource. The resources *OS::Heat::WaitCondition* and *OS::Heat::SwiftSignal* both perform this function using different techniques and tradeoffs.

OS::Heat::WaitCondition is implemented as a call to the Orchestration API resource signal. The token is created using credentials for a user account which is scoped only to the wait condition handle resource. This user is created when the handle is created, and is associated to a project which belongs to the stack, in an identity domain which is dedicated to the orchestration service.

Sending the signal is a simple HTTP request, as with this example using curl:

```
curl -i -X POST -H 'X-Auth-Token: <token>' \
     -H 'Content-Type: application/json' -H 'Accept: application/json' \
     '<wait condition URL>' --data-binary '<json containing signal data>'
```

The JSON containing the signal data is expected to be of the following format:

```
{
  "status": "SUCCESS",
  "reason": "The reason which will appear in the 'heat event-list' output",
  "data": "Data to be used elsewhere in the template via get_attr",
  "id": "Optional unique ID of signal"
}
```

All of these values are optional, and if not specified will be set to the following defaults:

```
{
  "status": "SUCCESS",
  "reason": "Signal <id> received",
  "data": null,
```

(continues on next page)

(continued from previous page)

```
  "id": "<sequential number starting from 1 for each signal received>"
}
```

If `status` is set to `FAILURE` then the resource (and the stack) will go into a `FAILED` state using the `reason` as failure reason.

The following template example uses the convenience attribute `curl_cli` which builds a curl command with a valid token:

```
resources:
  wait_condition:
    type: OS::Heat::WaitCondition
    properties:
      handle: {get_resource: wait_handle}
      # Note, count of 5 vs 6 is due to duplicate signal ID 5 sent below
      count: 5
      timeout: 300

  wait_handle:
    type: OS::Heat::WaitConditionHandle

  the_server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: RAW
      user_data:
        str_replace:
          template: |
            #!/bin/sh
            # Below are some examples of the various ways signals
            # can be sent to the Handle resource

            # Simple success signal
            wc_notify --data-binary '{"status": "SUCCESS"}'

            # Or you optionally can specify any of the additional fields
            wc_notify --data-binary '{"status": "SUCCESS", "reason": "signal2
↪"}'
            wc_notify --data-binary '{"status": "SUCCESS", "reason": "signal3
↪", "data": "data3"}'
            wc_notify --data-binary '{"status": "SUCCESS", "reason": "signal4
↪", "id": "id4", "data": "data4"}'

            # If you require control of the ID, you can pass it.
            # The ID should be unique, unless you intend for duplicate
            # signals to overwrite each other.  The following two calls
            # do the exact same thing, and will be treated as one signal
            # (You can prove this by changing count above to 7)
            wc_notify --data-binary '{"status": "SUCCESS", "id": "id5"}'
```

(continues on next page)

(continued from previous page)

```
            wc_notify --data-binary '{"status": "SUCCESS", "id": "id5"}'

            # Example of sending a failure signal, optionally
            # reason, id, and data can be specified as above
            # wc_notify --data-binary '{"status": "FAILURE"}'
          params:
            wc_notify: { get_attr: [wait_handle, curl_cli] }

outputs:
  wc_data:
    value: { get_attr: [wait_condition, data] }
    # this would return the following json
    # {"1": null, "2": null, "3": "data3", "id4": "data4", "id5": null}

  wc_data_4:
    value: { 'Fn::Select': ['id4', { get_attr: [wait_condition, data] }] }
    # this would return "data4"
```

OS::Heat::SwiftSignal is implemented by creating an Object Storage API temporary URL which is populated with signal data with an HTTP PUT. The orchestration service will poll this object until the signal data is available. Object versioning is used to store multiple signals.

Sending the signal is a simple HTTP request, as with this example using curl:

```
curl -i -X PUT '<object URL>' --data-binary '<json containing signal data>'
```

The above template example only needs to have the `type` changed to the swift signal resources:

```
resources:
  signal:
    type: OS::Heat::SwiftSignal
    properties:
      handle: {get_resource: wait_handle}
      timeout: 300

  signal_handle:
    type: OS::Heat::SwiftSignalHandle
  # ...
```

The decision to use *OS::Heat::WaitCondition* or *OS::Heat::SwiftSignal* will depend on a few factors:

- *OS::Heat::SwiftSignal* depends on the availability of an Object Storage API
- *OS::Heat::WaitCondition* depends on whether the orchestration service has been configured with a dedicated stack domain (which may depend on the availability of an Identity V3 API).
- The preference to protect signal URLs with token authentication or a secret webhook URL.

Software config resources

Boot configuration scripts can also be managed as their own resources. This allows configuration to be defined once and run on multiple server resources. These software-config resources are stored and retrieved via dedicated calls to the Orchestration API. It is not possible to modify the contents of an existing software-config resource, so a stack-update which changes any existing software-config resource will result in API calls to create a new config and delete the old one.

The resource *OS::Heat::SoftwareConfig* is used for storing configs represented by text scripts, for example:

```
resources:
  boot_script:
    type: OS::Heat::SoftwareConfig
    properties:
      group: ungrouped
      config: |
        #!/bin/bash
        echo "Running boot script"
        # ...

  server_with_boot_script:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: SOFTWARE_CONFIG
      user_data: {get_resource: boot_script}
```

The resource *OS::Heat::CloudConfig* allows cloud-init cloud-config to be represented as template YAML rather than a block string. This allows intrinsic functions to be included when building the cloud-config. This also ensures that the cloud-config is valid YAML, although no further checks for valid cloud-config are done.

```
parameters:
  file_content:
    type: string
    description: The contents of the file /tmp/file

resources:
  boot_config:
    type: OS::Heat::CloudConfig
    properties:
      cloud_config:
        write_files:
        - path: /tmp/file
          content: {get_param: file_content}

  server_with_cloud_config:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: SOFTWARE_CONFIG
```

(continues on next page)

(continued from previous page)

```
      user_data: {get_resource: boot_config}
```

The resource *OS::Heat::MultipartMime* allows multiple *OS::Heat::SoftwareConfig* and *OS::Heat::CloudConfig* resources to be combined into a single cloud-init multi-part message:

```
parameters:
  file_content:
    type: string
    description: The contents of the file /tmp/file

  other_config:
    type: string
    description: The ID of a software-config resource created elsewhere

resources:
  boot_config:
    type: OS::Heat::CloudConfig
    properties:
      cloud_config:
        write_files:
        - path: /tmp/file
          content: {get_param: file_content}

  boot_script:
    type: OS::Heat::SoftwareConfig
    properties:
      group: ungrouped
      config: |
        #!/bin/bash
        echo "Running boot script"
        # ...

  server_init:
    type: OS::Heat::MultipartMime
    properties:
      parts:
      - config: {get_resource: boot_config}
      - config: {get_resource: boot_script}
      - config: {get_param: other_config}

  server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: SOFTWARE_CONFIG
      user_data: {get_resource: server_init}
```

Software deployment resources

There are many situations where it is not desirable to replace the server whenever there is a configuration change. The *OS::Heat::SoftwareDeployment* resource allows any number of software configurations to be added or removed from a server throughout its life-cycle.

Building custom image for software deployments

OS::Heat::SoftwareConfig resources are used to store software configuration, and a *OS::Heat::SoftwareDeployment* resource is used to associate a config resource with one server. The `group` attribute on *OS::Heat::SoftwareConfig* specifies what tool will consume the config content.

OS::Heat::SoftwareConfig has the ability to define a schema of `inputs` and which the configuration script supports. Inputs are mapped to whatever concept the configuration tool has for assigning variables/parameters.

Likewise, `outputs` are mapped to the tools capability to export structured data after configuration execution. For tools which do not support this, outputs can always be written to a known file path for the hook to read.

The *OS::Heat::SoftwareDeployment* resource allows values to be assigned to the config inputs, and the resource remains in an `IN_PROGRESS` state until the server signals to heat what (if any) output values were generated by the config script.

Custom image script

Each of the following examples requires that the servers be booted with a custom image. The following script uses diskimage-builder to create an image required in later examples:

```
# Clone the required repositories. Some of these are also available
# via pypi or as distro packages.
git clone https://opendev.org/openstack/tripleo-image-elements
git clone https://opendev.org/openstack/heat-agents

# Install diskimage-builder from source
sudo pip install git+https://opendev.org/openstack/diskimage-builder

# Required by diskimage-builder to discover element collections
export ELEMENTS_PATH=tripleo-image-elements/elements:heat-agents/

# The base operating system element(s) provided by the diskimage-builder
# elements collection. Other values which may work include:
# centos7, debian, opensuse, rhel, rhel7, or ubuntu
export BASE_ELEMENTS="fedora selinux-permissive"
# Install and configure the os-collect-config agent to poll the metadata
# server (heat service or zaqar message queue and so on) for configuration
# changes to execute
export AGENT_ELEMENTS="os-collect-config os-refresh-config os-apply-config"


# heat-config installs an os-refresh-config script which will invoke the
# appropriate hook to perform configuration. The element heat-config-script
# installs a hook to perform configuration with shell scripts
```

(continues on next page)

(continued from previous page)

```
export DEPLOYMENT_BASE_ELEMENTS="heat-config heat-config-script"

# Install a hook for any other chosen configuration tool(s).
# Elements which install hooks include:
# heat-config-cfn-init, heat-config-puppet, or heat-config-salt
export DEPLOYMENT_TOOL=""

# The name of the qcow2 image to create, and the name of the image
# uploaded to the OpenStack image registry.
export IMAGE_NAME=fedora-software-config

# Create the image
disk-image-create vm $BASE_ELEMENTS $AGENT_ELEMENTS \
     $DEPLOYMENT_BASE_ELEMENTS $DEPLOYMENT_TOOL -o $IMAGE_NAME.qcow2

# Upload the image, assuming valid credentials are already sourced
openstack image create --disk-format qcow2 --container-format bare \
    $IMAGE_NAME < $IMAGE_NAME.qcow2
```

Note

Above script uses diskimage-builder, make sure the environment already fulfill all requirements in requirements.txt of diskimage-builder.

Configuring with scripts

The *Custom image script* already includes the `heat-config-script` element so the built image will already have the ability to configure using shell scripts.

Config inputs are mapped to shell environment variables. The script can communicate outputs to heat by writing to the `$heat_outputs_path.`*`output name`* file. See the following example for a script which expects inputs `foo`, `bar` and generates an output `result`.

```
resources:
  config:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      inputs:
      - name: foo
      - name: bar
      outputs:
      - name: result
      config: |
        #!/bin/sh -x
        echo "Writing to /tmp/$bar"
        echo $foo > /tmp/$bar
        echo -n "The file /tmp/$bar contains `cat /tmp/$bar` for server
↪$deploy_server_id during $deploy_action" > $heat_outputs_path.result
```

(continues on next page)

(continued from previous page)

```
        echo "Written to /tmp/$bar"
        echo "Output to stderr" 1>&2

  deployment:
    type: OS::Heat::SoftwareDeployment
    properties:
      config:
        get_resource: config
      server:
        get_resource: server
      input_values:
        foo: fooooo
        bar: baaaaa

  server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: SOFTWARE_CONFIG

outputs:
  result:
    value:
      get_attr: [deployment, result]
  stdout:
    value:
      get_attr: [deployment, deploy_stdout]
  stderr:
    value:
      get_attr: [deployment, deploy_stderr]
  status_code:
    value:
      get_attr: [deployment, deploy_status_code]
```

Note

A config resource can be associated with multiple deployment resources, and each deployment can specify the same or different values for the `server` and `input_values` properties.

As can be seen in the `outputs` section of the above template, the `result` config output value is available as an attribute on the `deployment` resource. Likewise the captured stdout, stderr and status_code are also available as attributes.

Configuring with os-apply-config

The agent toolchain of `os-collect-config`, `os-refresh-config` and `os-apply-config` can actually be used on their own to inject heat stack configuration data into a server running a custom image.

The custom image needs to have the following to use this approach:

- All software dependencies installed
- os-refresh-config scripts to be executed on configuration changes
- os-apply-config templates to transform the heat-provided config data into service configuration files

The projects tripleo-image-elements and tripleo-heat-templates demonstrate this approach.

Configuring with cfn-init

Likely the only reason to use the `cfn-init` hook is to migrate templates which contain AWS::CloudFormation::Init metadata without needing a complete rewrite of the config metadata. It is included here as it introduces a number of new concepts.

To use the `cfn-init` tool the `heat-config-cfn-init` element is required to be on the built image, so *Custom image script* needs to be modified with the following:

```
export DEPLOYMENT_TOOL="heat-config-cfn-init"
```

Configuration data which used to be included in the `AWS::CloudFormation::Init` section of resource metadata is instead moved to the `config` property of the config resource, as in the following example:

```
resources:

  config:
    type: OS::Heat::StructuredConfig
    properties:
      group: cfn-init
      inputs:
      - name: bar
      config:
        config:
          files:
            /tmp/foo:
              content:
                get_input: bar
              mode: '000644'

  deployment:
    type: OS::Heat::StructuredDeployment
    properties:
      name: 10_deployment
      signal_transport: NO_SIGNAL
      config:
        get_resource: config
      server:
        get_resource: server
      input_values:
        bar: baaaaa

  other_deployment:
    type: OS::Heat::StructuredDeployment
```

(continues on next page)

(continued from previous page)

```
    properties:
      name: 20_other_deployment
      signal_transport: NO_SIGNAL
      config:
        get_resource: config
      server:
        get_resource: server
      input_values:
        bar: barmy

  server:
    type: OS::Nova::Server
    properties:
      image: {get_param: image}
      flavor: {get_param: flavor}
      key_name: {get_param: key_name}
      user_data_format: SOFTWARE_CONFIG
```

There are a number of things to note about this template example:

- *OS::Heat::StructuredConfig* is like *OS::Heat::SoftwareConfig* except that the `config` property contains structured YAML instead of text script. This is useful for a number of other configuration tools including ansible, salt and os-apply-config.
- `cfn-init` has no concept of inputs, so `{get_input: bar}` acts as a placeholder which gets replaced with the *OS::Heat::StructuredDeployment* `input_values` value when the deployment resource is created.
- `cfn-init` has no concept of outputs, so specifying `signal_transport: NO_SIGNAL` will mean that the deployment resource will immediately go into the `CREATED` state instead of waiting for a completed signal from the server.
- The template has 2 deployment resources deploying the same config with different `input_values`. The order these are deployed in on the server is determined by sorting the values of the `name` property for each resource (10_deployment, 20_other_deployment)

Configuring with puppet

The puppet hook makes it possible to write configuration as puppet manifests which are deployed and run in a masterless environment.

To specify configuration as puppet manifests the `heat-config-puppet` element is required to be on the built image, so *Custom image script* needs to be modified with the following:

```
export DEPLOYMENT_TOOL="heat-config-puppet"
```

```
resources:

  config:
    type: OS::Heat::SoftwareConfig
    properties:
      group: puppet
```

(continues on next page)

(continued from previous page)

```
      inputs:
      - name: foo
      - name: bar
      outputs:
      - name: result
      config:
        get_file: example-puppet-manifest.pp

  deployment:
    type: OS::Heat::SoftwareDeployment
    properties:
      config:
        get_resource: config
      server:
        get_resource: server
      input_values:
        foo: fooooo
        bar: baaaaa

  server:
    type: OS::Nova::Server
    properties:
      image: {get_param: image}
      flavor: {get_param: flavor}
      key_name: {get_param: key_name}
      user_data_format: SOFTWARE_CONFIG

outputs:
  result:
    value:
      get_attr: [deployment, result]
  stdout:
    value:
get_attr: [deployment, deploy_stdout]
```

This demonstrates the use of the `get_file` function, which will attach the contents of the file `example-puppet-manifest.pp`, containing:

```
file { 'barfile':
    ensure  => file,
    mode    => '0644',
    path    => '/tmp/$::bar',
    content => '$::foo',
}

file { 'output_result':
    ensure  => file,
    path    => '$::heat_outputs_path.result',
    mode    => '0644',
    content => 'The file /tmp/$::bar contains $::foo',
```

(continues on next page)

(continued from previous page)

```
}
```

Environments

The environment affects the runtime behavior of a template. It provides a way to override the resource implementations and a mechanism to place parameters that the service needs.

To fully understand the runtime behavior you have to consider what plug-ins are installed on the cloud youre using.

Environment file format

The environment is a yaml text file that contains two main sections:

parameters
: A list of key/value pairs.

resource_registry
: Definition of custom resources.

It also can contain some other sections:

parameter_defaults
: Default parameters passed to all template resources.

encrypted_parameters
: List of encrypted parameters.

event_sinks
: List of endpoints that would receive stack events.

parameter_merge_strategies
: Merge strategies for merging parameters and parameter defaults from the environment file.

Use the *-e* option of the **openstack stack create** command to create a stack using the environment defined in such a file.

You can also provide environment parameters as a list of key/value pairs using the *parameter* option of the **openstack stack create** command.

In the following example the environment is read from the `my_env.yaml` file and an extra parameter is provided using the *parameter* option:

```
$ openstack stack create my_stack -e my_env.yaml --parameter "param1=val1;
↪param2=val2" -t my_tmpl.yaml
```

Environment Merging

Parameters and their defaults (`parameter_defaults`) are merged based on merge strategies in an environment file.

There are three merge strategy types:

overwrite
: Overwrites a parameter, existing parameter values are replaced.

merge
: Merges the existing parameter value and the new value. String values are concatenated, comma delimited lists are extended and json values are updated.

deep_merge
: Json values are deep merged. Not useful for other types like comma delimited lists and strings. If specified for them, it falls back to `merge`.

You can provide a default merge strategy and/or parameter specific merge strategies per environment file. Parameter specific merge strategy is only used for that parameter. An example of `parameter_merge_strategies` section in an environment file:

```
parameter_merge_strategies:
  default: merge
  param1: overwrite
  param2: deep_merge
```

If no merge strategy is provided in an environment file, `overwrite` becomes the default merge strategy for all `parameters` and `parameter_defaults` in that environment file.

Global and effective environments

The environment used for a stack is the combination of the environment you use with the template for the stack, and a global environment that is determined by your cloud operator. An entry in the user environment takes precedence over the global environment. OpenStack includes a default global environment, but your cloud operator can add additional environment entries.

The cloud operator can add to the global environment by putting environment files in a configurable directory wherever the Orchestration engine runs. The configuration variable is named `environment_dir` and is found in the `[DEFAULT]` section of `/etc/heat/heat.conf`. The default for that directory is `/etc/heat/environment.d`. Its contents are combined in whatever order the shell delivers them when the service starts up, which is the time when these files are read. If the `my_env.yaml` file from the example above had been put in the `environment_dir` then the users command line could be this:

```
openstack stack create my_stack --parameter "some_parm=bla" -t my_tmpl.yaml
```

Global templates

A global template directory allows files to be pre-loaded in the global environment. A global template is determined by your cloud operator. An entry in the user template takes precedence over the global environment. OpenStack includes a default global template, but your cloud operator can add additional template entries.

The cloud operator can add new global templates by putting template files in a configurable directory wherever the Orchestration engine runs. The configuration variable is named `template_dir` and is found in the `[DEFAULT]` section of `/etc/heat/heat.conf`. The default for that directory is `/etc/heat/templates`. Its contents are combined in whatever order the shell delivers them when the service starts up, which is the time when these files are read. If the `my_tmpl.yaml` file from the example below has been put in the `template_dir`, other templates which we used to create stacks could contain following way to include *my_tmpl.yaml* in it:

```
resourceA:
  type: {get_file: "my_tmpl.yaml"}
```

Usage examples

Define values for template arguments

You can define values for the template arguments in the `parameters` section of an environment file:

```
parameters:
  KeyName: heat_key
  InstanceType: m1.micro
  ImageId: F18-x86_64-cfntools
```

Define defaults to parameters

You can define default values for all template arguments in the `parameter_defaults` section of an environment file. These defaults are passed into all template resources:

```
parameter_defaults:
  KeyName: heat_key
```

Mapping resources

You can map one resource to another in the `resource_registry` section of an environment file. The resource you provide in this manner must have an identifier, and must reference either another resources ID or the URL of an existing template file.

The following example maps a new `OS::Networking::FloatingIP` resource to an existing `OS::Neutron::FloatingIP` resource:

```
resource_registry:
  "OS::Networking::FloatingIP": "OS::Neutron::FloatingIP"
```

You can use wildcards to map multiple resources, for example to map all `OS::Neutron` resources to `OS::Network`:

```
resource_registry:
  "OS::Network*": "OS::Neutron*"
```

Override a resource with a custom resource

To create or override a resource with a custom resource, create a template file to define this resource, and provide the URL to the template file in the environment file:

```
resource_registry:
  "AWS::EC2::Instance": file:///path/to/my_instance.yaml
```

The supported URL schemes are `file`, `http` and `https`.

> **Note**
>
> The template file extension must be `.yaml` or `.template`, or it will not be treated as a custom template resource.

You can limit the usage of a custom resource to a specific resource of the template:

```
resource_registry:
  resources:
    my_db_server:
      "OS::DBInstance": file:///home/mine/all_my_cool_templates/db.yaml
```

Pause stack creation, update or deletion on a given resource

If you want to debug your stack as its being created, updated or deleted, or if you want to run it in phases, you can set `pre-create`, `pre-update`, `pre-delete`, `post-create`, `post-update` and `post-delete` hooks in the `resources` section of `resource_registry`.

To set a hook, add either `hooks: $hook_name` (for example `hooks: pre-update`) to the resources dictionary. You can also use a list (`hooks: [pre-create, pre-update]`) to stop on several actions.

You can combine hooks with other `resources` properties such as provider templates or type mapping:

```
resource_registry:
  resources:
    my_server:
      "OS::DBInstance": file:///home/mine/all_my_cool_templates/db.yaml
      hooks: pre-create
    nested_stack:
      nested_resource:
        hooks: pre-update
      another_resource:
        hooks: [pre-create, pre-update]
```

When heat encounters a resource that has a hook, it pauses the resource action until the hook clears. Any resources that depend on the paused action wait as well. Non-dependent resources are created in parallel unless they have their own hooks.

It is possible to perform a wild card match using an asterisk (*) in the resource name. For example, the following entry pauses while creating `app_server` and `database_server`, but not `server` or `app_network`:

```
resource_registry:
  resources:
    "*_server":
      hooks: pre-create
```

Clear hooks by signaling the resource with `{unset_hook: $hook_name}` (for example `{unset_hook: pre-update}`).

Retrieving events

By default events are stored in the database and can be retrieved via the API. Using the environment, you can register an endpoint which will receive events produced by your stack, so that you dont have to poll Heat.

You can specify endpoints using the `event_sinks` property:

```
event_sinks:
  - type: zaqar-queue
    target: myqueue
    ttl: 1200
```

Restrict update or replace of a given resource

If you want to restrict update or replace of a resource when your stack is being updated, you can set `restricted_actions` in the `resources` section of `resource_registry`.

To restrict update or replace, add `restricted_actions: update` or `restricted_actions: replace` to the resource dictionary. You can also use `[update, replace]` to restrict both actions.

You can combine restricted actions with other `resources` properties such as provider templates or type mapping or hooks:

```
resource_registry:
  resources:
    my_server:
      "OS::DBInstance": file:///home/mine/all_my_cool_templates/db.yaml
      restricted_actions: replace
      hooks: pre-create
    nested_stack:
      nested_resource:
        restricted_actions: update
      another_resource:
        restricted_actions: [update, replace]
```

It is possible to perform a wild card match using an asterisk (*) in the resource name. For example, the following entry restricts replace for `app_server` and `database_server`, but not `server` or `app_network`:

```
resource_registry:
  resources:
    "*_server":
      restricted_actions: replace
```

Template composition

When writing complex templates you are encouraged to break up your template into separate smaller templates. These can then be brought together using template resources. This is a mechanism to define a resource using a template, thus composing one logical stack with multiple templates.

Template resources provide a feature similar to the *AWS::CloudFormation::Stack* resource, but also provide a way to:

- Define new resource types and build your own resource library.
- Override the default behavior of existing resource types.

To achieve this:

- The Orchestration client gets the associated template files and passes them along in the `files` section of the `POST stacks/` API request.

- The environment in the Orchestration engine manages the mapping of resource type to template creation.
- The Orchestration engine translates template parameters into resource properties.

The following examples illustrate how you can use a custom template to define new types of resources. These examples use a custom template stored in a `my_nova.yaml` file

```
heat_template_version: 2015-04-30

parameters:
  key_name:
    type: string
    description: Name of a KeyPair

resources:
  server:
    type: OS::Nova::Server
    properties:
      key_name: {get_param: key_name}
      flavor: m1.small
      image: ubuntu-trusty-x86_64
```

Use the template filename as type

The following template defines the `my_nova.yaml` file as value for the `type` property of a resource

```
heat_template_version: 2015-04-30

resources:
  my_server:
    type: my_nova.yaml
    properties:
      key_name: my_key
```

The `key_name` argument of the `my_nova.yaml` template gets its value from the `key_name` property of the new template.

Note

The above reference to `my_nova.yaml` assumes it is in the same directory. You can use any of the following forms:

- Relative path (`my_nova.yaml`)
- Absolute path (`file:///home/user/templates/my_nova.yaml`)
- Http URL (`http://example.com/templates/my_nova.yaml`)
- Https URL (`https://example.com/templates/my_nova.yaml`)

To create the stack run:

```
$ openstack stack create -t main.yaml stack1
```

Define a new resource type

You can associate a name to the `my_nova.yaml` template in an environment file. If the name is already known by the Orchestration module then your new resource will override the default one.

In the following example a new `OS::Nova::Server` resource overrides the default resource of the same name.

An `env.yaml` environment file holds the definition of the new resource

```yaml
resource_registry:
  "OS::Nova::Server": my_nova.yaml
```

> **Note**
>
> See *Environments* for more detail about environment files.

You can now use the new `OS::Nova::Server` in your new template

```yaml
heat_template_version: 2015-04-30

resources:
  my_server:
    type: OS::Nova::Server
    properties:
      key_name: my_key
```

To create the stack run:

```
$ openstack stack create -t main.yaml -e env.yaml example-two
```

Get access to nested attributes

There are implicit attributes of a template resource. Accessing nested attributes requires `heat_template_version` 2014-10-16 or higher. These are accessible as follows

```yaml
heat_template_version: 2015-04-30

resources:
  my_server:
    type: my_nova.yaml

outputs:
  test_out:
    value: {get_attr: [my_server, resource.server, first_address]}
```

Making your template resource more transparent

Note

Available since 2015.1 (Kilo).

If you wish to be able to return the ID of one of the inner resources instead of the nested stacks identifier, you can add the special reserved output `OS::stack_id` to your template resource

```
heat_template_version: 2015-04-30

resources:
  server:
    type: OS::Nova::Server

outputs:
  OS::stack_id:
    value: {get_resource: server}
```

Now when you use `get_resource` from the outer template heat will use the nova server id and not the template resource identifier.

OpenStack Resource Types

OS::Aodh::CompositeAlarm

Available since 8.0.0 (Ocata)

A resource that implements Aodh composite alarm.

Allows to specify multiple rules when creating a composite alarm, and the rules combined with logical operators: and, or.

Required Properties

composite_rule*ű*

Composite threshold rules in JSON format.

Map value expected.

Can be updated without replacement.

Map properties:

operator*ű*

Required.

The operator indicates how to combine the rules.

String value expected.

Can be updated without replacement.

Allowed values: or, and

rules*ű*

Rules list. Basic threshold/gnocchi rules and nested dict which combine threshold/gnocchi rules by and or or are allowed. For example, the form is like: [RULE1, RULE2, {and: [RULE3, RULE4]}], the basic threshold/gnocchi rules must include a type field.

List value expected.

Can be updated without replacement.

The length must be at least 2.

Optional Properties

alarm_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to alarm.

List value expected.

Can be updated without replacement.

alarm_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to alarm.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

description*ű*

Description for the alarm.

String value expected.

Can be updated without replacement.

enabled*ű*

True if alarm evaluation/actioning is enabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

insufficient_data_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

insufficient_data_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

ok_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to ok.

List value expected.

Can be updated without replacement.

ok_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to ok.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

repeat_actions*ű*

False to trigger actions when the threshold is reached AND the alarms state has changed. By default, actions are called each time the threshold is reached.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

severity*ŕí*

Available since 5.0.0 (Liberty)

Severity of the alarm.

String value expected.

Can be updated without replacement.

Defaults to `"low"`

Allowed values: low, moderate, critical

time_constraints*ŕí*

Available since 5.0.0 (Liberty)

Describe time constraints for the alarm. Only evaluate the alarm if the time at evaluation is within this time constraint. Start point(s) of the constraint are specified with a cron expression, whereas its duration is given in seconds.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ŕí*

Optional.

Description for the time constraint.

String value expected.

Updates cause replacement.

duration*ŕí*

Required.

Duration for the time constraint.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

name*ŕí*

Required.

Name for the time constraint.

String value expected.

Updates cause replacement.

start*ŕí*

Required.

Start time for the time constraint. A CRON expression property.

String value expected.

Updates cause replacement.

Value must be of type cron_expression

timezone*ŕí*

Optional.

Timezone for the time constraint (eg. Asia/Taipei, Europe/Amsterdam).

String value expected.

Updates cause replacement.

Value must be of type timezone

Attributes

show*ŕí*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Aodh::CompositeAlarm
    properties:
      alarm_actions: [Value, Value, ...]
      alarm_queues: [String, String, ...]
      composite_rule: {"operator": String, "rules": [Value, Value, ...]}
      description: String
      enabled: Boolean
      insufficient_data_actions: [Value, Value, ...]
      insufficient_data_queues: [String, String, ...]
      ok_actions: [Value, Value, ...]
      ok_queues: [String, String, ...]
      repeat_actions: Boolean
```

(continues on next page)

(continued from previous page)

```
    severity: String
    time_constraints: [{"name": String, "start": String, "description":␣
↪String, "duration": Integer, "timezone": String}, {"name": String, "start":␣
↪String, "description": String, "duration": Integer, "timezone": String}, ...
↪]
```

OS::Aodh::EventAlarm

Available since 8.0.0 (Ocata)

A resource that implements event alarms.

Allows users to define alarms which can be evaluated based on events passed from other OpenStack services. The events can be emitted when the resources from other OpenStack services have been updated, created or deleted, such as compute.instance.reboot.end, scheduler.select_destinations.end.

Optional Properties

alarm_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to alarm.

List value expected.

Can be updated without replacement.

alarm_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to alarm.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

description*ű*

Description for the alarm.

String value expected.

Can be updated without replacement.

enabled*ű*

True if alarm evaluation/actioning is enabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

event_type*ű*

Event type to evaluate against. If not specified will match all events.

String value expected.

Can be updated without replacement.

Defaults to `"*"`

insufficient_data_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

insufficient_data_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

ok_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to ok.

List value expected.

Can be updated without replacement.

ok_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to ok.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

query*ű*

A list for filtering events. Query conditions used to filter specific events when evaluating the alarm.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

field*ű*

Optional.

Name of attribute to compare.

String value expected.

Can be updated without replacement.

op*ű*

Optional.

Comparison operator.

String value expected.

Can be updated without replacement.

Allowed values: le, ge, eq, lt, gt, ne

type*ű*

Optional.

The type of the attribute.

String value expected.

Can be updated without replacement.

Defaults to `"string"`

Allowed values: integer, float, string, boolean, datetime

value*ű*

Optional.

String value with which to compare.

String value expected.

Can be updated without replacement.

repeat_actions*ŕí*

False to trigger actions when the threshold is reached AND the alarms state has changed. By default, actions are called each time the threshold is reached.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

severity*ŕí*

Available since 5.0.0 (Liberty)

Severity of the alarm.

String value expected.

Can be updated without replacement.

Defaults to `"low"`

Allowed values: low, moderate, critical

time_constraints*ŕí*

Available since 5.0.0 (Liberty)

Describe time constraints for the alarm. Only evaluate the alarm if the time at evaluation is within this time constraint. Start point(s) of the constraint are specified with a cron expression, whereas its duration is given in seconds.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ŕí*

Optional.

Description for the time constraint.

String value expected.

Updates cause replacement.

duration*ű*

Required.

Duration for the time constraint.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

name*ű*

Required.

Name for the time constraint.

String value expected.

Updates cause replacement.

start*ű*

Required.

Start time for the time constraint. A CRON expression property.

String value expected.

Updates cause replacement.

Value must be of type cron_expression

timezone*ű*

Optional.

Timezone for the time constraint (eg. Asia/Taipei, Europe/Amsterdam).

String value expected.

Updates cause replacement.

Value must be of type timezone

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Aodh::EventAlarm
    properties:
```

(continues on next page)

(continued from previous page)

```
    alarm_actions: [Value, Value, ...]
    alarm_queues: [String, String, ...]
    description: String
    enabled: Boolean
    event_type: String
    insufficient_data_actions: [Value, Value, ...]
    insufficient_data_queues: [String, String, ...]
    ok_actions: [Value, Value, ...]
    ok_queues: [String, String, ...]
    query: [{"field": String, "type": String, "op": String, "value": String}
↪, {"field": String, "type": String, "op": String, "value": String}, ...]
    repeat_actions: Boolean
    severity: String
    time_constraints: [{"name": String, "start": String, "description":␣
↪String, "duration": Integer, "timezone": String}, {"name": String, "start":␣
↪String, "description": String, "duration": Integer, "timezone": String}, ...
↪]
```

OS::Aodh::GnocchiAggregationByMetricsAlarm

Available since 2015.1 (Kilo)

A resource that implements alarm with specified metrics.

A resource that implements alarm which allows to use specified by user metrics in metrics list.

Required Properties

metrics*ű*

A list of metric ids.

List value expected.

Can be updated without replacement.

threshold*ű*

Threshold to evaluate against.

Number value expected.

Can be updated without replacement.

Optional Properties

aggregation_method*ű*

The aggregation method to compare to the threshold.

String value expected.

Can be updated without replacement.

alarm_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to alarm.

List value expected.

Can be updated without replacement.

alarm_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to alarm.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

comparison_operator*ű*

Operator used to compare specified statistic with threshold.

String value expected.

Can be updated without replacement.

Allowed values: le, ge, eq, lt, gt, ne

description*ű*

Description for the alarm.

String value expected.

Can be updated without replacement.

enabled*ű*

True if alarm evaluation/actioning is enabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

evaluation_periods*ű*

Number of periods to evaluate over.

Integer value expected.

Can be updated without replacement.

granularity*ű*

The time range in seconds.

Integer value expected.

Can be updated without replacement.

insufficient_data_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

insufficient_data_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

ok_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to ok.

List value expected.

Can be updated without replacement.

ok_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to ok.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

repeat_actions*ű*

False to trigger actions when the threshold is reached AND the alarms state has changed. By default, actions are called each time the threshold is reached.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

severity*ű*

Available since 5.0.0 (Liberty)

Severity of the alarm.

String value expected.

Can be updated without replacement.

Defaults to `"low"`

Allowed values: low, moderate, critical

time_constraints*ű*

Available since 5.0.0 (Liberty)

Describe time constraints for the alarm. Only evaluate the alarm if the time at evaluation is within this time constraint. Start point(s) of the constraint are specified with a cron expression, whereas its duration is given in seconds.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ű*

Optional.

Description for the time constraint.

String value expected.

Updates cause replacement.

duration*ű*

Required.

Duration for the time constraint.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

name*ű*

Required.

Name for the time constraint.

String value expected.

Updates cause replacement.

start*ű*

Required.

Start time for the time constraint. A CRON expression property.

String value expected.

Updates cause replacement.

Value must be of type cron_expression

timezone*ű*

Optional.

Timezone for the time constraint (eg. Asia/Taipei, Europe/Amsterdam).

String value expected.

Updates cause replacement.

Value must be of type timezone

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Aodh::GnocchiAggregationByMetricsAlarm
    properties:
      aggregation_method: String
      alarm_actions: [Value, Value, ...]
      alarm_queues: [String, String, ...]
      comparison_operator: String
```

(continues on next page)

(continued from previous page)

```
    description: String
    enabled: Boolean
    evaluation_periods: Integer
    granularity: Integer
    insufficient_data_actions: [Value, Value, ...]
    insufficient_data_queues: [String, String, ...]
    metrics: [Value, Value, ...]
    ok_actions: [Value, Value, ...]
    ok_queues: [String, String, ...]
    repeat_actions: Boolean
    severity: String
    threshold: Number
    time_constraints: [{"name": String, "start": String, "description":␣
↪String, "duration": Integer, "timezone": String}, {"name": String, "start":␣
↪String, "description": String, "duration": Integer, "timezone": String}, ...
↪]
```

OS::Aodh::GnocchiAggregationByResourcesAlarm

Available since 2015.1 (Kilo)

A resource that implements alarm as an aggregation of resources alarms.

A resource that implements alarm which uses aggregation of resources alarms with some condition. If state of a system is satisfied alarm condition, alarm is activated.

Required Properties

metric*ű*

Metric name watched by the alarm.

String value expected.

Can be updated without replacement.

query*ű*

The query to filter the metrics.

String value expected.

Can be updated without replacement.

resource_type*ű*

Resource type.

String value expected.

Can be updated without replacement.

threshold*ű*

Threshold to evaluate against.

Number value expected.

Can be updated without replacement.

Optional Properties

aggregation_method*ű*

The aggregation method to compare to the threshold.

String value expected.

Can be updated without replacement.

alarm_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to alarm.

List value expected.

Can be updated without replacement.

alarm_queues*ű*

> Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to alarm.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

> Optional.
>
> String value expected.
>
> Can be updated without replacement.
>
> Value must be of type zaqar.queue

comparison_operator*ű*

Operator used to compare specified statistic with threshold.

String value expected.

Can be updated without replacement.

Allowed values: le, ge, eq, lt, gt, ne

description*ű*

Description for the alarm.

String value expected.

Can be updated without replacement.

enabled*ű*

True if alarm evaluation/actioning is enabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

evaluation_periods*ří*

Number of periods to evaluate over.

Integer value expected.

Can be updated without replacement.

granularity*ří*

The time range in seconds.

Integer value expected.

Can be updated without replacement.

insufficient_data_actions*ří*

A list of URLs (webhooks) to invoke when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

insufficient_data_queues*ří*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

ok_actions*ří*

A list of URLs (webhooks) to invoke when state transitions to ok.

List value expected.

Can be updated without replacement.

ok_queues*ří*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to ok.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

repeat_actions*ű*

False to trigger actions when the threshold is reached AND the alarms state has changed. By default, actions are called each time the threshold is reached.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

severity*ű*

Available since 5.0.0 (Liberty)

Severity of the alarm.

String value expected.

Can be updated without replacement.

Defaults to `"low"`

Allowed values: low, moderate, critical

time_constraints*ű*

Available since 5.0.0 (Liberty)

Describe time constraints for the alarm. Only evaluate the alarm if the time at evaluation is within this time constraint. Start point(s) of the constraint are specified with a cron expression, whereas its duration is given in seconds.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ű*

Optional.

Description for the time constraint.

String value expected.

Updates cause replacement.

duration*ű*

Required.

Duration for the time constraint.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

name*ű*

Required.

Name for the time constraint.

String value expected.

Updates cause replacement.

start*ű*

Required.

Start time for the time constraint. A CRON expression property.

String value expected.

Updates cause replacement.

Value must be of type cron_expression

timezone*ű*

Optional.

Timezone for the time constraint (eg. Asia/Taipei, Europe/Amsterdam).

String value expected.

Updates cause replacement.

Value must be of type timezone

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Aodh::GnocchiAggregationByResourcesAlarm
    properties:
      aggregation_method: String
      alarm_actions: [Value, Value, ...]
      alarm_queues: [String, String, ...]
      comparison_operator: String
      description: String
      enabled: Boolean
      evaluation_periods: Integer
      granularity: Integer
      insufficient_data_actions: [Value, Value, ...]
      insufficient_data_queues: [String, String, ...]
      metric: String
      ok_actions: [Value, Value, ...]
      ok_queues: [String, String, ...]
      query: String
      repeat_actions: Boolean
      resource_type: String
      severity: String
      threshold: Number
      time_constraints: [{"name": String, "start": String, "description":␣
↪String, "duration": Integer, "timezone": String}, {"name": String, "start":␣
↪String, "description": String, "duration": Integer, "timezone": String}, ...
↪]
```

OS::Aodh::GnocchiResourcesAlarm

Available since 2015.1 (Kilo)

A resource allowing for the watch of some specified resource.

An alarm that evaluates threshold based on some metric for the specified resource.

Required Properties

metric*ŕ*

Metric name watched by the alarm.

String value expected.

Can be updated without replacement.

resource_id*ŕ*

Id of a resource.

String value expected.

Can be updated without replacement.

resource_type*ú*

Resource type.

String value expected.

Can be updated without replacement.

threshold*ú*

Threshold to evaluate against.

Number value expected.

Can be updated without replacement.

Optional Properties

aggregation_method*ú*

The aggregation method to compare to the threshold.

String value expected.

Can be updated without replacement.

alarm_actions*ú*

A list of URLs (webhooks) to invoke when state transitions to alarm.

List value expected.

Can be updated without replacement.

alarm_queues*ú*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to alarm.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

comparison_operator*ú*

Operator used to compare specified statistic with threshold.

String value expected.

Can be updated without replacement.

Allowed values: le, ge, eq, lt, gt, ne

description*ŕ*

Description for the alarm.

String value expected.

Can be updated without replacement.

enabled*ŕ*

True if alarm evaluation/actioning is enabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

evaluation_periods*ŕ*

Number of periods to evaluate over.

Integer value expected.

Can be updated without replacement.

granularity*ŕ*

The time range in seconds.

Integer value expected.

Can be updated without replacement.

insufficient_data_actions*ŕ*

A list of URLs (webhooks) to invoke when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

insufficient_data_queues*ŕ*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

ok_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to ok.

List value expected.

Can be updated without replacement.

ok_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to ok.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

repeat_actions*ű*

False to trigger actions when the threshold is reached AND the alarms state has changed. By default, actions are called each time the threshold is reached.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

severity*ű*

Available since 5.0.0 (Liberty)

Severity of the alarm.

String value expected.

Can be updated without replacement.

Defaults to `"low"`

Allowed values: low, moderate, critical

time_constraints*ű*

Available since 5.0.0 (Liberty)

Describe time constraints for the alarm. Only evaluate the alarm if the time at evaluation is within this time constraint. Start point(s) of the constraint are specified with a cron expression, whereas its duration is given in seconds.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ǐ*

Optional.

Description for the time constraint.

String value expected.

Updates cause replacement.

duration*ǐ*

Required.

Duration for the time constraint.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

name*ǐ*

Required.

Name for the time constraint.

String value expected.

Updates cause replacement.

start*ǐ*

Required.

Start time for the time constraint. A CRON expression property.

String value expected.

Updates cause replacement.

Value must be of type cron_expression

timezone*ǐ*

Optional.

Timezone for the time constraint (eg. Asia/Taipei, Europe/Amsterdam).

String value expected.

Updates cause replacement.

Value must be of type timezone

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Aodh::GnocchiResourcesAlarm
    properties:
      aggregation_method: String
      alarm_actions: [Value, Value, ...]
      alarm_queues: [String, String, ...]
      comparison_operator: String
      description: String
      enabled: Boolean
      evaluation_periods: Integer
      granularity: Integer
      insufficient_data_actions: [Value, Value, ...]
      insufficient_data_queues: [String, String, ...]
      metric: String
      ok_actions: [Value, Value, ...]
      ok_queues: [String, String, ...]
      repeat_actions: Boolean
      resource_id: String
      resource_type: String
      severity: String
      threshold: Number
      time_constraints: [{"name": String, "start": String, "description": String, "duration": Integer, "timezone": String}, {"name": String, "start": String, "description": String, "duration": Integer, "timezone": String}, ...]
```

OS::Aodh::LBMemberHealthAlarm

Available since 13.0.0 (Train)

A resource that implements a Loadbalancer Member Health Alarm.

Allows setting alarms based on the health of load balancer pool members, where the health of a member is determined by the member reporting an operating_status of ERROR beyond an initial grace period after creation (120 seconds by default).

Required Properties

autoscaling_group_id*ű*

ID of the Heat autoscaling group that contains the loadbalancer members. Unhealthy members will be marked as such before an update is triggered on the root stack.

String value expected.

Can be updated without replacement.

pool*ű*

Name or ID of the loadbalancer pool for which the health of each member will be evaluated.

String value expected.

Can be updated without replacement.

stack*ű*

Name or ID of the root / top level Heat stack containing the loadbalancer pool and members. An update will be triggered on the root Stack if an unhealthy member is detected in the loadbalancer pool.

String value expected.

Updates cause replacement.

Optional Properties

alarm_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to alarm.

List value expected.

Can be updated without replacement.

alarm_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to alarm.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

description*ű*

Description for the alarm.

String value expected.

Can be updated without replacement.

enabled*ű*

True if alarm evaluation/actioning is enabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

insufficient_data_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

insufficient_data_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

ok_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to ok.

List value expected.

Can be updated without replacement.

ok_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to ok.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

repeat_actions*ű*

False to trigger actions when the threshold is reached AND the alarms state has changed. By default, actions are called each time the threshold is reached.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

severity*ű*

Available since 5.0.0 (Liberty)

Severity of the alarm.

String value expected.

Can be updated without replacement.

Defaults to `"low"`

Allowed values: low, moderate, critical

time_constraints*ű*

Available since 5.0.0 (Liberty)

Describe time constraints for the alarm. Only evaluate the alarm if the time at evaluation is within this time constraint. Start point(s) of the constraint are specified with a cron expression, whereas its duration is given in seconds.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ű*

Optional.

Description for the time constraint.

String value expected.

Updates cause replacement.

duration*ű*

Required.

Duration for the time constraint.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

name*ű*

Required.

Name for the time constraint.

String value expected.

Updates cause replacement.

start*ű*

Required.

Start time for the time constraint. A CRON expression property.

String value expected.

Updates cause replacement.

Value must be of type cron_expression

timezone*ű*

Optional.

Timezone for the time constraint (eg. Asia/Taipei, Europe/Amsterdam).

String value expected.

Updates cause replacement.

Value must be of type timezone

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Aodh::LBMemberHealthAlarm
    properties:
      alarm_actions: [Value, Value, ...]
      alarm_queues: [String, String, ...]
      autoscaling_group_id: String
      description: String
      enabled: Boolean
      insufficient_data_actions: [Value, Value, ...]
      insufficient_data_queues: [String, String, ...]
      ok_actions: [Value, Value, ...]
      ok_queues: [String, String, ...]
      pool: String
      repeat_actions: Boolean
      severity: String
      stack: String
      time_constraints: [{"name": String, "start": String, "description":␣
↪String, "duration": Integer, "timezone": String}, {"name": String, "start":␣
↪String, "description": String, "duration": Integer, "timezone": String}, ...
↪]
```

OS::Aodh::PrometheusAlarm

Available since 22.0.0

A resource that implements Aodh alarm of type prometheus.

An alarm that evaluates threshold based on metric data fetched from Prometheus.

Required Properties

query*ű*

The PromQL query string to fetch metrics data from Prometheus.

String value expected.

Can be updated without replacement.

threshold*ű*

Threshold to evaluate against.

Number value expected.

Can be updated without replacement.

Optional Properties

alarm_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to alarm.

List value expected.

Can be updated without replacement.

alarm_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to alarm.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

comparison_operator*ű*

Operator used to compare specified statistic with threshold.

String value expected.

Can be updated without replacement.

Allowed values: le, ge, eq, lt, gt, ne

description*ű*

Description for the alarm.

String value expected.

Can be updated without replacement.

enabled*ű*

True if alarm evaluation/actioning is enabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

insufficient_data_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

insufficient_data_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

ok_actions*ű*

A list of URLs (webhooks) to invoke when state transitions to ok.

List value expected.

Can be updated without replacement.

ok_queues*ű*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to ok.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

repeat_actions*ű*

False to trigger actions when the threshold is reached AND the alarms state has changed. By default, actions are called each time the threshold is reached.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

severity*ǘ*

Available since 5.0.0 (Liberty)

Severity of the alarm.

String value expected.

Can be updated without replacement.

Defaults to `"low"`

Allowed values: low, moderate, critical

time_constraints*ǘ*

Available since 5.0.0 (Liberty)

Describe time constraints for the alarm. Only evaluate the alarm if the time at evaluation is within this time constraint. Start point(s) of the constraint are specified with a cron expression, whereas its duration is given in seconds.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ǘ*

Optional.

Description for the time constraint.

String value expected.

Updates cause replacement.

duration*ǘ*

Required.

Duration for the time constraint.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

name*ǘ*

Required.

Name for the time constraint.

String value expected.

Updates cause replacement.

start*ű*

Required.

Start time for the time constraint. A CRON expression property.

String value expected.

Updates cause replacement.

Value must be of type cron_expression

timezone*ű*

Optional.

Timezone for the time constraint (eg. Asia/Taipei, Europe/Amsterdam).

String value expected.

Updates cause replacement.

Value must be of type timezone

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Aodh::PrometheusAlarm
    properties:
      alarm_actions: [Value, Value, ...]
      alarm_queues: [String, String, ...]
      comparison_operator: String
      description: String
      enabled: Boolean
      insufficient_data_actions: [Value, Value, ...]
      insufficient_data_queues: [String, String, ...]
      ok_actions: [Value, Value, ...]
      ok_queues: [String, String, ...]
      query: String
      repeat_actions: Boolean
      severity: String
      threshold: Number
```

(continues on next page)

(continued from previous page)

```
    time_constraints: [{"name": String, "start": String, "description":␣
↪String, "duration": Integer, "timezone": String}, {"name": String, "start":␣
↪String, "description": String, "duration": Integer, "timezone": String}, ...
↪]
```

OS::Barbican::CertificateContainer

Available since 6.0.0 (Mitaka)

A resource for creating barbican certificate container.

A certificate container is used for storing the secrets that are relevant to certificates.

Optional Properties

certificate_ref*ű*

Reference to certificate.

String value expected.

Updates cause replacement.

Value must be of type barbican.secret

intermediates_ref*ű*

Reference to intermediates.

String value expected.

Updates cause replacement.

Value must be of type barbican.secret

name*ű*

Human-readable name for the container.

String value expected.

Updates cause replacement.

private_key_passphrase_ref*ű*

Reference to private key passphrase.

String value expected.

Updates cause replacement.

Value must be of type barbican.secret

private_key_ref*ű*

Reference to private key.

String value expected.

Updates cause replacement.

Value must be of type barbican.secret

Attributes

consumers*ú*
The URIs to container consumers.

container_ref*ú*
The URI to the container.

secret_refs*ú*
The URIs to secrets stored in container.

show*ú*
Detailed information about resource.

status*ú*
The status of the container.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Barbican::CertificateContainer
    properties:
      certificate_ref: String
      intermediates_ref: String
      name: String
      private_key_passphrase_ref: String
      private_key_ref: String
```

OS::Barbican::GenericContainer

Available since 6.0.0 (Mitaka)

A resource for creating Barbican generic container.

A generic container is used for any type of secret that a user may wish to aggregate. There are no restrictions on the amount of secrets that can be held within this container.

Optional Properties

name*ú*

Human-readable name for the container.

String value expected.

Updates cause replacement.

secrets*ú*

References to secrets that will be stored in container.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

name*ű*

Required.

Name of the secret.

String value expected.

Updates cause replacement.

ref*ű*

Required.

Reference to the secret.

String value expected.

Updates cause replacement.

Value must be of type barbican.secret

Attributes

consumers*ű*
The URIs to container consumers.

container_ref*ű*
The URI to the container.

secret_refs*ű*
The URIs to secrets stored in container.

show*ű*
Detailed information about resource.

status*ű*
The status of the container.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Barbican::GenericContainer
```

(continues on next page)

(continued from previous page)

```
  properties:
    name: String
    secrets: [{"name": String, "ref": String}, {"name": String, "ref":␣
↪String}, ...]
```

OS::Barbican::Order

Available since 2014.2 (Juno)

A resource allowing for the generation secret material by Barbican.

The resource allows to generate some secret material. It can be, for example, some key or certificate. The order encapsulates the workflow and history for the creation of a secret. The time to generate a secret can vary depending on the type of secret.

Required Properties

type*ű*

Available since 5.0.0 (Liberty)

The type of the order.

String value expected.

Updates cause replacement.

Allowed values: key, asymmetric, certificate

Optional Properties

algorithm*ű*

The algorithm type used to generate the secret. Required for key and asymmetric types of order.

String value expected.

Updates cause replacement.

bit_length*ű*

The bit-length of the secret. Required for key and asymmetric types of order.

Integer value expected.

Updates cause replacement.

ca_id*ű*

Available since 5.0.0 (Liberty)

The identifier of the CA to use.

String value expected.

Updates cause replacement.

expiration*ŕ*

The expiration date for the secret in ISO-8601 format.

String value expected.

Updates cause replacement.

Value must be of type expiration

mode*ŕ*

The type/mode of the algorithm associated with the secret information.

String value expected.

Updates cause replacement.

name*ŕ*

Human readable name for the secret.

String value expected.

Updates cause replacement.

pass_phrase*ŕ*

Available since 5.0.0 (Liberty)

The passphrase of the created key. Can be set only for asymmetric type of order.

String value expected.

Updates cause replacement.

payload_content_type*ŕ*

The type/format the secret data is provided in.

String value expected.

Updates cause replacement.

profile*ŕ*

Available since 5.0.0 (Liberty)

The profile of certificate to use.

String value expected.

Updates cause replacement.

request_data*ŕ*

Available since 5.0.0 (Liberty)

The content of the CSR. Only for certificate orders.

String value expected.

Updates cause replacement.

request_type*ű*

Available since 5.0.0 (Liberty)

The type of the certificate request.

String value expected.

Updates cause replacement.

Allowed values: stored-key, simple-cmc, custom

source_container_ref*ű*

Available since 5.0.0 (Liberty)

The source of certificate request.

String value expected.

Updates cause replacement.

Value must be of type barbican.container

subject_dn*ű*

Available since 5.0.0 (Liberty)

The subject of the certificate request.

String value expected.

Updates cause replacement.

Attributes

certificate*ű*

Available since 5.0.0 (Liberty)

The payload of the created certificate, if available.

container_ref*ű*

Available since 5.0.0 (Liberty)

The URI to the created container.

intermediates*ű*

Available since 5.0.0 (Liberty)

The payload of the created intermediates, if available.

order_ref*ű*
The URI to the order.

private_key*ű*

Available since 5.0.0 (Liberty)

The payload of the created private key, if available.

public_key*ű*

Available since 5.0.0 (Liberty)

The payload of the created public key, if available.

secret_ref*ű*
The URI to the created secret.

show*ű*
Detailed information about resource.

status*ű*
The status of the order.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Barbican::Order
    properties:
      algorithm: String
      bit_length: Integer
      ca_id: String
      expiration: String
      mode: String
      name: String
      pass_phrase: String
      payload_content_type: String
      profile: String
      request_data: String
      request_type: String
      source_container_ref: String
      subject_dn: String
      type: String
```

OS::Barbican::RSAContainer

Available since 6.0.0 (Mitaka)

A resource for creating barbican RSA container.

An RSA container is used for storing RSA public keys, private keys, and private key pass phrases.

Optional Properties

name*ú*

Human-readable name for the container.

String value expected.

Updates cause replacement.

private_key_passphrase_ref*ú*

Reference to private key passphrase.

String value expected.

Updates cause replacement.

Value must be of type barbican.secret

private_key_ref*ú*

Reference to private key.

String value expected.

Updates cause replacement.

Value must be of type barbican.secret

public_key_ref*ú*

Reference to public key.

String value expected.

Updates cause replacement.

Value must be of type barbican.secret

Attributes

consumers*ú*
: The URIs to container consumers.

container_ref*ú*
: The URI to the container.

secret_refs*ú*
: The URIs to secrets stored in container.

show*ú*
: Detailed information about resource.

status*ű*
The status of the container.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Barbican::RSAContainer
    properties:
      name: String
      private_key_passphrase_ref: String
      private_key_ref: String
      public_key_ref: String
```

OS::Barbican::Secret

Available since 2014.2 (Juno)

The resource provides access to the secret/keying stored material.

A secret is a singular item that stored within Barbican. A secret is anything you want it to be; however, the formal use case is a key that you wish to store away from prying eyes. Secret may include private keys, passwords and so on.

Optional Properties

algorithm*ű*

The algorithm type used to generate the secret.

String value expected.

Updates cause replacement.

bit_length*ű*

The bit-length of the secret.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

expiration*ű*

The expiration date for the secret in ISO-8601 format.

String value expected.

Updates cause replacement.

Value must be of type expiration

mode*ŕí*

The type/mode of the algorithm associated with the secret information.

String value expected.

Updates cause replacement.

name*ŕí*

Human readable name for the secret.

String value expected.

Updates cause replacement.

payload*ŕí*

The unencrypted plain text of the secret.

String value expected.

Updates cause replacement.

payload_content_encoding*ŕí*

The encoding format used to provide the payload data.

String value expected.

Updates cause replacement.

Allowed values: base64

payload_content_type*ŕí*

The type/format the secret data is provided in.

String value expected.

Updates cause replacement.

Allowed values: text/plain, application/octet-stream

secret_type*ŕí*

Available since 5.0.0 (Liberty)

The type of the secret.

String value expected.

Updates cause replacement.

Defaults to `"opaque"`

Allowed values: symmetric, public, private, certificate, passphrase, opaque

Attributes

decrypted_payload*ŕí*
The decrypted secret payload.

show*ű*
Detailed information about resource.

status*ű*
The status of the secret.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Barbican::Secret
    properties:
      algorithm: String
      bit_length: Integer
      expiration: String
      mode: String
      name: String
      payload: String
      payload_content_encoding: String
      payload_content_type: String
      secret_type: String
```

OS::Blazar::Host

Available since 12.0.0 (Stein)

A resource to manage Blazar hosts.

Host resource manages the physical hosts for the lease/reservation within OpenStack.

TODO(asmita): Based on an agreement with Blazar team, this resource class does not support updating host resource as currently Blazar does not support to delete existing extra_capability keys while updating host. Also, in near future, when Blazar team will come up with a new alternative API to resolve this issue, we will need to modify this class.

Required Properties

name*ű*

The name of the host.

String value expected.

Updates cause replacement.

Optional Properties

extra_capability*ú*

The extra capability of the host.

Map value expected.

Updates cause replacement.

Attributes

cpu_info*ú*
Information of the CPU of the host.

created_at*ú*
The date and time when the host was created. The date and time format must be CCYY-MM-DD hh:mm.

extra_capability*ú*
The extra capability of the host.

hypervisor_hostname*ú*
The hypervisor name of the host.

hypervisor_type*ú*
The hypervisor type the host.

hypervisor_version*ú*
The hypervisor version of the host.

local_gb*ú*
Gigabytes of the disk of the host.

memory_mb*ú*
Megabytes of the memory of the host.

reservable*ú*
The flag which represents whether the host is reservable or not.

service_name*ú*
The compute service name of the host.

show*ú*
Detailed information about resource.

status*ú*
The status of the host.

trust_id*ú*
The UUID of the trust of the host operator.

updated_at*ú*
The date and time when the host was updated. The date and time format must be CCYY-MM-DD hh:mm.

vcpus*ú*
The number of the VCPUs of the host.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Blazar::Host
    properties:
      extra_capability: {...}
      name: String
```

OS::Blazar::Lease

Available since 12.0.0 (Stein)

A resource to manage Blazar leases.

Lease resource manages the reservations of specific type/amount of cloud resources within OpenStack.

Note: Based on an agreement with Blazar team, this resource class does not support updating, because current Blazar lease scheme is not suitable for Heat, if you want to update a lease, you need to specify reservations id, which is one of attribute of lease.

Required Properties

end_date*ű*

The end date and time of the lease The date and time format must be CCYY-MM-DD hh:mm.

String value expected.

Updates cause replacement.

Value must match pattern: \d{4}-\d{2}-\d{2}\s\d{2}:\d{2}

name*ű*

The name of the lease.

String value expected.

Updates cause replacement.

reservations*ű*

The list of reservations.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

affinity*ű*

Optional.

The affinity of instances to reserve.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

amount*ű*

Optional.

The amount of instances to reserve.

Integer value expected.

Updates cause replacement.

The value must be in the range 0 to 2147483647.

before_end*ű*

Optional.

The before-end-action of the reservation.

String value expected.

Updates cause replacement.

Defaults to `"default"`

Allowed values: default, snapshot

disk_gb*ű*

Optional.

Gigabytes of the local disk per the instance.

Integer value expected.

Updates cause replacement.

The value must be in the range 0 to 2147483647.

hypervisor_properties*ű*

Optional.

Properties of the hypervisor to reserve.

String value expected.

Updates cause replacement.

max*ű*

Optional.

The maximum number of hosts to reserve.

Integer value expected.

Updates cause replacement.

The value must be at least 1.

memory_mb*ú*

Optional.

Megabytes of memory per the instance.

Integer value expected.

Updates cause replacement.

The value must be in the range 0 to 2147483647.

min*ú*

Optional.

The minimum number of hosts to reserve.

Integer value expected.

Updates cause replacement.

The value must be at least 1.

resource_properties*ú*

Optional.

Properties of the resource to reserve.

String value expected.

Updates cause replacement.

resource_type*ú*

Required.

The type of the resource to reserve.

String value expected.

Updates cause replacement.

Allowed values: virtual:instance, physical:host

vcpus*ú*

Optional.

The number of VCPUs per the instance.

Integer value expected.

Updates cause replacement.

The value must be in the range 0 to 2147483647.

start_date*ú*

The start date and time of the lease. The date and time format must be CCYY-MM-DD hh:mm.

String value expected.

Updates cause replacement.

Value must match pattern: \d{4}-\d{2}-\d{2}\s\d{2}:\d{2}

Optional Properties

before_end_date*ű*

The date and time for the before-end-action of the lease. The date and time format must be CCYY-MM-DD hh:mm.

String value expected.

Updates cause replacement.

Value must match pattern: \d{4}-\d{2}-\d{2}\s\d{2}:\d{2}

events*ű*

A list of event objects.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

event_type*ű*

Required.

The type of the event (e.g. notification).

String value expected.

Updates cause replacement.

time*ű*

Required.

The date and time of the event. The date and time format must be CCYY-MM-DD hh:mm.

String value expected.

Updates cause replacement.

Attributes

created_at*ű*
The date and time when the lease was created. The date and time format is CCYY-MM-DD hh:mm.

degraded*ű*
The flag which represents condition of reserved resources of the lease. If it is true, the amount of reserved resources is less than the request or reserved resources were changed.

end_date*ű*
The end date and time of the lease. The date and time format is CCYY-MM-DD hh:mm.

events*ű*
Event information of the lease.

name*ű*
The name of the lease.

project_id*ű*
The UUID the project which owns the lease.

reservations*ű*
A list of reservation objects.

show*ű*
Detailed information about resource.

start_date*ű*
The start date and time of the lease. The date and time format is CCYY-MM-DD hh:mm.

status*ű*
The status of the lease.

trust_id*ű*
The UUID of the trust of the lease owner.

updated_at*ű*
The date and time when the lease was updated. The date and time format is CCYY-MM-DD hh:mm.

user_id*ű*
The UUID of the lease owner.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Blazar::Lease
    properties:
      before_end_date: String
      end_date: String
      events: [{"event_type": String, "time": String}, {"event_type": String,
↪"time": String}, ...]
      name: String
      reservations: [{"resource_type": String, "min": Integer, "max": Integer,
↪ "hypervisor_properties": String, "resource_properties": String, "before_end
↪": String, "amount": Integer, "vcpus": Integer, "memory_mb": Integer, "disk_
↪gb": Integer, "affinity": Boolean}, {"resource_type": String, "min":␣
↪Integer, "max": Integer, "hypervisor_properties": String, "resource_
↪properties": String, "before_end": String, "amount": Integer, "vcpus":␣
↪Integer, "memory_mb": Integer, "disk_gb": Integer, "affinity": Boolean}, ...
```

(continues on next page)

(continued from previous page)

```
↪]
    start_date: String
```

OS::Cinder::EncryptedVolumeType

Available since 5.0.0 (Liberty)

A resource for encrypting a cinder volume type.

A Volume Encryption Type is a collection of settings used to conduct encryption for a specific volume type.

Note that default cinder security policy usage of this resource is limited to being used by administrators only.

Required Properties

provider*ií*

The class that provides encryption support. For example, nova.volume.encryptors.luks.LuksEncryptor.

String value expected.

Can be updated without replacement.

volume_type*ií*

Name or id of volume type (OS::Cinder::VolumeType).

String value expected.

Updates cause replacement.

Value must be of type cinder.vtype

Optional Properties

cipher*ií*

The encryption algorithm or mode. For example, aes-xts-plain64.

String value expected.

Can be updated without replacement.

Allowed values: aes-xts-plain64, aes-cbc-essiv

control_location*ií*

Notional service where encryption is performed For example, front-end. For Nova.

String value expected.

Can be updated without replacement.

Defaults to `"front-end"`

Allowed values: front-end, back-end

key_size*ű*

Size of encryption key, in bits. For example, 128 or 256.

Integer value expected.

Can be updated without replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Cinder::EncryptedVolumeType
    properties:
      cipher: String
      control_location: String
      key_size: Integer
      provider: String
      volume_type: String
```

OS::Cinder::QoSAssociation

Available since 8.0.0 (Ocata)

A resource to associate cinder QoS specs with volume types.

Usage of this resource restricted to admins only by default policy.

Required Properties

qos_specs*ű*

ID or Name of the QoS specs.

String value expected.

Updates cause replacement.

Value must be of type cinder.qos_specs

volume_types*ű*

List of volume type IDs or Names to be attached to QoS specs.

List value expected.

Can be updated without replacement.

List contents:

Optional.

A volume type to attach specs.

String value expected.

Can be updated without replacement.

Value must be of type cinder.vtype

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Cinder::QoSAssociation
    properties:
      qos_specs: String
      volume_types: [String, String, ...]
```

OS::Cinder::QoSSpecs

Available since 7.0.0 (Newton)

A resource for creating cinder QoS specs.

Users can ask for a specific volume type. Part of that volume type is a string that defines the QoS of the volume IO (fast, normal, or slow). Backends that can handle all of the demands of the volume type become candidates for scheduling. Usage of this resource restricted to admins only by default policy.

Required Properties

specs*ű*

The specs key and value pairs of the QoS.

Map value expected.

Can be updated without replacement.

Optional Properties

name*ű*

Name of the QoS.

String value expected.

Updates cause replacement.

Attributes

show*ú*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Cinder::QoSSpecs
    properties:
      name: String
      specs: {...}
```

OS::Cinder::Quota

Available since 7.0.0 (Newton)

A resource for creating cinder quotas.

Cinder Quota is used to manage operational limits for projects. Currently, this resource can manage Cinders gigabytes, snapshots, and volumes quotas.

Note that default cinder security policy usage of this resource is limited to being used by administrators only. Administrators should be careful to create only one Cinder Quota resource per project, otherwise it will be hard for them to manage the quota properly.

Required Properties

project*ú*

OpenStack Keystone Project.

String value expected.

Updates cause replacement.

Value must be of type keystone.project

Optional Properties

backup_gigabytes*ú*

Available since 16.0.0 (Wallaby)

Quota for the amount of backups disk space (in Gigabytes). Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

backups*ű*

Available since 16.0.0 (Wallaby)

Quota for the number of backups. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

gigabytes*ű*

Quota for the amount of disk space (in Gigabytes). Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

snapshots*ű*

Quota for the number of snapshots. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

volumes*ű*

Quota for the number of volumes. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
```

(continues on next page)

(continued from previous page)

```
    type: OS::Cinder::Quota
    properties:
      backup_gigabytes: Integer
      backups: Integer
      gigabytes: Integer
      project: String
      snapshots: Integer
      volumes: Integer
```

OS::Cinder::Volume

A resource that implements Cinder volumes.

Cinder volume is a storage in the form of block devices. It can be used, for example, for providing storage to instance. Volume supports creation from snapshot, backup or image. Also volume can be created only by size.

Optional Properties

availability_zone*ű*

The availability zone in which the volume will be created.

String value expected.

Updates cause replacement.

backup_id*ű*

If specified, the backup to create the volume from.

String value expected.

Can be updated without replacement.

Value must be of type cinder.backup

description*ű*

A description of the volume.

String value expected.

Can be updated without replacement.

image*ű*

If specified, the name or ID of the image to create the volume from.

String value expected.

Updates cause replacement.

Value must be of type glance.image

metadata*ű*

Key/value pairs to associate with the volume.

Map value expected.

Can be updated without replacement.

Defaults to {}

name*ű*

A name used to distinguish the volume.

String value expected.

Can be updated without replacement.

read_only*ű*

Available since 5.0.0 (Liberty)

Enables or disables read-only access mode of volume.

Boolean value expected.

Can be updated without replacement.

scheduler_hints*ű*

Available since 2015.1 (Kilo)

Arbitrary key-value pairs specified by the client to help the Cinder scheduler creating a volume.

Map value expected.

Updates cause replacement.

size*ű*

The size of the volume in GB. On update only increase in size is supported. This property is required unless property backup_id or source_volid or snapshot_id is specified.

Integer value expected.

Can be updated without replacement.

The value must be at least 1.

snapshot_id*ű*

If specified, the snapshot to create the volume from.

String value expected.

Updates cause replacement.

Value must be of type cinder.snapshot

source_volid*ű*

If specified, the volume to use as source.

String value expected.

Updates cause replacement.

Value must be of type cinder.volume

volume_type*ŕ*

If specified, the type of volume to use, mapping to a specific backend.

String value expected.

Can be updated without replacement.

Value must be of type cinder.vtype

Attributes

attachments*ŕ*

DEPRECATED since 9.0.0 (Pike) - Use property attachments_list.

Available since 2015.1 (Kilo)

A string representation of the list of attachments of the volume.

attachments_list*ŕ*

Available since 9.0.0 (Pike)

The list of attachments of the volume.

availability_zone*ŕ*
The availability zone in which the volume is located.

bootable*ŕ*
Boolean indicating if the volume can be booted or not.

created_at*ŕ*
The timestamp indicating volume creation.

display_description*ŕ*
Description of the volume.

display_name*ŕ*
Name of the volume.

encrypted*ŕ*
Boolean indicating if the volume is encrypted or not.

metadata*ŕ*
Key/value pairs associated with the volume.

metadata_values*ŕ*
Key/value pairs associated with the volume in raw dict form.

multiattach*ŕ*

Available since 6.0.0 (Mitaka)

Boolean indicating whether allow the volume to be attached more than once.

show*ű*
Detailed information about resource.

size*ű*
The size of the volume in GB.

snapshot_id*ű*
The snapshot the volume was created from, if any.

source_volid*ű*
The volume used as source, if any.

status*ű*
The current status of the volume.

volume_type*ű*
The type of the volume mapping to a backend, if any.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Cinder::Volume
    properties:
      availability_zone: String
      backup_id: String
      description: String
      image: String
      metadata: {...}
      name: String
      read_only: Boolean
      scheduler_hints: {...}
      size: Integer
      snapshot_id: String
      source_volid: String
      volume_type: String
```

OS::Cinder::VolumeAttachment

Resource for associating volume to instance.

Resource for associating existing volume to instance. Also, the location where the volume is exposed on the instance can be specified.

Required Properties

instance_uuid*ű*

The ID of the server to which the volume attaches.

String value expected.

Can be updated without replacement.

volume_id*ű*

The ID of the volume to be attached.

String value expected.

Can be updated without replacement.

Value must be of type cinder.volume

Optional Properties

mountpoint*ű*

The location where the volume is exposed on the instance. This assignment may not be honored and it is advised that the path /dev/disk/by-id/virtio-<VolumeId> be used instead.

String value expected.

Can be updated without replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Cinder::VolumeAttachment
    properties:
      instance_uuid: String
      mountpoint: String
      volume_id: String
```

OS::Cinder::VolumeType

Available since 2015.1 (Kilo)

A resource for creating cinder volume types.

Volume type resource allows to define, whether volume, which will be use this type, will public and which projects are allowed to work with it. Also, there can be some user-defined metadata.

Note that default cinder security policy usage of this resource is limited to being used by administrators only.

Required Properties

name*ŕí*

Name of the volume type.

String value expected.

Can be updated without replacement.

Optional Properties

description*ŕí*

Available since 5.0.0 (Liberty)

Description of the volume type.

String value expected.

Can be updated without replacement.

is_public*ŕí*

Available since 5.0.0 (Liberty)

Whether the volume type is accessible to the public.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

metadata*ŕí*

The extra specs key and value pairs of the volume type.

Map value expected.

Can be updated without replacement.

projects*ŕí*

Available since 5.0.0 (Liberty)

Projects to add volume type access to. NOTE: This property is only supported since Cinder API V2.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type keystone.project

Attributes

show*ú*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Cinder::VolumeType
    properties:
      description: String
      is_public: Boolean
      metadata: {...}
      name: String
      projects: [String, String, ...]
```

OS::Designate::RecordSet

Available since 8.0.0 (Ocata)

Heat Template Resource for Designate RecordSet.

Designate provides DNS-as-a-Service services for OpenStack. RecordSet helps to add more than one records.

Required Properties

records*ú*

A list of data for this RecordSet. Each item will be a separate record in Designate These items should conform to the DNS spec for the record type - e.g. A records must be IPv4 addresses, CNAME records must be a hostname. DNS record data varies based on the type of record. For more details, please refer rfc 1035.

List value expected.

Can be updated without replacement.

type*ú*

DNS RecordSet type.

String value expected.

Updates cause replacement.

Allowed values: A, AAAA, CNAME, MX, SRV, TXT, SPF, NS, PTR, SSHFP, SOA, CAA, CERT, NAPTR

zone*ű*

DNS Zone id or name.

String value expected.

Updates cause replacement.

Value must be of type designate.zone

Optional Properties

description*ű*

Description of RecordSet.

String value expected.

Can be updated without replacement.

The length must be no greater than 160.

name*ű*

RecordSet name.

String value expected.

Updates cause replacement.

The length must be no greater than 255.

ttl*ű*

Time To Live (Seconds).

Integer value expected.

Can be updated without replacement.

The value must be in the range 1 to 2147483647.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Designate::RecordSet
    properties:
      description: String
```

(continues on next page)

(continued from previous page)

```
    name: String
    records: [Value, Value, ...]
    ttl: Integer
    type: String
    zone: String
```

OS::Designate::Zone

Available since 8.0.0 (Ocata)

Heat Template Resource for Designate Zone.

Designate provides DNS-as-a-Service services for OpenStack. So, zone, part of domain is a realm with an identification string, unique in DNS.

Required Properties

name*ű*

DNS Name for the zone.

String value expected.

Updates cause replacement.

The length must be no greater than 255.

Optional Properties

attributes*ű*

Available since 24.0.0

Key:Value pairs of information about this zone, and the pool the user would like to place the zone in. This information can be used by the scheduler to place zones on the correct pool.

Map value expected.

Updates cause replacement.

description*ű*

Description of zone.

String value expected.

Can be updated without replacement.

The length must be no greater than 160.

email*ű*

E-mail for the zone. Used in SOA records for the zone. It is required for PRIMARY Type, otherwise ignored.

String value expected.

Can be updated without replacement.

primaries*ű*

The primary servers to transfer DNS zone information from. Mandatory for zone type SECONDARY, otherwise ignored.

List value expected.

Can be updated without replacement.

ttl*ű*

Time To Live (Seconds) for the zone.

Integer value expected.

Can be updated without replacement.

The value must be in the range 1 to 2147483647.

type*ű*

Type of zone. PRIMARY is controlled by Designate, SECONDARY zones are transferred from another DNS Server.

String value expected.

Updates cause replacement.

Defaults to `"PRIMARY"`

Allowed values: PRIMARY, SECONDARY

masters*ű*

DEPRECATED since 15.0.0 (Victoria) - Use "primaries" instead.

The primary servers to transfer DNS zone information from. Mandatory for zone type SECONDARY, otherwise ignored.

List value expected.

Can be updated without replacement.

Attributes

serial*ű*
DNS zone serial number.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
```

(continues on next page)

(continued from previous page)

```
  the_resource:
    type: OS::Designate::Zone
    properties:
      attributes: {...}
      description: String
      email: String
      name: String
      primaries: [Value, Value, ...]
      ttl: Integer
      type: String
```

OS::Glance::WebImage

Available since 12.0.0 (Stein)

A resource managing images in Glance using web-download import.

This provides image support for recent Glance installation.

Required Properties

container_format*ří*

Container format of image.

String value expected.

Updates cause replacement.

Allowed values: ami, ari, aki, bare, ovf, ova, docker

disk_format*ří*

Disk format of image.

String value expected.

Updates cause replacement.

Allowed values: ami, ari, aki, vhd, vhdx, vmdk, raw, qcow2, vdi, iso, ploop

location*ří*

URL where the data for this image already resides. For example, if the image data is stored in swift, you could specify swift://example.com/container/obj.

String value expected.

Updates cause replacement.

Optional Properties

active*ří*

Available since 16.0.0 (Wallaby)

Activate or deactivate the image. Requires Admin Access.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

architecture*ű*

Operating system architecture.

String value expected.

Can be updated without replacement.

extra_properties*ű*

Available since 17.0.0 (Xena)

Arbitrary properties to associate with the image.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

id*ű*

The image ID. Glance will generate a UUID if not specified.

String value expected.

Updates cause replacement.

kernel_id*ű*

ID of image stored in Glance that should be used as the kernel when booting an AMI-style image.

String value expected.

Can be updated without replacement.

Value must match pattern: ^([0-9a-fA-F]){8}-([0-9a-fA-F]){4}-([0-9a-fA-F]){4}-([0-9a-fA-F]){4}-([0-9a-fA-F]){12}$

members*ű*

Available since 16.0.0 (Wallaby)

List of additional members that are permitted to read the image. This may be a Keystone Project IDs or User IDs, depending on the Glance configuration in use.

List value expected.

Can be updated without replacement.

List contents:

Optional.

A member ID. This may be a Keystone Project ID or User ID, depending on the Glance configuration in use.

String value expected.

Can be updated without replacement.

min_disk*ú*

Amount of disk space (in GB) required to boot image. Default value is 0 if not specified and means no limit on the disk size.

Integer value expected.

Updates cause replacement.

Defaults to `0`

The value must be at least 0.

min_ram*ú*

Amount of ram (in MB) required to boot image. Default value is 0 if not specified and means no limit on the ram size.

Integer value expected.

Updates cause replacement.

Defaults to `0`

The value must be at least 0.

name*ú*

Name for the image. The name of an image is not unique to a Image Service node.

String value expected.

Updates cause replacement.

os_distro*ú*

The common name of the operating system distribution in lowercase.

String value expected.

Can be updated without replacement.

os_version*ú*

Operating system version as specified by the distributor.

String value expected.

Can be updated without replacement.

owner*ú*

Owner of the image.

String value expected.

Can be updated without replacement.

protected*ű*

Whether the image can be deleted. If the value is True, the image is protected and cannot be deleted.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

ramdisk_id*ű*

ID of image stored in Glance that should be used as the ramdisk when booting an AMI-style image.

String value expected.

Can be updated without replacement.

Value must match pattern: ^([0-9a-fA-F]){8}-([0-9a-fA-F]){4}-([0-9a-fA-F]){4}-([0-9a-fA-F]){4}-([0-9a-fA-F]){12}$

tags*ű*

List of image tags.

List value expected.

Can be updated without replacement.

visibility*ű*

Scope of image accessibility.

String value expected.

Can be updated without replacement.

Defaults to `"private"`

Allowed values: public, private, community, shared

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Glance::WebImage
    properties:
      active: Boolean
      architecture: String
      container_format: String
      disk_format: String
      extra_properties: {...}
```

(continues on next page)

(continued from previous page)

```
      id: String
      kernel_id: String
      location: String
      members: [String, String, ...]
      min_disk: Integer
      min_ram: Integer
      name: String
      os_distro: String
      os_version: String
      owner: String
      protected: Boolean
      ramdisk_id: String
      tags: [Value, Value, ...]
      visibility: String
```

OS::Heat::AccessPolicy

Resource for defining which resources can be accessed by users.

NOTE: Now this resource is actually associated with an AWS user resource, not any OS:: resource though it is registered under the OS namespace below.

Resource for defining resources that users are allowed to access by the DescribeStackResource API.

Required Properties

AllowedResources*ű*

Resources that users are allowed to access by the DescribeStackResource API.

List value expected.

Updates cause replacement.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::AccessPolicy
    properties:
      AllowedResources: [Value, Value, ...]
```

OS::Heat::AutoScalingGroup

Available since 2014.1 (Icehouse)

An autoscaling group that can scale arbitrary resources.

An autoscaling group allows the creation of a desired count of similar resources, which are defined with the resource property in HOT format. If there is a need to create many of the same resources (e.g. one hundred sets of Server, WaitCondition and WaitConditionHandle or even Neutron Nets), AutoScalingGroup is a convenient and easy way to do that.

Required Properties

max_size*ű*

Maximum number of resources in the group.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

min_size*ű*

Minimum number of resources in the group.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

resource*ű*

Resource definition for the resources in the group, in HOT format. The value of this property is the definition of a resource just as if it had been declared in the template itself.

Map value expected.

Can be updated without replacement.

Optional Properties

cooldown*ű*

Cooldown period, in seconds.

Integer value expected.

Can be updated without replacement.

desired_capacity*ű*

Desired initial number of resources.

Integer value expected.

Can be updated without replacement.

rolling_updates*ű*

Policy for rolling updates for this scaling group.

Map value expected.

Can be updated without replacement.

Defaults to `{"min_in_service": 0, "max_batch_size": 1, "pause_time": 0}`

Map properties:

max_batch_size*ű*

Optional.

The maximum number of resources to replace at once.

Integer value expected.

Can be updated without replacement.

Defaults to `1`

The value must be at least 1.

min_in_service*ű*

Optional.

The minimum number of resources in service while rolling updates are being executed.

Integer value expected.

Can be updated without replacement.

Defaults to `0`

The value must be at least 0.

pause_time*ű*

Optional.

The number of seconds to wait between batches of updates.

Number value expected.

Can be updated without replacement.

Defaults to `0`

The value must be at least 0.

Attributes

current_size*ű*

Available since 2015.1 (Kilo)

The current size of AutoscalingResourceGroup.

outputs*ű*

Available since 2014.2 (Juno)

A map of resource names to the specified attribute of each individual resource that is part of the AutoScalingGroup. This map specifies output parameters that are available once the AutoScalingGroup has been instantiated.

outputs_list*ű*

Available since 2014.2 (Juno)

A list of the specified attribute of each individual resource that is part of the AutoScalingGroup. This list of attributes is available as an output once the AutoScalingGroup has been instantiated.

refs*ű*

Available since 7.0.0 (Newton)

A list of resource IDs for the resources in the group.

refs_map*ű*

Available since 7.0.0 (Newton)

A map of resource names to IDs for the resources in the group.

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::AutoScalingGroup
    properties:
      cooldown: Integer
      desired_capacity: Integer
      max_size: Integer
      min_size: Integer
      resource: {...}
      rolling_updates: {"min_in_service": Integer, "max_batch_size": Integer,
↪"pause_time": Number}
```

OS::Heat::CloudConfig

Available since 2014.1 (Icehouse)

A configuration resource for representing cloud-init cloud-config.

This resource allows cloud-config YAML to be defined and stored by the config API. Any intrinsic functions called in the config will be resolved before storing the result.

This resource will generally be referenced by OS::Nova::Server user_data, or OS::Heat::MultipartMime parts config. Since cloud-config is boot-only configuration, any changes to the definition will result in the replacement of all servers which reference it.

Optional Properties

cloud_config*ű*

Map representing the cloud-config data structure which will be formatted as YAML.

Map value expected.

Updates cause replacement.

Attributes

config*ű*
The config value of the software config.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::CloudConfig
    properties:
      cloud_config: {...}
```

OS::Heat::Delay

Available since 11.0.0 (Rocky)

A resource that pauses for a configurable delay.

By manipulating the dependency relationships between resources in the template, a delay can be inserted at an arbitrary point during e.g. stack creation or deletion. They delay will occur after any resource that it depends on during CREATE or SUSPEND, and before any resource that it depends on during DELETE or RESUME. Similarly, it will occur before any resource that depends on it during CREATE or SUSPEND, and after any resource thet depends on it during DELETE or RESUME.

If a non-zero maximum jitter is specified, a random amount of jitter - chosen with uniform probability in the range from 0 to the product of the maximum jitter value and the jitter multiplier (1s by default) - is added to the minimum delay time. This can be used, for example, in the scaled unit of a large scaling group to prevent thundering herd issues.

Optional Properties

actions*ű*

Actions during which the delay will occur.

List value expected.

Can be updated without replacement.

Defaults to `["CREATE"]`

Allowed values: CREATE, DELETE, SUSPEND, RESUME

jitter_multiplier*ű*

Number of seconds to multiply the maximum jitter value by.

Number value expected.

Can be updated without replacement.

Defaults to `1.0`

The value must be at least 0.

max_jitter*ű*

Maximum jitter to add to the minimum wait time.

Number value expected.

Can be updated without replacement.

Defaults to `0`

The value must be at least 0.

min_wait*ű*

Minimum time in seconds to wait during the specified actions.

Number value expected.

Can be updated without replacement.

Defaults to `0`

The value must be at least 0.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
```

(continues on next page)

(continued from previous page)

```
    type: OS::Heat::Delay
    properties:
      actions: [Value, Value, ...]
      jitter_multiplier: Number
      max_jitter: Number
      min_wait: Number
```

OS::Heat::DeployedServer

A resource for managing servers that are already deployed.

A DeployedServer resource manages resources for servers that have been deployed externally from OpenStack. These servers can be associated with SoftwareDeployments for further orchestration via Heat.

Optional Properties

deployment_swift_data*ií*

Available since 9.0.0 (Pike)

Swift container and object to use for storing deployment data for the server resource. The parameter is a map value with the keys container and object, and the values are the corresponding container and object names. The software_config_transport parameter must be set to POLL_TEMP_URL for swift to be used. If not specified, and software_config_transport is set to POLL_TEMP_URL, a container will be automatically created from the resource name, and the object name will be a generated uuid.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

Map properties:

container*ií*

Optional.

Name of the container.

String value expected.

Can be updated without replacement.

The length must be at least 1.

object*ií*

Optional.

Name of the object.

String value expected.

Can be updated without replacement.

The length must be at least 1.

name*ű*

Server name.

String value expected.

Can be updated without replacement.

software_config_transport*ű*

How the server should receive the metadata required for software configuration. POLL_SERVER_CFN will allow calls to the cfn API action DescribeStackResource authenticated with the provided keypair. POLL_SERVER_HEAT will allow calls to the Heat API resource-show using the provided keystone credentials. POLL_TEMP_URL will create and populate a Swift TempURL with metadata for polling. ZAQAR_MESSAGE will create a dedicated zaqar queue and post the metadata for polling.

String value expected.

Can be updated without replacement.

Defaults to `"POLL_SERVER_CFN"`

Allowed values: POLL_SERVER_CFN, POLL_SERVER_HEAT, POLL_TEMP_URL, ZAQAR_MESSAGE

metadata*ű*

DEPRECATED since 9.0.0 (Pike) - This property will be ignored

Available since 8.0.0 (Ocata)

Arbitrary key/value metadata to store for this server. Both keys and values must be 255 characters or less. Non-string values will be serialized to JSON (and the serialized string must be 255 characters or less).

Map value expected.

Can be updated without replacement.

Attributes

name*ű*
Name of the server.

os_collect_config*ű*

Available since 9.0.0 (Pike)

The os-collect-config configuration for the servers local agent to be configured to connect to Heat to retrieve deployment data.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::DeployedServer
    properties:
      deployment_swift_data: {"container": String, "object": String}
      name: String
      software_config_transport: String
```

OS::Heat::InstanceGroup

An instance group that can scale arbitrary instances.

A resource allowing for the creating number of defined with AWS::AutoScaling::LaunchConfiguration instances. Allows to associate scaled resources with loadbalancer resources.

Required Properties

AvailabilityZones*ű*

Not Implemented.

List value expected.

Updates cause replacement.

LaunchConfigurationName*ű*

The reference to a LaunchConfiguration resource.

String value expected.

Can be updated without replacement.

Size*ű*

Desired number of instances.

Integer value expected.

Can be updated without replacement.

Optional Properties

LoadBalancerNames*ű*

List of LoadBalancer resources. Currently only the AWS::ElasticLoadBalancing::LoadBalancer resource type is supported.

List value expected.

Updates cause replacement.

Tags*ű*

Tags to attach to this group.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

Key*ű*

Required.

Tag key.

String value expected.

Updates cause replacement.

Value*ű*

Required.

Tag value.

String value expected.

Updates cause replacement.

Attributes

InstanceList*ű*
A comma-delimited list of server ip addresses. (Heat extension).

show*ű*
Detailed information about resource.

update_policy

RollingUpdate*ű*

Map value expected.

Updates cause replacement.

Map properties:

MaxBatchSize*ű*

Optional.

Integer value expected.

Updates cause replacement.

Defaults to 1

MinInstancesInService*ű*

Optional.

Integer value expected.

Updates cause replacement.

Defaults to `0`

PauseTime*ű*

Optional.

String value expected.

Updates cause replacement.

Defaults to `"PT0S"`

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::InstanceGroup
    properties:
      AvailabilityZones: [Value, Value, ...]
      LaunchConfigurationName: String
      LoadBalancerNames: [Value, Value, ...]
      Size: Integer
      Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
```

OS::Heat::MultipartMime

Available since 2014.1 (Icehouse)

Assembles a collection of software configurations as a multi-part mime.

Parts in the message can be populated with inline configuration or references to other config resources. If the referenced resource is itself a valid multi-part mime message, that will be broken into parts and those parts appended to this message.

The resulting multi-part mime message will be stored by the configs API and can be referenced in properties such as OS::Nova::Server user_data.

This resource is generally used to build a list of cloud-init configuration elements including scripts and cloud-config. Since cloud-init is boot-only configuration, any changes to the definition will result in the replacement of all servers which reference it.

Optional Properties

group*ű*

Available since 14.0.0 (Ussuri)

Namespace to group this multi-part configs by when delivered to a server. This may imply what configuration tool is going to perform the configuration.

String value expected.

Updates cause replacement.

Defaults to `"Heat::Ungrouped"`

parts*ű*

Parts belonging to this message.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

config*ű*

Required.

Content of part to attach, either inline or by referencing the ID of another software config resource.

String value expected.

Updates cause replacement.

filename*ű*

Optional.

Optional filename to associate with part.

String value expected.

Updates cause replacement.

subtype*ű*

Optional.

Optional subtype to specify with the type.

String value expected.

Updates cause replacement.

type*ű*

Optional.

Whether the part content is text or multipart.

String value expected.

Updates cause replacement.

Defaults to "text"

Allowed values: text, multipart

Attributes

config*ű*
The config value of the software config.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::MultipartMime
    properties:
      group: String
      parts: [{"config": String, "filename": String, "type": String, "subtype
↪": String}, {"config": String, "filename": String, "type": String, "subtype
↪": String}, ...]
```

OS::Heat::None

Available since 5.0.0 (Liberty)

Enables easily disabling certain resources via the resource_registry.

It does nothing, but can effectively stub out any other resource because it will accept any properties and return any attribute (as None). Note this resource always does nothing on update (e.g it is not replaced even if a change to the stubbed resource properties would cause replacement).

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
```

(continues on next page)

(continued from previous page)

```
the_resource:
  type: OS::Heat::None
```

OS::Heat::RandomString

Available since 2014.1 (Icehouse)

A resource which generates a random string.

This is useful for configuring passwords and secrets on services. Random string can be generated from specified character sequences, which means that all characters will be randomly chosen from specified sequences, or with some classes, e.g. letterdigits, which means that all character will be randomly chosen from union of ascii letters and digits. Output string will be randomly generated string with specified length (or with length of 32, if length property doesnt specified).

Optional Properties

character_classes*ű*

A list of character class and their constraints to generate the random string from.

List value expected.

Updates cause replacement.

Defaults to `[{"class": "lettersdigits", "min": 1}]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

class*ű*

Optional.

A character class and its corresponding min constraint to generate the random string from.

String value expected.

Updates cause replacement.

Defaults to `"lettersdigits"`

Allowed values: lettersdigits, letters, lowercase, uppercase, digits, hexdigits, octdigits

min*ű*

Optional.

The minimum number of characters from this character class that will be in the generated string.

Integer value expected.

Updates cause replacement.

Defaults to 1

The value must be in the range 1 to 512.

character_sequences*ú*

A list of character sequences and their constraints to generate the random string from.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

min*ú*

Optional.

The minimum number of characters from this sequence that will be in the generated string.

Integer value expected.

Updates cause replacement.

Defaults to 1

The value must be in the range 1 to 512.

sequence*ú*

Required.

A character sequence and its corresponding min constraint to generate the random string from.

String value expected.

Updates cause replacement.

length*ú*

Length of the string to generate.

Integer value expected.

Updates cause replacement.

Defaults to 32

The value must be in the range 1 to 512.

salt*ú*

Value which can be set or changed on stack update to trigger the resource for replacement with a new random string. The salt value itself is ignored by the random generator.

String value expected.

Updates cause replacement.

Attributes

show*ří*
Detailed information about resource.

value*ří*
The random string generated by this resource. This value is also available by referencing the resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::RandomString
    properties:
      character_classes: [{"class": String, "min": Integer}, {"class": String,
↪ "min": Integer}, ...]
      character_sequences: [{"sequence": String, "min": Integer}, {"sequence
↪": String, "min": Integer}, ...]
      length: Integer
      salt: String
```

OS::Heat::ResourceChain

Available since 6.0.0 (Mitaka)

Creates one or more resources with the same configuration.

The types of resources to be created are passed into the chain through the `resources` property. One resource will be created for each type listed. Each is passed the configuration specified under `resource_properties`.

The `concurrent` property controls if the resources will be created concurrently. If omitted or set to false, each resource will be treated as having a dependency on the resource before it in the list.

Required Properties

resources*ří*

The list of resource types to create. This list may contain type names or aliases defined in the resource registry. Specific template names are not supported.

List value expected.

Can be updated without replacement.

Optional Properties

concurrent*ű*

If true, the resources in the chain will be created concurrently. If false or omitted, each resource will be treated as having a dependency on the previous resource in the list.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

resource_properties*ű*

Properties to pass to each resource being created in the chain.

Map value expected.

Updates cause replacement.

Attributes

attributes*ű*
A map of resource names to the specified attribute of each individual resource.

refs*ű*
A list of resource IDs for the resources in the chain.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::ResourceChain
    properties:
      concurrent: Boolean
      resource_properties: {...}
      resources: [Value, Value, ...]
```

OS::Heat::ResourceGroup

Available since 2014.1 (Icehouse)

Creates one or more identically configured nested resources.

In addition to the *refs* attribute, this resource implements synthetic attributes that mirror those of the resources in the group. When getting an attribute from this resource, however, a list of attribute values for each resource in the group is returned. To get attribute values for a single resource in the group, synthetic attributes of the form *resource.{resource index}.{attribute name}* can be used. The resource

ID of a particular resource in the group can be obtained via the synthetic attribute *resource.{resource index}*. Note, that if you get attribute without *{resource index}*, e.g. *[resource, {attribute_name}]*, youll get a list of this attributes value for all resources in group.

While each resource in the group will be identically configured, this resource does allow for some index-based customization of the properties of the resources in the group. For example:

```
resources:
  my_indexed_group:
    type: OS::Heat::ResourceGroup
    properties:
      count: 3
      resource_def:
        type: OS::Nova::Server
        properties:
          # create a unique name for each server
          # using its index in the group
          name: my_server_%index%
          image: CentOS 6.5
          flavor: 4GB Performance
```

would result in a group of three servers having the same image and flavor, but names of *my_server_0*, *my_server_1*, and *my_server_2*. The variable used for substitution can be customized by using the *index_var* property.

Required Properties

resource_def*ű*

Resource definition for the resources in the group. The value of this property is the definition of a resource just as if it had been declared in the template itself.

Map value expected.

Can be updated without replacement.

Map properties:

metadata*ű*

Available since 5.0.0 (Liberty)

Supplied metadata for the resources in the group.

Map value expected.

Can be updated without replacement.

properties*ű*

Property values for the resources in the group.

Map value expected.

Can be updated without replacement.

type*ű*

Required.

The type of the resources in the group.

String value expected.

Can be updated without replacement.

Optional Properties

count*ű*

The number of resources to create.

Integer value expected.

Can be updated without replacement.

Defaults to 1

The value must be at least 0.

index_var*ű*

Available since 2014.2 (Juno)

A variable that this resource will use to replace with the current index of a given resource in the group. Can be used, for example, to customize the name property of grouped servers in order to differentiate them when listed with nova client.

String value expected.

Updates cause replacement.

Defaults to `"%index%"`

The length must be at least 3.

removal_policies*ű*

Available since 2015.1 (Kilo)

Policies for removal of resources on update.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Policy to be processed when doing an update which requires removal of specific resources.

Map value expected.

Can be updated without replacement.

Map properties:

resource_list*ű*

List of resources to be removed when doing an update which requires removal of specific resources. The resource may be specified several ways: (1) The resource name, as in the nested stack, (2) The resource reference returned from get_resource in a template, as available via the refs attribute. Note this is destructive on update when specified; even if the count is not being reduced, and once a resource name is removed, its name is never reused in subsequent updates.

List value expected.

Can be updated without replacement.

Defaults to `[]`

removal_policies_mode*ŕí*

Available since 10.0.0 (Queens)

How to handle changes to removal_policies on update. The default append mode appends to the internal list, update replaces it on update.

String value expected.

Can be updated without replacement.

Defaults to `"append"`

Allowed values: append, update

Attributes

attributes*ŕí*

Available since 2014.2 (Juno)

A map of resource names to the specified attribute of each individual resource. Requires heat_template_version: 2014-10-16.

refs*ŕí*
A list of resource IDs for the resources in the group.

refs_map*ŕí*

Available since 7.0.0 (Newton)

A map of resource names to IDs for the resources in the group.

removed_rsrc_list*ŕí*

Available since 7.0.0 (Newton)

A list of removed resource names.

show*ŕí*
Detailed information about resource.

update_policy

batch_create*ří*

Available since 5.0.0 (Liberty)

Map value expected.

Updates cause replacement.

Map properties:

max_batch_size*ří*

Optional.

The maximum number of resources to create at once.

Integer value expected.

Updates cause replacement.

Defaults to `1`

The value must be at least 1.

pause_time*ří*

Optional.

The number of seconds to wait between batches.

Number value expected.

Updates cause replacement.

Defaults to `0`

The value must be at least 0.

rolling_update*ří*

Available since 5.0.0 (Liberty)

Map value expected.

Updates cause replacement.

Map properties:

max_batch_size*ří*

Optional.

The maximum number of resources to replace at once.

Integer value expected.

Updates cause replacement.

Defaults to `1`

The value must be at least 1.

min_in_service*ří*

Optional.

The minimum number of resources in service while rolling updates are being executed.

Integer value expected.

Updates cause replacement.

Defaults to `0`

The value must be at least 0.

pause_time*ří*

Optional.

The number of seconds to wait between batches of updates.

Number value expected.

Updates cause replacement.

Defaults to `0`

The value must be at least 0.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::ResourceGroup
    properties:
      count: Integer
      index_var: String
      removal_policies: [{"resource_list": [Value, Value, ...]}, {"resource_
↪list": [Value, Value, ...]}, ...]
      removal_policies_mode: String
      resource_def: {"type": String, "properties": {...}, "metadata": {...}}
```

OS::Heat::ScalingPolicy

A resource to manage scaling of *OS::Heat::AutoScalingGroup*.

Note while it may incidentally support *AWS::AutoScaling::AutoScalingGroup* for now, please dont use it for that purpose and use *AWS::AutoScaling::ScalingPolicy* instead.

Resource to manage scaling for *OS::Heat::AutoScalingGroup*, i.e. define which metric should be scaled and scaling adjustment, set cooldown etc.

Required Properties

adjustment_type*ű*

Type of adjustment (absolute or percentage).

String value expected.

Can be updated without replacement.

Allowed values: change_in_capacity, exact_capacity, percent_change_in_capacity

auto_scaling_group_id*ű*

AutoScaling group ID to apply policy to.

String value expected.

Updates cause replacement.

scaling_adjustment*ű*

Size of adjustment.

Number value expected.

Can be updated without replacement.

Optional Properties

cooldown*ű*

Cooldown period, in seconds.

Number value expected.

Can be updated without replacement.

min_adjustment_step*ű*

Minimum number of resources that are added or removed when the AutoScaling group scales up or down. This can be used only when specifying percent_change_in_capacity for the adjustment_type property.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

Attributes

alarm_url*ű*
A signed url to handle the alarm.

show*ű*
Detailed information about resource.

signal_url*ű*

Available since 5.0.0 (Liberty)

A url to handle the alarm using native API.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::ScalingPolicy
    properties:
      adjustment_type: String
      auto_scaling_group_id: String
      cooldown: Number
      min_adjustment_step: Integer
      scaling_adjustment: Number
```

OS::Heat::SoftwareComponent

Available since 2014.2 (Juno)

A resource for describing and storing a software component.

This resource is similar to OS::Heat::SoftwareConfig. In contrast to SoftwareConfig which allows for storing only one configuration (e.g. one script), SoftwareComponent allows for storing multiple configurations to address handling of all lifecycle hooks (CREATE, UPDATE, SUSPEND, RESUME, DELETE) for a software component in one place.

This resource is backed by the persistence layer and the API of the SoftwareConfig resource, and only adds handling for the additional configs property and attribute.

Required Properties

configs*ű*

The list of configurations for the different lifecycle actions of the represented software component.

List value expected.

Updates cause replacement.

The length must be at least 1.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

actions*ű*

Lifecycle actions to which the configuration applies. The string values provided for this property can include the standard resource actions CREATE, DELETE, UPDATE, SUSPEND and RESUME supported by Heat.

List value expected.

Updates cause replacement.

Defaults to `["CREATE", "UPDATE"]`

The length must be at least 1.

List contents:

Optional.

String value expected.

Updates cause replacement.

config*ǘ*

Optional.

Configuration script or manifest which specifies what actual configuration is performed.

String value expected.

Updates cause replacement.

tool*ǘ*

Required.

The configuration tool used to actually apply the configuration on a server. This string property has to be understood by in-instance tools running inside deployed servers.

String value expected.

Updates cause replacement.

Optional Properties

inputs*ǘ*

Schema representing the inputs that this software config is expecting.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

default*ǘ*

Optional.

Default value for the input if none is specified.

Any value expected.

Updates cause replacement.

description*ǘ*

Optional.

Description of the input.

String value expected.

Updates cause replacement.

name*ǘ*

Required.

Name of the input.

String value expected.

Updates cause replacement.

replace_on_change*ǘ*

Optional.

Replace the deployment instead of updating it when the input value changes.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

type*ǘ*

Optional.

Type of the value of the input.

String value expected.

Updates cause replacement.

Defaults to `"String"`

Allowed values: String, Number, CommaDelimitedList, Json, Boolean

options*ǘ*

Map containing options specific to the configuration management tool used by this resource.

Map value expected.

Updates cause replacement.

outputs*ǘ*

Schema representing the outputs that this software config will produce.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ű*

Optional.

Description of the output.

String value expected.

Updates cause replacement.

error_output*ű*

Optional.

Denotes that the deployment is in an error state if this output has a value.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

name*ű*

Required.

Name of the output.

String value expected.

Updates cause replacement.

type*ű*

Optional.

Type of the value of the output.

String value expected.

Updates cause replacement.

Defaults to `"String"`

Allowed values: String, Number, CommaDelimitedList, Json, Boolean

Attributes

config*ű*
The config value of the software config.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
```

(continues on next page)

(continued from previous page)

```
...
the_resource:
  type: OS::Heat::SoftwareComponent
  properties:
    configs: [{"actions": [String, String, ...], "config": String, "tool":␣
↪String}, {"actions": [String, String, ...], "config": String, "tool":␣
↪String}, ...]
    inputs: [{"name": String, "description": String, "type": String,
↪"default": Any, "replace_on_change": Boolean}, {"name": String, "description
↪": String, "type": String, "default": Any, "replace_on_change": Boolean}, ..
↪.]
    options: {...}
    outputs: [{"name": String, "description": String, "type": String,
↪"error_output": Boolean}, {"name": String, "description": String, "type":␣
↪String, "error_output": Boolean}, ...]
```

OS::Heat::SoftwareConfig

Available since 2014.1 (Icehouse)

A resource for describing and storing software configuration.

The software_configs API which backs this resource creates immutable configs, so any change to the template resource definition will result in a new config being created, and the old one being deleted.

Configs can be defined in the same template which uses them, or they can be created in one stack, and passed to another stack via a parameter.

A config resource can be referenced in other resource properties which are config-aware. This includes the properties OS::Nova::Server user_data, OS::Heat::SoftwareDeployment config and OS::Heat::MultipartMime parts config.

Along with the config script itself, this resource can define schemas for inputs and outputs which the config script is expected to consume and produce. Inputs and outputs are optional and will map to concepts which are specific to the configuration tool being used.

Optional Properties

config*ű*

Configuration script or manifest which specifies what actual configuration is performed.

String value expected.

Updates cause replacement.

group*ű*

Namespace to group this software config by when delivered to a server. This may imply what configuration tool is going to perform the configuration.

String value expected.

Updates cause replacement.

Defaults to `"Heat::Ungrouped"`

inputs*ǘ*

Schema representing the inputs that this software config is expecting.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

default*ǘ*

Optional.

Default value for the input if none is specified.

Any value expected.

Updates cause replacement.

description*ǘ*

Optional.

Description of the input.

String value expected.

Updates cause replacement.

name*ǘ*

Required.

Name of the input.

String value expected.

Updates cause replacement.

replace_on_change*ǘ*

Optional.

Replace the deployment instead of updating it when the input value changes.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

type*ǘ*

Optional.

Type of the value of the input.

String value expected.

Updates cause replacement.

Defaults to `"String"`

Allowed values: String, Number, CommaDelimitedList, Json, Boolean

options*ŕí*

Map containing options specific to the configuration management tool used by this resource.

Map value expected.

Updates cause replacement.

outputs*ŕí*

Schema representing the outputs that this software config will produce.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ŕí*

Optional.

Description of the output.

String value expected.

Updates cause replacement.

error_output*ŕí*

Optional.

Denotes that the deployment is in an error state if this output has a value.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

name*ŕí*

Required.

Name of the output.

String value expected.

Updates cause replacement.

type*ŕí*

Optional.

Type of the value of the output.

String value expected.

Updates cause replacement.

Defaults to "String"

Allowed values: String, Number, CommaDelimitedList, Json, Boolean

Attributes

config*ú*
The config value of the software config.

show*ú*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::SoftwareConfig
    properties:
      config: String
      group: String
      inputs: [{"name": String, "description": String, "type": String,
↪"default": Any, "replace_on_change": Boolean}, {"name": String, "description
↪": String, "type": String, "default": Any, "replace_on_change": Boolean}, ..
↪.]
      options: {...}
      outputs: [{"name": String, "description": String, "type": String,
↪"error_output": Boolean}, {"name": String, "description": String, "type":␣
↪String, "error_output": Boolean}, ...]
```

OS::Heat::SoftwareDeployment

Available since 2014.1 (Icehouse)

This resource associates a server with some configuration.

The configuration is to be deployed to that server.

A deployment allows input values to be specified which map to the inputs schema defined in the config resource. These input values are interpreted by the configuration tool in a tool-specific manner.

Whenever this resource goes to an IN_PROGRESS state, it creates an ephemeral config that includes the inputs values plus a number of extra inputs which have names prefixed with `deploy_`. The extra inputs relate to the current state of the stack, along with the information and credentials required to signal back the deployment results.

Unless signal_transport=NO_SIGNAL, this resource will remain in an IN_PROGRESS state until the server signals it with the output values for that deployment. Those output values are then available as resource attributes, along with the default attributes deploy_stdout, deploy_stderr and deploy_status_code.

Specifying actions other than the default CREATE and UPDATE will result in the deployment being triggered in those actions. For example this would allow cleanup configuration to be performed during actions SUSPEND and DELETE. A config could be designed to only work with some specific actions, or a config can read the value of the deploy_action input to allow conditional logic to perform different configuration for different actions.

Required Properties

server*ű*

ID of resource to apply configuration to. Normally this should be a Nova server ID.

String value expected.

Updates cause replacement.

Optional Properties

actions*ű*

Which lifecycle actions of the deployment resource will result in this deployment being triggered.

List value expected.

Can be updated without replacement.

Defaults to `["CREATE", "UPDATE"]`

Allowed values: CREATE, UPDATE, DELETE, SUSPEND, RESUME

config*ű*

ID of software configuration resource to execute when applying to the server.

String value expected.

Can be updated without replacement.

input_values*ű*

Input values to apply to the software configuration on this server.

Map value expected.

Can be updated without replacement.

name*ű*

Name of the derived config associated with this deployment. This is used to apply a sort order to the list of configurations currently deployed to a server.

String value expected.

Can be updated without replacement.

signal_transport*ű*

How the server should signal to heat with the deployment output values. CFN_SIGNAL will allow an HTTP POST to a CFN keypair signed URL. TEMP_URL_SIGNAL will create a Swift TempURL to be signaled via HTTP PUT. HEAT_SIGNAL will allow calls to the Heat API resource-signal using the provided keystone credentials. ZAQAR_SIGNAL will create a dedicated zaqar queue to be signaled using the provided keystone credentials. NO_SIGNAL will result in the resource going to the COMPLETE state without waiting for any signal.

String value expected.

Updates cause replacement.

Defaults to `"CFN_SIGNAL"`

Allowed values: CFN_SIGNAL, TEMP_URL_SIGNAL, HEAT_SIGNAL, NO_SIGNAL, ZAQAR_SIGNAL

Attributes

deploy_status_code*ű*
Returned status code from the configuration execution.

deploy_stderr*ű*
Captured stderr from the configuration execution.

deploy_stdout*ű*
Captured stdout from the configuration execution.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::SoftwareDeployment
    properties:
      actions: [Value, Value, ...]
      config: String
      input_values: {...}
      name: String
      server: String
      signal_transport: String
```

OS::Heat::SoftwareDeploymentGroup

Available since 5.0.0 (Liberty)

This resource associates a group of servers with some configuration.

The configuration is to be deployed to all servers in the group.

The properties work in a similar way to OS::Heat::SoftwareDeployment, and in addition to the attributes documented, you may pass any attribute supported by OS::Heat::SoftwareDeployment, including those exposing arbitrary outputs, and return a map of deployment names to the specified attribute.

Required Properties

servers*ű*

A map of names and server IDs to apply configuration to. The name is arbitrary and is used as the Heat resource name for the corresponding deployment.

Map value expected.

Can be updated without replacement.

Optional Properties

actions*ű*

Which lifecycle actions of the deployment resource will result in this deployment being triggered.

List value expected.

Can be updated without replacement.

Defaults to `["CREATE", "UPDATE"]`

Allowed values: CREATE, UPDATE, DELETE, SUSPEND, RESUME

config*ű*

ID of software configuration resource to execute when applying to the server.

String value expected.

Can be updated without replacement.

input_values*ű*

Input values to apply to the software configuration on this server.

Map value expected.

Can be updated without replacement.

name*ű*

Name of the derived config associated with this deployment. This is used to apply a sort order to the list of configurations currently deployed to a server.

String value expected.

Can be updated without replacement.

signal_transport*ű*

How the server should signal to heat with the deployment output values. CFN_SIGNAL will allow an HTTP POST to a CFN keypair signed URL. TEMP_URL_SIGNAL will create a Swift TempURL to be signaled via HTTP PUT. HEAT_SIGNAL will allow calls to the Heat API resource-signal using the provided keystone credentials. ZAQAR_SIGNAL will create a dedicated zaqar queue to be signaled using the provided keystone credentials. NO_SIGNAL will result in the resource going to the COMPLETE state without waiting for any signal.

String value expected.

Updates cause replacement.

Defaults to `"CFN_SIGNAL"`

Allowed values: CFN_SIGNAL, TEMP_URL_SIGNAL, HEAT_SIGNAL, NO_SIGNAL, ZAQAR_SIGNAL

Attributes

deploy_status_codes*ú*
A map of Nova names and returned status code from the configuration execution.

deploy_stderrs*ú*
A map of Nova names and captured stderrs from the configuration execution to each server.

deploy_stdouts*ú*
A map of Nova names and captured stdouts from the configuration execution to each server.

show*ú*
Detailed information about resource.

update_policy

batch_create*ú*

Available since 7.0.0 (Newton)

Map value expected.

Updates cause replacement.

Map properties:

max_batch_size*ú*

Optional.

The maximum number of resources to create at once.

Integer value expected.

Updates cause replacement.

Defaults to `1`

The value must be at least 1.

pause_time*ú*

Optional.

The number of seconds to wait between batches.

Number value expected.

Updates cause replacement.

Defaults to `0`

The value must be at least 0.

rolling_update*ű*

Available since 7.0.0 (Newton)

Map value expected.

Updates cause replacement.

Map properties:

max_batch_size*ű*

Optional.

The maximum number of deployments to replace at once.

Integer value expected.

Updates cause replacement.

Defaults to `1`

The value must be at least 1.

pause_time*ű*

Optional.

The number of seconds to wait between batches of updates.

Number value expected.

Updates cause replacement.

Defaults to `0`

The value must be at least 0.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::SoftwareDeploymentGroup
    properties:
      actions: [Value, Value, ...]
      config: String
      input_values: {...}
      name: String
      servers: {...}
      signal_transport: String
```

OS::Heat::Stack

A Resource representing a stack.

A resource that allowing for the creating stack, where should be defined stack template in HOT format, parameters (if template has any parameters with no default value), and timeout of creating. After creating current stack will have remote stack.

Required Properties

template*ű*

Template that specifies the stack to be created as a resource.

String value expected.

Can be updated without replacement.

Optional Properties

context*ű*

Context for this stack.

Map value expected.

Can be updated without replacement.

Map properties:

ca_cert*ű*

Available since 12.0.0 (Stein)

Optional.

CA Cert for SSL.

String value expected.

Can be updated without replacement.

credential_secret_id*ű*

Available since 12.0.0 (Stein)

Optional.

A Barbican secret ID. The Barbican secret should contain an OpenStack credential that can be used to access a remote cloud.

String value expected.

Can be updated without replacement.

insecure*ű*

Available since 12.0.0 (Stein)

Optional.

If set, then the servers certificate will not be verified.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

region_name*ű*

Optional.

Region name in which this stack will be created.

String value expected.

Can be updated without replacement.

parameters*ű*

Set of parameters passed to this stack.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

timeout*ű*

Number of minutes to wait for this stack creation.

Integer value expected.

Can be updated without replacement.

Attributes

outputs*ű*
A dict of key-value pairs output from the stack.

show*ű*
Detailed information about resource.

stack_name*ű*
Name of the stack.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::Stack
    properties:
```

(continues on next page)

(continued from previous page)

```
    context: {"region_name": String, "credential_secret_id": String, "ca_
↪cert": String, "insecure": Boolean}
    parameters: {...}
    template: String
    timeout: Integer
```

OS::Heat::StructuredConfig

Available since 2014.1 (Icehouse)

A resource which has same logic with OS::Heat::SoftwareConfig.

This resource is like OS::Heat::SoftwareConfig except that the config property is represented by a Map rather than a String.

This is useful for configuration tools which use YAML or JSON as their configuration syntax. The resulting configuration is transferred, stored and returned by the software_configs API as parsed JSON.

Optional Properties

config*ű*

Map representing the configuration data structure which will be serialized to JSON format.

Map value expected.

Updates cause replacement.

group*ű*

Namespace to group this software config by when delivered to a server. This may imply what configuration tool is going to perform the configuration.

String value expected.

Updates cause replacement.

Defaults to `"Heat::Ungrouped"`

inputs*ű*

Schema representing the inputs that this software config is expecting.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

default*ű*

Optional.

Default value for the input if none is specified.

Any value expected.

Updates cause replacement.

description*ŕ*

Optional.

Description of the input.

String value expected.

Updates cause replacement.

name*ŕ*

Required.

Name of the input.

String value expected.

Updates cause replacement.

replace_on_change*ŕ*

Optional.

Replace the deployment instead of updating it when the input value changes.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

type*ŕ*

Optional.

Type of the value of the input.

String value expected.

Updates cause replacement.

Defaults to `"String"`

Allowed values: String, Number, CommaDelimitedList, Json, Boolean

options*ŕ*

Map containing options specific to the configuration management tool used by this resource.

Map value expected.

Updates cause replacement.

outputs*ŕ*

Schema representing the outputs that this software config will produce.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ǐ*

Optional.

Description of the output.

String value expected.

Updates cause replacement.

error_output*ǐ*

Optional.

Denotes that the deployment is in an error state if this output has a value.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

name*ǐ*

Required.

Name of the output.

String value expected.

Updates cause replacement.

type*ǐ*

Optional.

Type of the value of the output.

String value expected.

Updates cause replacement.

Defaults to `"String"`

Allowed values: String, Number, CommaDelimitedList, Json, Boolean

Attributes

config*ǐ*

The config value of the software config.

show*ǐ*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::StructuredConfig
    properties:
      config: {...}
      group: String
      inputs: [{"name": String, "description": String, "type": String,
↪"default": Any, "replace_on_change": Boolean}, {"name": String, "description
↪": String, "type": String, "default": Any, "replace_on_change": Boolean}, ..
↪.]
      options: {...}
      outputs: [{"name": String, "description": String, "type": String,
↪"error_output": Boolean}, {"name": String, "description": String, "type":␣
↪String, "error_output": Boolean}, ...]
```

OS::Heat::StructuredDeployment

Available since 2014.1 (Icehouse)

A resource which has same logic with OS::Heat::SoftwareDeployment.

A deployment resource like OS::Heat::SoftwareDeployment, but which performs input value substitution on the config defined by a OS::Heat::StructuredConfig resource.

Some configuration tools have no concept of inputs, so the input value substitution needs to occur in the deployment resource. An example of this is the JSON metadata consumed by the cfn-init tool.

Where the config contains {get_input: input_name} this will be substituted with the value of input_name in this resources input_values. If get_input needs to be passed through to the substituted configuration then a different input_key property value can be specified.

Required Properties

server*ű*

ID of resource to apply configuration to. Normally this should be a Nova server ID.

String value expected.

Updates cause replacement.

Optional Properties

actions*ű*

Which lifecycle actions of the deployment resource will result in this deployment being triggered.

List value expected.

Can be updated without replacement.

Defaults to `["CREATE", "UPDATE"]`

Allowed values: CREATE, UPDATE, DELETE, SUSPEND, RESUME

config*ű*

ID of software configuration resource to execute when applying to the server.

String value expected.

Can be updated without replacement.

input_key*ű*

Name of key to use for substituting inputs during deployment.

String value expected.

Updates cause replacement.

Defaults to `"get_input"`

input_values*ű*

Input values to apply to the software configuration on this server.

Map value expected.

Can be updated without replacement.

input_values_validate*ű*

Perform a check on the input values passed to verify that each required input has a corresponding value. When the property is set to STRICT and no value is passed, an exception is raised.

String value expected.

Updates cause replacement.

Defaults to `"LAX"`

Allowed values: LAX, STRICT

name*ű*

Name of the derived config associated with this deployment. This is used to apply a sort order to the list of configurations currently deployed to a server.

String value expected.

Can be updated without replacement.

signal_transport*ű*

How the server should signal to heat with the deployment output values. CFN_SIGNAL will allow an HTTP POST to a CFN keypair signed URL. TEMP_URL_SIGNAL will create a Swift TempURL to be signaled via HTTP PUT. HEAT_SIGNAL will allow calls to the Heat API resource-signal using the provided keystone credentials. ZAQAR_SIGNAL will create a dedicated zaqar queue to be signaled using the provided keystone credentials. NO_SIGNAL will result in the resource going to the COMPLETE state without waiting for any signal.

String value expected.

Updates cause replacement.

Defaults to "CFN_SIGNAL"

Allowed values: CFN_SIGNAL, TEMP_URL_SIGNAL, HEAT_SIGNAL, NO_SIGNAL, ZAQAR_SIGNAL

Attributes

deploy_status_code*ű*
Returned status code from the configuration execution.

deploy_stderr*ű*
Captured stderr from the configuration execution.

deploy_stdout*ű*
Captured stdout from the configuration execution.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::StructuredDeployment
    properties:
      actions: [Value, Value, ...]
      config: String
      input_key: String
      input_values: {...}
      input_values_validate: String
      name: String
      server: String
      signal_transport: String
```

OS::Heat::StructuredDeploymentGroup

Available since 5.0.0 (Liberty)

This resource associates a group of servers with some configuration.

This resource works similar as OS::Heat::SoftwareDeploymentGroup, but for structured resources.

Required Properties

servers*ű*

A map of names and server IDs to apply configuration to. The name is arbitrary and is used as the Heat resource name for the corresponding deployment.

Map value expected.

Can be updated without replacement.

Optional Properties

actions*ű*

Which lifecycle actions of the deployment resource will result in this deployment being triggered.

List value expected.

Can be updated without replacement.

Defaults to `["CREATE", "UPDATE"]`

Allowed values: CREATE, UPDATE, DELETE, SUSPEND, RESUME

config*ű*

ID of software configuration resource to execute when applying to the server.

String value expected.

Can be updated without replacement.

input_key*ű*

Name of key to use for substituting inputs during deployment.

String value expected.

Updates cause replacement.

Defaults to `"get_input"`

input_values*ű*

Input values to apply to the software configuration on this server.

Map value expected.

Can be updated without replacement.

input_values_validate*ű*

Perform a check on the input values passed to verify that each required input has a corresponding value. When the property is set to STRICT and no value is passed, an exception is raised.

String value expected.

Updates cause replacement.

Defaults to `"LAX"`

Allowed values: LAX, STRICT

name*ű*

Name of the derived config associated with this deployment. This is used to apply a sort order to the list of configurations currently deployed to a server.

String value expected.

Can be updated without replacement.

signal_transport*ű*

How the server should signal to heat with the deployment output values. CFN_SIGNAL will allow an HTTP POST to a CFN keypair signed URL. TEMP_URL_SIGNAL will create a Swift TempURL to be signaled via HTTP PUT. HEAT_SIGNAL will allow calls to the Heat API resource-signal using the provided keystone credentials. ZAQAR_SIGNAL will create a dedicated zaqar queue to be signaled using the provided keystone credentials. NO_SIGNAL will result in the resource going to the COMPLETE state without waiting for any signal.

String value expected.

Updates cause replacement.

Defaults to `"CFN_SIGNAL"`

Allowed values: CFN_SIGNAL, TEMP_URL_SIGNAL, HEAT_SIGNAL, NO_SIGNAL, ZAQAR_SIGNAL

Attributes

deploy_status_codes*ű*
A map of Nova names and returned status code from the configuration execution.

deploy_stderrs*ű*
A map of Nova names and captured stderrs from the configuration execution to each server.

deploy_stdouts*ű*
A map of Nova names and captured stdouts from the configuration execution to each server.

show*ű*
Detailed information about resource.

update_policy

batch_create*ű*

Available since 7.0.0 (Newton)

Map value expected.

Updates cause replacement.

Map properties:

max_batch_size*ű*

Optional.

The maximum number of resources to create at once.

Integer value expected.

Updates cause replacement.

Defaults to 1

The value must be at least 1.

pause_time*ű*

Optional.

The number of seconds to wait between batches.

Number value expected.

Updates cause replacement.

Defaults to `0`

The value must be at least 0.

rolling_update*ří*

Available since 7.0.0 (Newton)

Map value expected.

Updates cause replacement.

Map properties:

max_batch_size*ří*

Optional.

The maximum number of deployments to replace at once.

Integer value expected.

Updates cause replacement.

Defaults to `1`

The value must be at least 1.

pause_time*ří*

Optional.

The number of seconds to wait between batches of updates.

Number value expected.

Updates cause replacement.

Defaults to `0`

The value must be at least 0.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::StructuredDeploymentGroup
    properties:
      actions: [Value, Value, ...]
      config: String
      input_key: String
```

(continues on next page)

(continued from previous page)

```
    input_values: {...}
    input_values_validate: String
    name: String
    servers: {...}
    signal_transport: String
```

OS::Heat::SwiftSignal

Available since 2014.2 (Juno)

Resource for handling signals received by SwiftSignalHandle.

This resource handles signals received by SwiftSignalHandle and is same as WaitCondition resource.

Required Properties

handle*ű*

URL of TempURL where resource will signal completion and optionally upload data.

String value expected.

Updates cause replacement.

timeout*ű*

The maximum number of seconds to wait for the resource to signal completion. Once the timeout is reached, creation of the signal resource will fail.

Number value expected.

Updates cause replacement.

The value must be in the range 1 to 43200.

Optional Properties

count*ű*

The number of success signals that must be received before the stack creation process continues.

Integer value expected.

Updates cause replacement.

Defaults to 1

The value must be in the range 1 to 1000.

Attributes

data*ű*
JSON data that was uploaded via the SwiftSignalHandle.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::SwiftSignal
    properties:
      count: Integer
      handle: String
      timeout: Number
```

OS::Heat::SwiftSignalHandle

Available since 2014.2 (Juno)

Resource for managing signals from Swift resources.

This resource is same as WaitConditionHandle, but designed for using by Swift resources.

Attributes

curl_cli*ű*
: Convenience attribute, provides curl CLI command prefix, which can be used for signalling handle completion or failure. You can signal success by adding data-binary {status: SUCCESS} , or signal failure by adding data-binary {status: FAILURE}.

endpoint*ű*
: Endpoint/url which can be used for signalling handle.

show*ű*
: Detailed information about resource.

token*ű*
: Tokens are not needed for Swift TempURLs. This attribute is being kept for compatibility with the OS::Heat::WaitConditionHandle resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::SwiftSignalHandle
```

OS::Heat::TestResource

Available since 5.0.0 (Liberty)

A resource which stores the string value that was provided.

This resource is to be used only for testing. It has control knobs such as update_replace, fail, wait_secs.

Optional Properties

action_wait_secs*ií*

Options for simulating waiting.

Map value expected.

Can be updated without replacement.

Map properties:

create*ií*

Optional.

Seconds to wait after a create. Defaults to the global wait_secs.

Number value expected.

Can be updated without replacement.

delete*ií*

Optional.

Seconds to wait after a delete. Defaults to the global wait_secs.

Number value expected.

Can be updated without replacement.

update*ií*

Optional.

Seconds to wait after an update. Defaults to the global wait_secs.

Number value expected.

Can be updated without replacement.

attr_wait_secs*ií*

Available since 6.0.0 (Mitaka)

Number value for timeout during resolving output value.

Number value expected.

Can be updated without replacement.

Defaults to `0`

client_name*ří*

Client to poll.

String value expected.

Can be updated without replacement.

Defaults to `""`

constraint_prop_secs*ří*

Available since 6.0.0 (Mitaka)

Number value for delay during resolve constraint.

Number value expected.

Can be updated without replacement.

Defaults to `0`

Value must be of type test_constr

entity_name*ří*

Client entity to poll.

String value expected.

Can be updated without replacement.

Defaults to `""`

fail*ří*

Value which can be set to fail the resource operation to test failure scenarios.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

update_replace*ří*

Value which can be set to trigger update replace for the particular resource.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

update_replace_value*ří*

Available since 7.0.0 (Newton)

Some value that can be stored but can not be updated.

String value expected.

Updates cause replacement.

value*ű*

The input string to be stored.

String value expected.

Can be updated without replacement.

Defaults to `"test_string"`

wait_secs*ű*

Seconds to wait after an action (-1 is infinite).

Number value expected.

Can be updated without replacement.

Defaults to `0`

Attributes

output*ű*
The string that was stored. This value is also available by referencing the resource.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::TestResource
    properties:
      action_wait_secs: {"create": Number, "update": Number, "delete": Number}
      attr_wait_secs: Number
      client_name: String
      constraint_prop_secs: Number
      entity_name: String
      fail: Boolean
      update_replace: Boolean
      update_replace_value: String
      value: String
      wait_secs: Number
```

OS::Heat::UpdateWaitConditionHandle

Available since 2014.1 (Icehouse)

WaitConditionHandle that clears signals and changes handle on update.

This works similarly to an AWS::CloudFormation::WaitConditionHandle, except that on update it clears all signals received and changes the handle. Using this handle means that you must setup the signal senders to send their signals again any time the update handle changes. This allows us to roll out new configurations and be confident that they are rolled out once UPDATE COMPLETE is reached.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::UpdateWaitConditionHandle
```

OS::Heat::Value

Available since 7.0.0 (Newton)

A resource which exposes its value property as an attribute.

This is useful for exposing a value that is a simple manipulation of other template parameters and/or other resources.

Required Properties

value*ű*

The expression to generate the value attribute.

Any value expected.

Can be updated without replacement.

Optional Properties

type*ű*

The type of the value property.

String value expected.

Can be updated without replacement.

Allowed values: string, number, comma_delimited_list, json, boolean

Attributes

show*ű*
Detailed information about resource.

value*ű*
The value generated by this resources properties value expression, with type determined from the properties type.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::Value
    properties:
      type: String
      value: Any
```

OS::Heat::WaitCondition

Available since 2014.2 (Juno)

Resource for handling signals received by WaitConditionHandle.

Resource takes WaitConditionHandle and starts to create. Resource is in CREATE_IN_PROGRESS status until WaitConditionHandle doesnt receive sufficient number of successful signals (this number can be specified with count property) and successfully creates after that, or fails due to timeout.

Required Properties

handle*ű*

A reference to the wait condition handle used to signal this wait condition.

String value expected.

Updates cause replacement.

timeout*ű*

The number of seconds to wait for the correct number of signals to arrive.

Number value expected.

Updates cause replacement.

The value must be in the range 1 to 43200.

Optional Properties

count*ű*

The number of success signals that must be received before the stack creation process continues.

Integer value expected.

Can be updated without replacement.

Defaults to 1

The value must be at least 1.

Attributes

data*ű*
JSON string containing data associated with wait condition signals sent to the handle.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::WaitCondition
    properties:
      count: Integer
      handle: String
      timeout: Number
```

OS::Heat::WaitConditionHandle

Available since 2014.2 (Juno)

Resource for managing instance signals.

The main points of this resource are:

- have no dependencies (so the instance can reference it).
- create credentials to allow for signalling from the instance.
- handle signals from the instance, validate and store result.

Optional Properties

signal_transport*ű*

Available since 6.0.0 (Mitaka)

How the client will signal the wait condition. CFN_SIGNAL will allow an HTTP POST to a CFN keypair signed URL. TEMP_URL_SIGNAL will create a Swift TempURL to be signalled via HTTP PUT. HEAT_SIGNAL will allow calls to the Heat API resource-signal using the provided keystone credentials. ZAQAR_SIGNAL will create a dedicated zaqar queue to be signalled using the provided keystone credentials. TOKEN_SIGNAL will allow and HTTP POST to a Heat API endpoint with the provided keystone token. NO_SIGNAL will result in the resource going to a signalled state without waiting for any signal.

String value expected.

Updates cause replacement.

Defaults to `"TOKEN_SIGNAL"`

Allowed values: CFN_SIGNAL, TEMP_URL_SIGNAL, HEAT_SIGNAL, NO_SIGNAL, ZAQAR_SIGNAL, TOKEN_SIGNAL

Attributes

curl_cli*ú*
Convenience attribute, provides curl CLI command prefix, which can be used for signalling handle completion or failure when signal_transport is set to TOKEN_SIGNAL. You can signal success by adding data-binary {status: SUCCESS} , or signal failure by adding data-binary {status: FAILURE}. This attribute is set to None for all other signal transports.

endpoint*ú*
Endpoint/url which can be used for signalling handle when signal_transport is set to TOKEN_SIGNAL. None for all other signal transports.

show*ú*
Detailed information about resource.

signal*ú*
JSON serialized map that includes the endpoint, token and/or other attributes the client must use for signalling this handle. The contents of this map depend on the type of signal selected in the signal_transport property.

token*ú*
Token for stack-user which can be used for signalling handle when signal_transport is set to TOKEN_SIGNAL. None for all other signal transports.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Heat::WaitConditionHandle
    properties:
      signal_transport: String
```

OS::Ironic::Port

Available since 13.0.0 (Train)

A resource that creates a ironic port.

Node UUID and physical hardware address for the Port (MAC address in most cases) are needed (all Ports must be associated to a Node when created).

Required Properties

address*ú*

Physical hardware address of this network Port, typically the hardware MAC address.

String value expected.

Can be updated without replacement.

node*ú*

UUID or name of the Node this resource belongs to.

String value expected.

Can be updated without replacement.

Value must be of type ironic.node

Optional Properties

extra*ú*

A set of one or more arbitrary metadata key and value pairs.

Map value expected.

Can be updated without replacement.

is_smartnic*ú*

Indicates whether the Port is a Smart NIC port.

Boolean value expected.

Can be updated without replacement.

local_link_connection*ú*

The Port binding profile. If specified, must contain switch_id (only a MAC address or an OpenFlow based datapath_id of the switch are accepted in this field) and port_id (identifier of the physical port on the switch to which nodes port is connected to) fields. switch_info is an optional string field to be used to store any vendor-specific information.

Map value expected.

Can be updated without replacement.

physical_network*ú*

The name of the physical network to which a port is connected. May be empty.

String value expected.

Can be updated without replacement.

portgroup*ú*

UUID or name of the Portgroup this resource belongs to.

String value expected.

Can be updated without replacement.

Value must be of type ironic.portgroup

pxe_enabled*ú*

Indicates whether PXE is enabled or disabled on the Port.

Boolean value expected.

Can be updated without replacement.

Attributes

address*ú*
Physical hardware address of this network Port, typically the hardware MAC address.

extra*ú*
A set of one or more arbitrary metadata key and value pairs.

internal_info*ú*
Internal metadata set and stored by the Port. This field is read-only.

is_smartnic*ú*
Indicates whether the Port is a Smart NIC port.

local_link_connection*ú*
The Port binding profile. If specified, must contain switch_id (only a MAC address or an OpenFlow based datapath_id of the switch are accepted in this field) and port_id (identifier of the physical port on the switch to which nodes port is connected to) fields. switch_info is an optional string field to be used to store any vendor-specific information.

node_uuid*ú*
UUID of the Node this resource belongs to.

physical_network*ú*
The name of the physical network to which a port is connected. May be empty.

portgroup_uuid*ú*
UUID of the Portgroup this resource belongs to.

pxe_enabled*ú*
Indicates whether PXE is enabled or disabled on the Port.

show*ú*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Ironic::Port
    properties:
      address: String
      extra: {...}
      is_smartnic: Boolean
      local_link_connection: {...}
      node: String
      physical_network: String
      portgroup: String
      pxe_enabled: Boolean
```

OS::Keystone::Domain

Available since 8.0.0 (Ocata) - Supported versions: keystone v3

Heat Template Resource for Keystone Domain.

This plug-in helps to create, update and delete a keystone domain. Also it can be used for enable or disable a given keystone domain.

Optional Properties

description*ű*

Description of keystone domain.

String value expected.

Can be updated without replacement.

enabled*ű*

This domain is enabled or disabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

name*ű*

The name of the domain.

String value expected.

Can be updated without replacement.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::Domain
    properties:
      description: String
      enabled: Boolean
      name: String
```

OS::Keystone::Endpoint

Available since 5.0.0 (Liberty) - Supported versions: keystone v3

Heat Template Resource for Keystone Service Endpoint.

Keystone endpoint is just the URL that can be used for accessing a service within OpenStack. Endpoint can be accessed by admin, by services or public, i.e. everyone can use this endpoint.

Required Properties

interface*ű*

Interface type of keystone service endpoint.

String value expected.

Can be updated without replacement.

Allowed values: public, internal, admin

service*ű*

Name or Id of keystone service.

String value expected.

Can be updated without replacement.

Value must be of type keystone.service

url*ű*

URL of keystone service endpoint.

String value expected.

Can be updated without replacement.

Optional Properties

enabled*ű*

Available since 6.0.0 (Mitaka)

This endpoint is enabled or disabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

name*ű*

Name of keystone endpoint.

String value expected.

Can be updated without replacement.

region*ű*

Name or Id of keystone region.

String value expected.

Can be updated without replacement.

Value must be of type keystone.region

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::Endpoint
    properties:
      enabled: Boolean
      interface: String
      name: String
      region: String
      service: String
      url: String
```

OS::Keystone::Group

Available since 2015.1 (Kilo) - Supported versions: keystone v3

Heat Template Resource for Keystone Group.

Groups are a container representing a collection of users. A group itself must be owned by a specific domain, and hence all group names are not globally unique, but only unique to their domain.

Optional Properties

description*ű*

Description of keystone group.

String value expected.

Can be updated without replacement.

Defaults to `""`

domain*ű*

Name or id of keystone domain.

String value expected.

Can be updated without replacement.

Defaults to `"default"`

Value must be of type keystone.domain

name*ű*

Name of keystone group.

String value expected.

Can be updated without replacement.

roles*ű*

List of role assignments.

List value expected.

Can be updated without replacement.

List contents:

Map between role with either project or domain.

Map value expected.

Can be updated without replacement.

Map properties:

domain*ű*

Optional.

Keystone domain.

String value expected.

Can be updated without replacement.

Value must be of type keystone.domain

project*ű*

Optional.

Keystone project.

String value expected.

Can be updated without replacement.

Value must be of type keystone.project

role*ű*

Required.

Keystone role.

String value expected.

Can be updated without replacement.

Value must be of type keystone.role

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::Group
    properties:
      description: String
      domain: String
      name: String
      roles: [{"role": String, "project": String, "domain": String}, {"role":␣
↪String, "project": String, "domain": String}, ...]
```

OS::Keystone::GroupRoleAssignment

Available since 5.0.0 (Liberty) - Supported versions: keystone v3

Resource for granting roles to a group.

Resource for specifying groups and theirs roles.

Required Properties

group*ű*

Name or id of keystone group.

String value expected.

Can be updated without replacement.

Value must be of type keystone.group

Optional Properties

roles*ű*

List of role assignments.

List value expected.

Can be updated without replacement.

List contents:

Map between role with either project or domain.

Map value expected.

Can be updated without replacement.

Map properties:

domain*ű*

Optional.

Keystone domain.

String value expected.

Can be updated without replacement.

Value must be of type keystone.domain

project*ű*

Optional.

Keystone project.

String value expected.

Can be updated without replacement.

Value must be of type keystone.project

role*ű*

Required.

Keystone role.

String value expected.

Can be updated without replacement.

Value must be of type keystone.role

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::GroupRoleAssignment
    properties:
      group: String
      roles: [{"role": String, "project": String, "domain": String}, {"role":␣
↪String, "project": String, "domain": String}, ...]
```

OS::Keystone::Project

Available since 2015.1 (Kilo) - Supported versions: keystone v3

Heat Template Resource for Keystone Project.

Projects represent the base unit of ownership in OpenStack, in that all resources in OpenStack should be owned by a specific project. A project itself must be owned by a specific domain, and hence all project names are not globally unique, but unique to their domain. If the domain for a project is not specified, then it is added to the default domain.

Optional Properties

description*ű*

Description of keystone project.

String value expected.

Can be updated without replacement.

Defaults to `""`

domain*ű*

Name or id of keystone domain.

String value expected.

Can be updated without replacement.

Defaults to `"default"`

Value must be of type keystone.domain

enabled*ű*

This project is enabled or disabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

name*ű*

Name of keystone project.

String value expected.

Can be updated without replacement.

parent*ű*

Available since 6.0.0 (Mitaka)

The name or ID of parent of this keystone project in hierarchy.

String value expected.

Updates cause replacement.

Value must be of type keystone.project

tags*ű*

Available since 10.0.0 (Queens)

A list of tags for labeling and sorting projects.

List value expected.

Can be updated without replacement.

Defaults to `[]`

Attributes

domain_id*ű*

Available since 10.0.0 (Queens)

Domain id for project.

enabled*ű*

Available since 10.0.0 (Queens)

Flag of enable project.

is_domain*ű*

Available since 10.0.0 (Queens)

Indicates whether the project also acts as a domain.

name*ű*

Available since 10.0.0 (Queens)

Project name.

parent_id*ű*

Available since 10.0.0 (Queens)

Parent project id.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::Project
    properties:
      description: String
      domain: String
      enabled: Boolean
      name: String
      parent: String
      tags: [Value, Value, ...]
```

OS::Keystone::Region

Available since 6.0.0 (Mitaka) - Supported versions: keystone v3

Heat Template Resource for Keystone Region.

This plug-in helps to create, update and delete a keystone region. Also it can be used for enable or disable a given keystone region.

Optional Properties

description*ű*

Description of keystone region.

String value expected.

Can be updated without replacement.

enabled*ű*

This region is enabled or disabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

id*ű*

The user-defined region ID and should unique to the OpenStack deployment. While creating the region, heat will url encode this ID.

String value expected.

Updates cause replacement.

parent_region*ű*

If the region is hierarchically a child of another region, set this parameter to the ID of the parent region.

String value expected.

Can be updated without replacement.

Value must be of type keystone.region

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::Region
    properties:
      description: String
      enabled: Boolean
      id: String
      parent_region: String
```

OS::Keystone::Role

Available since 2015.1 (Kilo) - Supported versions: keystone v3

Heat Template Resource for Keystone Role.

Roles dictate the level of authorization the end user can obtain. Roles can be granted at either the domain or project level. Role can be assigned to the individual user or at the group level. Role name is unique within the owning domain.

Optional Properties

domain*ű*

Available since 16.0.0 (Wallaby)

Name or id of keystone domain.

String value expected.

Updates cause replacement.

Value must be of type keystone.domain

name*ű*

Name of keystone role.

String value expected.

Can be updated without replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::Role
    properties:
      domain: String
      name: String
```

OS::Keystone::Service

Available since 5.0.0 (Liberty) - Supported versions: keystone v3

Heat Template Resource for Keystone Service.

A resource that allows to create new service and manage it by Keystone.

Required Properties

type*ű*

Type of keystone Service.

String value expected.

Can be updated without replacement.

Optional Properties

description*ű*

Description of keystone service.

String value expected.

Can be updated without replacement.

enabled*ű*

Available since 6.0.0 (Mitaka)

This service is enabled or disabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

name*ű*

Name of keystone service.

String value expected.

Can be updated without replacement.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::Service
    properties:
      description: String
      enabled: Boolean
```

(continues on next page)

(continued from previous page)

```
    name: String
    type: String
```

OS::Keystone::User

Available since 2015.1 (Kilo) - Supported versions: keystone v3

Heat Template Resource for Keystone User.

Users represent an individual API consumer. A user itself must be owned by a specific domain, and hence all user names are not globally unique, but only unique to their domain.

Optional Properties

default_project*ií*

Name or ID of default project of keystone user.

String value expected.

Can be updated without replacement.

Value must be of type keystone.project

description*ií*

Description of keystone user.

String value expected.

Can be updated without replacement.

Defaults to ""

domain*ií*

Name or ID of keystone domain.

String value expected.

Can be updated without replacement.

Defaults to "default"

Value must be of type keystone.domain

email*ií*

Email address of keystone user.

String value expected.

Can be updated without replacement.

enabled*ií*

Keystone user is enabled or disabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

groups*ű*

Keystone user groups.

List value expected.

Can be updated without replacement.

List contents:

Optional.

Keystone user group.

String value expected.

Can be updated without replacement.

Value must be of type keystone.group

name*ű*

Name of keystone user.

String value expected.

Can be updated without replacement.

password*ű*

Password of keystone user.

String value expected.

Can be updated without replacement.

roles*ű*

List of role assignments.

List value expected.

Can be updated without replacement.

List contents:

Map between role with either project or domain.

Map value expected.

Can be updated without replacement.

Map properties:

domain*ű*

Optional.

Keystone domain.

String value expected.

Can be updated without replacement.

Value must be of type keystone.domain

project*ű*

Optional.

Keystone project.

String value expected.

Can be updated without replacement.

Value must be of type keystone.project

role*ű*

Required.

Keystone role.

String value expected.

Can be updated without replacement.

Value must be of type keystone.role

Attributes

default_project_id*ű*

Available since 9.0.0 (Pike)

Default project id for user.

domain_id*ű*

Available since 9.0.0 (Pike)

Domain id for user.

enabled*ű*

Available since 9.0.0 (Pike)

Flag of enable user.

name*ű*

Available since 9.0.0 (Pike)

User name.

password_expires_at*ű*

Available since 9.0.0 (Pike)

Show user password expiration time.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::User
    properties:
      default_project: String
      description: String
      domain: String
      email: String
      enabled: Boolean
      groups: [String, String, ...]
      name: String
      password: String
      roles: [{"role": String, "project": String, "domain": String}, {"role":␣
↪String, "project": String, "domain": String}, ...]
```

OS::Keystone::UserRoleAssignment

Available since 5.0.0 (Liberty) - Supported versions: keystone v3

Resource for granting roles to a user.

Resource for specifying users and theirs roles.

Required Properties

user*ű*

Name or id of keystone user.

String value expected.

Can be updated without replacement.

Value must be of type keystone.user

Optional Properties

roles*ű*

List of role assignments.

List value expected.

Can be updated without replacement.

List contents:

Map between role with either project or domain.

Map value expected.

Can be updated without replacement.

Map properties:

domain*ű*

Optional.

Keystone domain.

String value expected.

Can be updated without replacement.

Value must be of type keystone.domain

project*ű*

Optional.

Keystone project.

String value expected.

Can be updated without replacement.

Value must be of type keystone.project

role*ű*

Required.

Keystone role.

String value expected.

Can be updated without replacement.

Value must be of type keystone.role

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Keystone::UserRoleAssignment
    properties:
      roles: [{"role": String, "project": String, "domain": String}, {"role":␣
↪String, "project": String, "domain": String}, ...]
      user: String
```

OS::Magnum::Cluster

Available since 9.0.0 (Pike)

A resource that creates a magnum cluster.

This resource creates a magnum cluster, which is a collection of node objects where work is scheduled.

Required Properties

cluster_template*ří*

The name or ID of the cluster template.

String value expected.

Updates cause replacement.

Value must be of type magnum.cluster_template

Optional Properties

create_timeout*ří*

Timeout for creating the cluster in minutes. Set to 0 for no timeout.

Integer value expected.

Can be updated without replacement.

Defaults to `60`

The value must be at least 0.

discovery_url*ří*

Specifies a custom discovery url for node discovery.

String value expected.

Can be updated without replacement.

keypair*ří*

The name of the keypair. If not presented, use keypair in cluster template.

String value expected.

Updates cause replacement.

Value must be of type nova.keypair

master_count*ří*

The number of master nodes for this cluster.

Integer value expected.

Can be updated without replacement.

Defaults to `1`

The value must be at least 1.

name*ŕ*

The cluster name.

String value expected.

Updates cause replacement.

node_count*ŕ*

The node count for this cluster.

Integer value expected.

Can be updated without replacement.

Defaults to 1

The value must be at least 1.

Attributes

api_address*ŕ*
The endpoint URL of COE API exposed to end-users.

cluster_template_id*ŕ*
The UUID of the cluster template.

coe_version*ŕ*
Version info of chosen COE in cluster for helping client in picking the right version of client.

container_version*ŕ*
Version info of constainer engine in the chosen COE in cluster for helping client in picking the right version of client.

create_timeout*ŕ*
The timeout for cluster creation in minutes.

discovery_url*ŕ*
The custom discovery url for node discovery.

keypair*ŕ*
The name of the keypair.

master_addresses*ŕ*
List of floating IP of all master nodes.

master_count*ŕ*
The number of servers that will serve as master for the cluster.

name*ŕ*
Name of the resource.

node_addresses*ŕ*
List of floating IP of all servers that serve as node.

node_count*ŕ*
The number of servers that will serve as node in the cluster.

show*ŕ*
Detailed information about resource.

stack_id*ű*
The reference UUID of orchestration stack for this COE cluster.

status*ű*
The status for this COE cluster.

status_reason*ű*
The reason of cluster current status.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Magnum::Cluster
    properties:
      cluster_template: String
      create_timeout: Integer
      discovery_url: String
      keypair: String
      master_count: Integer
      name: String
      node_count: Integer
```

OS::Magnum::ClusterTemplate

Available since 9.0.0 (Pike)

A resource for the ClusterTemplate in Magnum.

ClusterTemplate is an object that stores template information about the cluster which is used to create new clusters consistently.

Required Properties

coe*ű*

The Container Orchestration Engine for cluster.

String value expected.

Updates cause replacement.

Allowed values: kubernetes, swarm, mesos

external_network*ű*

The external neutron network name or UUID to attach the Cluster.

String value expected.

Updates cause replacement.

Value must be of type neutron.network

image*ű*

The image name or UUID to use as a base image for cluster.

String value expected.

Updates cause replacement.

Value must be of type glance.image

Optional Properties

dns_nameserver*ű*

The DNS nameserver address.

String value expected.

Updates cause replacement.

Value must be of type ip_addr

docker_storage_driver*ű*

Select a docker storage driver.

String value expected.

Updates cause replacement.

Defaults to `"devicemapper"`

Allowed values: devicemapper, overlay

docker_volume_size*ű*

The size in GB of the docker volume.

Integer value expected.

Updates cause replacement.

The value must be at least 1.

fixed_network*ű*

The fixed neutron network name or UUID to attach the Cluster.

String value expected.

Updates cause replacement.

Value must be of type neutron.network

fixed_subnet*ű*

The fixed neutron subnet name or UUID to attach the Cluster.

String value expected.

Updates cause replacement.

Value must be of type neutron.subnet

flavor*ű*

The nova flavor name or UUID to use when launching the cluster.

String value expected.

Updates cause replacement.

Value must be of type nova.flavor

floating_ip_enabled*ű*

Indicates whether created clusters should have a floating ip or not.

Boolean value expected.

Updates cause replacement.

Defaults to `true`

http_proxy*ű*

The http_proxy address to use for nodes in cluster.

String value expected.

Updates cause replacement.

https_proxy*ű*

The https_proxy address to use for nodes in cluster.

String value expected.

Updates cause replacement.

keypair*ű*

The name of the SSH keypair to load into the cluster nodes.

String value expected.

Updates cause replacement.

Value must be of type nova.keypair

labels*ű*

Arbitrary labels in the form of key=value pairs to associate with cluster.

Map value expected.

Updates cause replacement.

master_flavor*ű*

The nova flavor name or UUID to use when launching the master node of the cluster.

String value expected.

Updates cause replacement.

Value must be of type nova.flavor

master_lb_enabled*ű*

Indicates whether created clusters should have a load balancer for master nodes or not.

Boolean value expected.

Updates cause replacement.

Defaults to `true`

name*ű*

The cluster template name.

String value expected.

Updates cause replacement.

network_driver*ű*

The name of the driver used for instantiating container networks. By default, Magnum will choose the pre-configured network driver based on COE type.

String value expected.

Updates cause replacement.

no_proxy*ű*

A comma separated list of addresses for which proxies should not be used in the cluster.

String value expected.

Updates cause replacement.

public*ű*

Make the cluster template public. To enable this option, you must own the right to publish in magnum. Which default set to admin only.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

registry_enabled*ű*

Enable the docker registry in the cluster.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

server_type*ű*

Specify the server type to be used.

String value expected.

Updates cause replacement.

Defaults to `"vm"`

Allowed values: vm, bm

tls_disabled*ű*

Disable TLS in the cluster.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

volume_driver*ű*

The volume driver name for instantiating container volume.

String value expected.

Updates cause replacement.

Allowed values: cinder, rexray

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Magnum::ClusterTemplate
    properties:
      coe: String
      dns_nameserver: String
      docker_storage_driver: String
      docker_volume_size: Integer
      external_network: String
      fixed_network: String
      fixed_subnet: String
      flavor: String
      floating_ip_enabled: Boolean
      http_proxy: String
      https_proxy: String
      image: String
      keypair: String
      labels: {...}
      master_flavor: String
      master_lb_enabled: Boolean
      name: String
      network_driver: String
      no_proxy: String
      public: Boolean
      registry_enabled: Boolean
      server_type: String
      tls_disabled: Boolean
      volume_driver: String
```

OS::Manila::SecurityService

Available since 5.0.0 (Liberty)

A resource that implements security service of Manila.

A security_service is a set of options that defines a security domain for a particular shared filesystem protocol, such as an Active Directory domain or a Kerberos domain.

Required Properties

type*ű*

Security service type.

String value expected.

Updates cause replacement.

Allowed values: ldap, kerberos, active_directory

Optional Properties

description*ű*

Security service description.

String value expected.

Can be updated without replacement.

dns_ip*ű*

DNS IP address used inside tenants network.

String value expected.

Can be updated without replacement.

domain*ű*

Security service domain.

String value expected.

Can be updated without replacement.

name*ű*

Security service name.

String value expected.

Can be updated without replacement.

password*ű*

Password used by user.

String value expected.

Can be updated without replacement.

server*ű*

Security service IP address or hostname.

String value expected.

Can be updated without replacement.

user*ű*

Security service user or group used by tenant.

String value expected.

Can be updated without replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Manila::SecurityService
    properties:
      description: String
      dns_ip: String
      domain: String
      name: String
      password: String
      server: String
      type: String
      user: String
```

OS::Manila::Share

Available since 5.0.0 (Liberty)

A resource that creates shared mountable file system.

The resource creates a manila share - shared mountable filesystem that can be attached to any client(or clients) that has a network access and permission to mount filesystem. Share is a unit of storage with specific size that supports pre-defined share protocol and advanced security model (access lists, share networks and security services).

Required Properties

share_protocol*ű*

Share protocol supported by shared filesystem.

String value expected.

Updates cause replacement.

Allowed values: NFS, CIFS, GlusterFS, HDFS, CEPHFS

size*ű*

Share storage size in GB.

Integer value expected.

Updates cause replacement.

Optional Properties

access_rules*ű*

A list of access rules that define access from IP to Share.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

access_level*ű*

Optional.

Level of access that need to be provided for guest.

String value expected.

Can be updated without replacement.

Allowed values: ro, rw

access_to*ű*

Required.

IP or other address information about guest that allowed to access to Share.

String value expected.

Can be updated without replacement.

access_type*ű*

Required.

Type of access that should be provided to guest.

String value expected.

Can be updated without replacement.

Allowed values: ip, user, cert, cephx

description*ŕ*

Share description.

String value expected.

Can be updated without replacement.

is_public*ŕ*

Defines if shared filesystem is public or private.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

metadata*ŕ*

Metadata key-values defined for share.

Map value expected.

Can be updated without replacement.

name*ŕ*

Share name.

String value expected.

Can be updated without replacement.

share_network*ŕ*

Name or ID of shared network defined for shared filesystem.

String value expected.

Updates cause replacement.

Value must be of type manila.share_network

share_type*ŕ*

Name or ID of shared filesystem type. Types defines some share filesystem profiles that will be used for share creation.

String value expected.

Updates cause replacement.

Value must be of type manila.share_type

snapshot*ŕ*

Name or ID of shared file system snapshot that will be restored and created as a new share.

String value expected.

Updates cause replacement.

Value must be of type manila.share_snapshot

Attributes

availability_zone*ú*
The availability zone of shared filesystem.

created_at*ú*
Datetime when a share was created.

export_locations*ú*
Export locations of share.

host*ú*
Share host.

project_id*ú*
Share project ID.

share_server_id*ú*
ID of server (VM, etc) on host that is used for exporting network file-system.

show*ú*
Detailed information about resource.

status*ú*
Current share status.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Manila::Share
    properties:
      access_rules: [{"access_to": String, "access_type": String, "access_
↪level": String}, {"access_to": String, "access_type": String, "access_level
↪": String}, ...]
      description: String
      is_public: Boolean
      metadata: {...}
      name: String
      share_network: String
      share_protocol: String
      share_type: String
      size: Integer
      snapshot: String
```

OS::Manila::ShareNetwork

Available since 5.0.0 (Liberty)

A resource that stores network information for share servers.

Stores network information that will be used by share servers, where shares are hosted.

Optional Properties

description*ű*

Share network description.

String value expected.

Can be updated without replacement.

name*ű*

Name of the share network.

String value expected.

Can be updated without replacement.

neutron_network*ű*

Neutron network id.

String value expected.

Can be updated without replacement.

Value must be of type neutron.network

neutron_subnet*ű*

Neutron subnet id.

String value expected.

Can be updated without replacement.

Value must be of type neutron.subnet

security_services*ű*

A list of security services IDs or names.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Attributes

cidr*ű*
CIDR of subnet.

ip_version*ű*
Version of IP address.

network_type*ű*
The physical mechanism by which the virtual network is implemented.

segmentation_id*ű*
VLAN ID for VLAN networks or tunnel-id for GRE/VXLAN networks.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Manila::ShareNetwork
    properties:
      description: String
      name: String
      neutron_network: String
      neutron_subnet: String
      security_services: [String, String, ...]
```

OS::Manila::ShareType

Available since 5.0.0 (Liberty)

A resource for creating manila share type.

A share_type is an administrator-defined type of service, comprised of a tenant visible description, and a list of non-tenant-visible key/value pairs (extra_specs) which the Manila scheduler uses to make scheduling decisions for shared filesystem tasks.

Please note that share type is intended to use mostly by administrators. So it is very likely that Manila will prohibit creation of the resource without administration grants.

Required Properties

driver_handles_share_servers*ű*

Required extra specification. Defines if share drivers handles share servers.

Boolean value expected.

Updates cause replacement.

name*ű*

Name of the share type.

String value expected.

Updates cause replacement.

Optional Properties

extra_specs*ű*

Extra specs key-value pairs defined for share type.

Map value expected.

Can be updated without replacement.

is_public*ű*

Defines if share type is accessible to the public.

Boolean value expected.

Updates cause replacement.

Defaults to `true`

snapshot_support*ű*

> Available since 6.0.0 (Mitaka)

Boolean extra spec that used for filtering of backends by their capability to create share snapshots.

Boolean value expected.

Updates cause replacement.

Defaults to `true`

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Manila::ShareType
    properties:
      driver_handles_share_servers: Boolean
      extra_specs: {...}
      is_public: Boolean
```

(continues on next page)

(continued from previous page)

```
    name: String
    snapshot_support: Boolean
```

OS::Mistral::CronTrigger

Available since 5.0.0 (Liberty)

A resource implements Mistral cron trigger.

Cron trigger is an object allowing to run workflow on a schedule. User specifies what workflow with what input needs to be run and also specifies how often it should be run. Pattern property is used to describe the frequency of workflow execution.

Required Properties

workflow*ű*

Workflow to execute.

Map value expected.

Updates cause replacement.

Map properties:

input*ű*

Input values for the workflow.

Map value expected.

Updates cause replacement.

name*ű*

Required.

Name or ID of the workflow.

String value expected.

Updates cause replacement.

Value must be of type mistral.workflow

Optional Properties

count*ű*

Remaining executions.

Integer value expected.

Updates cause replacement.

first_time*ű*

Time of the first execution in format YYYY-MM-DD HH:MM.

String value expected.

Updates cause replacement.

name*ű*

Name of the cron trigger.

String value expected.

Updates cause replacement.

pattern*ű*

Cron expression.

String value expected.

Updates cause replacement.

Value must be of type cron_expression

Attributes

next_execution_time*ű*
Time of the next execution in format YYYY-MM-DD HH:MM:SS.

remaining_executions*ű*
Number of remaining executions.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Mistral::CronTrigger
    properties:
      count: Integer
      first_time: String
      name: String
      pattern: String
      workflow: {"name": String, "input": {...}}
```

OS::Mistral::ExternalResource

Available since 9.0.0 (Pike)

A plugin for managing user-defined resources via Mistral workflows.

This resource allows users to manage resources that are not known to Heat. The user may specify a Mistral workflow to handle each resource action, such as CREATE, UPDATE, or DELETE.

The workflows may return an output named resource_id, which will be treated as the physical ID of the resource by Heat.

Once the resource is created, subsequent workflow runs will receive the output of the last workflow execution in the heat_extresource_data key in the workflow environment (accessible as `env().heat_extresource_data` in the workflow).

The template author may specify a subset of inputs as causing replacement of the resource when they change, as an alternative to running the UPDATE workflow.

Required Properties

actions*ú*

Resource action which triggers a workflow execution.

Map value expected.

Updates cause replacement.

Map properties:

CREATE*ú*

Dictionary which defines the workflow to run and its params.

Map value expected.

Updates cause replacement.

Map properties:

params*ú*

Workflow additional parameters. If workflow is reverse typed, params requires task_name, which defines initial task.

Map value expected.

Updates cause replacement.

Defaults to `{}`

workflow*ú*

Required.

Workflow to execute.

String value expected.

Updates cause replacement.

Value must be of type mistral.workflow

DELETE*ú*

Dictionary which defines the workflow to run and its params.

Map value expected.

Updates cause replacement.

Map properties:

paramsűű

Workflow additional parameters. If workflow is reverse typed, params requires task_name, which defines initial task.

Map value expected.

Updates cause replacement.

Defaults to `{}`

workflowűű

Required.

Workflow to execute.

String value expected.

Updates cause replacement.

Value must be of type mistral.workflow

RESUMEűű

Dictionary which defines the workflow to run and its params.

Map value expected.

Updates cause replacement.

Map properties:

paramsűű

Workflow additional parameters. If workflow is reverse typed, params requires task_name, which defines initial task.

Map value expected.

Updates cause replacement.

Defaults to `{}`

workflowűű

Required.

Workflow to execute.

String value expected.

Updates cause replacement.

Value must be of type mistral.workflow

SUSPENDűű

Dictionary which defines the workflow to run and its params.

Map value expected.

Updates cause replacement.

Map properties:

paramsűű

Workflow additional parameters. If workflow is reverse typed, params requires task_name, which defines initial task.

Map value expected.

Updates cause replacement.

Defaults to `{}`

workflow*ű*

Required.

Workflow to execute.

String value expected.

Updates cause replacement.

Value must be of type mistral.workflow

UPDATE*ű*

Dictionary which defines the workflow to run and its params.

Map value expected.

Updates cause replacement.

Map properties:

params*ű*

Workflow additional parameters. If workflow is reverse typed, params requires task_name, which defines initial task.

Map value expected.

Updates cause replacement.

Defaults to `{}`

workflow*ű*

Required.

Workflow to execute.

String value expected.

Updates cause replacement.

Value must be of type mistral.workflow

Optional Properties

always_update*ű*

Triggers UPDATE action execution even if input is unchanged.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

description*ŕí*

Workflow execution description.

String value expected.

Updates cause replacement.

Defaults to `"Heat managed"`

input*ŕí*

Dictionary which contains input for the workflows.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

replace_on_change_inputs*ŕí*

A list of inputs that should cause the resource to be replaced when their values change.

List value expected.

Updates cause replacement.

Defaults to `[]`

Attributes

output*ŕí*
: Output from the execution.

show*ŕí*
: Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Mistral::ExternalResource
    properties:
      actions: {"CREATE": {"workflow": String, "params": {...}}, "UPDATE": {
↪"workflow": String, "params": {...}}, "SUSPEND": {"workflow": String,
↪"params": {...}}, "RESUME": {"workflow": String, "params": {...}}, "DELETE
↪": {"workflow": String, "params": {...}}}
      always_update: Boolean
      description: String
      input: {...}
      replace_on_change_inputs: [Value, Value, ...]
```

OS::Mistral::Workflow

Available since 2015.1 (Kilo)

A resource that implements Mistral workflow.

Workflow represents a process that can be described in a various number of ways and that can do some job interesting to the end user. Each workflow consists of tasks (at least one) describing what exact steps should be made during workflow execution.

For detailed description how to use Workflow, read Mistral documentation.

Required Properties

tasks*ú*

Dictionary containing workflow tasks.

List value expected.

Can be updated without replacement.

The length must be at least 1.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

action*ú*

Optional.

Name of the action associated with the task. Either action or workflow may be defined in the task.

String value expected.

Can be updated without replacement.

concurrency*ú*

Available since 8.0.0 (Ocata)

Optional.

Defines a max number of actions running simultaneously in a task. Applicable only for tasks that have with-items.

Integer value expected.

Can be updated without replacement.

description*ú*

Optional.

Task description.

String value expected.

Can be updated without replacement.

input*ű*

Actual input parameter values of the task.

Map value expected.

Can be updated without replacement.

join*ű*

Available since 6.0.0 (Mitaka)

Optional.

Allows to synchronize multiple parallel workflow branches and aggregate their data. Valid inputs: all - the task will run only if all upstream tasks are completed. Any numeric value - then the task will run once at least this number of upstream tasks are completed and corresponding conditions have triggered.

String value expected.

Can be updated without replacement.

keep_result*ű*

Available since 5.0.0 (Liberty)

Optional.

Allowing not to store action results after task completion.

Boolean value expected.

Can be updated without replacement.

name*ű*

Required.

Task name.

String value expected.

Can be updated without replacement.

on_complete*ű*

List of tasks which will run after the task has completed regardless of whether it is successful or not.

List value expected.

Can be updated without replacement.

on_error*ű*

List of tasks which will run after the task has completed with an error.

List value expected.

Can be updated without replacement.

on_success*ű*

List of tasks which will run after the task has completed successfully.

List value expected.

Can be updated without replacement.

pause_before*ű*

Available since 5.0.0 (Liberty)

Optional.

Defines whether Mistral Engine should put the workflow on hold or not before starting a task.

Boolean value expected.

Can be updated without replacement.

publish*ű*

Dictionary of variables to publish to the workflow context.

Map value expected.

Can be updated without replacement.

requires*ű*

List of tasks which should be executed before this task. Used only in reverse workflows.

List value expected.

Can be updated without replacement.

retry*ű*

Available since 5.0.0 (Liberty)

Defines a pattern how task should be repeated in case of an error.

Map value expected.

Can be updated without replacement.

target*ű*

Available since 5.0.0 (Liberty)

Optional.

It defines an executor to which task action should be sent to.

String value expected.

Can be updated without replacement.

timeout*ű*

Available since 5.0.0 (Liberty)

Optional.

Defines a period of time in seconds after which a task will be failed automatically by engine if hasnt completed.

Integer value expected.

Can be updated without replacement.

wait_after*ű*

Available since 5.0.0 (Liberty)

Optional.

Defines a delay in seconds that Mistral Engine should wait after a task has completed before starting next tasks defined in on-success, on-error or on-complete.

Integer value expected.

Can be updated without replacement.

wait_before*ű*

Available since 5.0.0 (Liberty)

Optional.

Defines a delay in seconds that Mistral Engine should wait before starting a task.

Integer value expected.

Can be updated without replacement.

with_items*ű*

Available since 5.0.0 (Liberty)

Optional.

If configured, it allows to run action or workflow associated with a task multiple times on a provided list of items.

String value expected.

Can be updated without replacement.

workflow*ű*

Optional.

Name of the workflow associated with the task. Can be defined by intrinsic function get_resource or by name of the referenced workflow, i.e. { workflow: wf_name } or { workflow: { get_resource: wf_name }}. Either action or workflow may be defined in the task.

String value expected.

Can be updated without replacement.

Value must be of type mistral.workflow

type*ří*

Workflow type.

String value expected.

Can be updated without replacement.

Allowed values: direct, reverse

Optional Properties

description*ří*

Workflow description.

String value expected.

Can be updated without replacement.

input*ří*

Dictionary which contains input for workflow.

Map value expected.

Can be updated without replacement.

name*ří*

Workflow name.

String value expected.

Updates cause replacement.

output*ří*

Any data structure arbitrarily containing YAQL expressions that defines workflow output. May be nested.

Map value expected.

Can be updated without replacement.

params*ří*

Workflow additional parameters. If Workflow is reverse typed, params requires task_name, which defines initial task.

Map value expected.

Can be updated without replacement.

tags*ű*

Available since 10.0.0 (Queens)

List of tags to set on the workflow.

List value expected.

Can be updated without replacement.

task_defaults*ű*

Available since 5.0.0 (Liberty)

Default settings for some of task attributes defined at workflow level.

Map value expected.

Can be updated without replacement.

Map properties:

concurrency*ű*

Available since 8.0.0 (Ocata)

Optional.

Defines a max number of actions running simultaneously in a task. Applicable only for tasks that have with-items.

Integer value expected.

Can be updated without replacement.

on_complete*ű*

List of tasks which will run after the task has completed regardless of whether it is successful or not.

List value expected.

Can be updated without replacement.

on_error*ű*

List of tasks which will run after the task has completed with an error.

List value expected.

Can be updated without replacement.

on_success*ű*

List of tasks which will run after the task has completed successfully.

List value expected.

Can be updated without replacement.

pause_before*ű*

Optional.

Defines whether Mistral Engine should put the workflow on hold or not before starting a task.

Boolean value expected.

Can be updated without replacement.

requires*ű*

List of tasks which should be executed before this task. Used only in reverse workflows.

List value expected.

Can be updated without replacement.

retry*ű*

Defines a pattern how task should be repeated in case of an error.

Map value expected.

Can be updated without replacement.

timeout*ű*

Optional.

Defines a period of time in seconds after which a task will be failed automatically by engine if hasnt completed.

Integer value expected.

Can be updated without replacement.

wait_after*ű*

Optional.

Defines a delay in seconds that Mistral Engine should wait after a task has completed before starting next tasks defined in on-success, on-error or on-complete.

Integer value expected.

Can be updated without replacement.

wait_before*ű*

Optional.

Defines a delay in seconds that Mistral Engine should wait before starting a task.

Integer value expected.

Can be updated without replacement.

use_request_body_as_input*ű*

Available since 6.0.0 (Mitaka)

Defines the method in which the request body for signaling a workflow would be parsed. In case this property is set to True, the body would be parsed as a simple json where each key is a workflow input, in other cases body would be parsed expecting a specific json format with two keys: input and params.

Boolean value expected.

Can be updated without replacement.

Attributes

alarm_url*ű*
A signed url to create executions for workflows specified in Workflow resource.

data*ű*
A dictionary which contains name and input of the workflow.

executions*ű*
List of workflows executions, each of them is a dictionary with information about execution. Each dictionary returns values for next keys: id, workflow_name, created_at, updated_at, state for current execution state, input, output.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Mistral::Workflow
    properties:
      description: String
      input: {...}
      name: String
      output: {...}
      params: {...}
      tags: [Value, Value, ...]
      task_defaults: {"on_success": [Value, Value, ...], "on_error": [Value,␣
↪Value, ...], "on_complete": [Value, Value, ...], "requires": [Value, Value,␣
↪...], "retry": {...}, "wait_before": Integer, "wait_after": Integer, "pause_
↪before": Boolean, "timeout": Integer, "concurrency": Integer}
      tasks: [{"name": String, "description": String, "input": {...}, "action
↪": String, "workflow": String, "publish": {...}, "on_success": [Value,␣
↪Value, ...], "on_error": [Value, Value, ...], "on_complete": [Value, Value,␣
↪...], "policies": {...}, "requires": [Value, Value, ...], "retry": {...},
↪"wait_before": Integer, "wait_after": Integer, "pause_before": Boolean,
↪"timeout": Integer, "with_items": String, "keep_result": Boolean,
↪"concurrency": Integer, "target": String, "join": String}, {"name": String,
↪"description": String, "input": {...}, "action": String, "workflow": String,
↪ "publish": {...}, "on_success": [Value, Value, ...], "on_error": [Value,␣
↪Value, ...], "on_complete": [Value, Value, ...], "policies": {...},
↪"requires": [Value, Value, ...], "retry": {...}, "wait_before": Integer,
↪"wait_after": Integer, "pause_before": Boolean, "timeout": Integer, "with_
↪items": String, "keep_result": Boolean, "concurrency": Integer, "target":␣
```

(continues on next page)

(continued from previous page)

```
↪String, "join": String}, ...]
      type: String
      use_request_body_as_input: Boolean
```

OS::Neutron::AddressScope

Available since 6.0.0 (Mitaka)

A resource for Neutron address scope.

This resource can be associated with multiple subnet pools in a one-to-many relationship. The subnet pools under an address scope must not overlap.

Optional Properties

ip_version*ű*

Address family of the address scope, which is 4 or 6.

Integer value expected.

Updates cause replacement.

Defaults to `4`

Allowed values: 4, 6

name*ű*

The name for the address scope.

String value expected.

Can be updated without replacement.

shared*ű*

Whether the address scope should be shared to other tenants. Note that the default policy setting restricts usage of this attribute to administrative users only, and restricts changing of shared address scope to unshared with update.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

tenant_id*ű*

The owner tenant ID of the address scope. Only administrative users can specify a tenant ID other than their own.

String value expected.

Updates cause replacement.

Value must be of type keystone.project

Attributes

show*ú*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::AddressScope
    properties:
      ip_version: Integer
      name: String
      shared: Boolean
      tenant_id: String
```

OS::Neutron::ExtraRouteSet

Available since 14.0.0 (Ussuri)

Resource for specifying extra routes for a Neutron router.

Requires Neutron `extraroute-atomic` extension to be enabled:

```
$ openstack extension show extraroute-atomic
```

An extra route is a static routing table entry that is added beyond the routes managed implicitly by router interfaces and router gateways.

The `destination` of an extra route is any IP network in /CIDR notation. The `nexthop` of an extra route is an IP in a subnet that is directly connected to the router.

In a single OS::Neutron::ExtraRouteSet resource you can specify a set of extra routes (represented as a list) on the same virtual router. As an improvement over the (never formally supported) OS::Neutron::ExtraRoute resource this resource plugin uses a Neutron API extension (`extraroute-atomic`) that is not prone to race conditions when used to manage multiple extra routes of the same router. It is safe to manage multiple extra routes of the same router from multiple stacks.

On the other hand use of the same route on the same router is not safe from multiple stacks (or between Heat and non-Heat managed Neutron extra routes).

Required Properties

router*ú*

The router id.

String value expected.

Updates cause replacement.

Value must be of type neutron.router

Optional Properties

routes*ŕ*

A set of route dictionaries for the router.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

destination*ŕ*

Required.

The destination network in CIDR notation.

String value expected.

Can be updated without replacement.

Value must be of type net_cidr

nexthop*ŕ*

Required.

The next hop for the destination.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

Attributes

show*ŕ*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::ExtraRouteSet
    properties:
```

(continues on next page)

(continued from previous page)

```
    router: String
    routes: [{"destination": String, "nexthop": String}, {"destination":␣
↪String, "nexthop": String}, ...]
```

OS::Neutron::Firewall

A resource for the Firewall resource in Neutron FWaaS.

Resource for using the Neutron firewall implementation. Firewall is a network security system that monitors and controls the incoming and outgoing network traffic based on predetermined security rules.

Required Properties

firewall_policy_id*ű*

The ID of the firewall policy that this firewall is associated with.

String value expected.

Can be updated without replacement.

Optional Properties

admin_state_up*ű*

Administrative state of the firewall. If false (down), firewall does not forward packets and will drop all traffic to/from VMs behind the firewall.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

description*ű*

Description for the firewall.

String value expected.

Can be updated without replacement.

name*ű*

Name for the firewall.

String value expected.

Can be updated without replacement.

value_specs*ű*

> Available since 5.0.0 (Liberty)

Extra parameters to include in the request. Parameters are often specific to installed hardware or extensions.

Map value expected.

Can be updated without replacement.

Defaults to {}

shared*ű*

> UNSUPPORTED since 6.0.0 (Mitaka) - There is no such option during 5.0.0, so need to make this property unsupported while it not used.

> Available since 2015.1 (Kilo)

Whether this firewall should be shared across all tenants. NOTE: The default policy setting in Neutron restricts usage of this property to administrative users only.

Boolean value expected.

Can be updated without replacement.

Attributes

admin_state_up*ű*
The administrative state of the firewall.

description*ű*
Description of the firewall.

firewall_policy_id*ű*
Unique identifier of the firewall policy used to create the firewall.

name*ű*
Name for the firewall.

shared*ű*

> UNSUPPORTED since 6.0.0 (Mitaka) - There is no such option during 5.0.0, so need to make this attribute unsupported, otherwise error will raised.

Shared status of this firewall.

show*ű*
Detailed information about resource.

status*ű*
The status of the firewall.

tenant_id*ű*
Id of the tenant owning the firewall.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
```

(continues on next page)

(continued from previous page)

```
...
the_resource:
  type: OS::Neutron::Firewall
  properties:
    admin_state_up: Boolean
    description: String
    firewall_policy_id: String
    name: String
    value_specs: {...}
```

OS::Neutron::FirewallPolicy

A resource for the FirewallPolicy resource in Neutron FWaaS.

FirewallPolicy resource is an ordered collection of firewall rules. A firewall policy can be shared across tenants.

Optional Properties

audited*ŕ*

Whether this policy should be audited. When set to True, each time the firewall policy or the associated firewall rules are changed, this attribute will be set to False and will have to be explicitly set to True through an update operation.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

description*ŕ*

Description for the firewall policy.

String value expected.

Can be updated without replacement.

firewall_rules*ŕ*

An ordered list of firewall rules to apply to the firewall. (Prior to version 14.0.0 this was a required property).

List value expected.

Can be updated without replacement.

name*ŕ*

Name for the firewall policy.

String value expected.

Can be updated without replacement.

shared*ŕ*

Whether this policy should be shared across all tenants.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

Attributes

audited*ű*
Audit status of this firewall policy.

description*ű*
Description of the firewall policy.

firewall_rules*ű*
List of firewall rules in this firewall policy.

name*ű*
Name for the firewall policy.

shared*ű*
Shared status of this firewall policy.

show*ű*
Detailed information about resource.

tenant_id*ű*
Id of the tenant owning the firewall policy.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::FirewallPolicy
    properties:
      audited: Boolean
      description: String
      firewall_rules: [Value, Value, ...]
      name: String
      shared: Boolean
```

OS::Neutron::FirewallRule

A resource for the FirewallRule resource in Neutron FWaaS.

FirewallRule represents a collection of attributes like ports, ip addresses etc. which define match criteria and action (allow, or deny) that needs to be taken on the matched data traffic.

Optional Properties

action*ű*

Action to be performed on the traffic matching the rule.

String value expected.

Can be updated without replacement.

Defaults to `"deny"`

Allowed values: allow, deny

description*ű*

Description for the firewall rule.

String value expected.

Can be updated without replacement.

destination_ip_address*ű*

Destination IP address or CIDR.

String value expected.

Can be updated without replacement.

Value must be of type net_cidr

destination_port*ű*

Destination port number or a range.

String value expected.

Can be updated without replacement.

enabled*ű*

Whether this rule should be enabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

ip_version*ű*

Internet protocol version.

String value expected.

Can be updated without replacement.

Defaults to `"4"`

Allowed values: 4, 6

name*ű*

Name for the firewall rule.

String value expected.

Can be updated without replacement.

protocol*ű*

Protocol for the firewall rule.

String value expected.

Can be updated without replacement.

Defaults to `"any"`

Allowed values: tcp, udp, icmp, any

shared*ű*

Whether this rule should be shared across all tenants.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

source_ip_address*ű*

Source IP address or CIDR.

String value expected.

Can be updated without replacement.

Value must be of type net_cidr

source_port*ű*

Source port number or a range.

String value expected.

Can be updated without replacement.

Attributes

action*ű*
: Allow or deny action for this firewall rule.

description*ű*
: Description of the firewall rule.

destination_ip_address*ű*
: Destination ip_address for this firewall rule.

destination_port*ű*
: Destination port range for this firewall rule.

enabled*ű*
: Indicates whether this firewall rule is enabled or not.

firewall_policy_id*ű*
: Unique identifier of the firewall policy to which this firewall rule belongs.

ip_version*ű*
: Ip_version for this firewall rule.

name*ű*
Name for the firewall rule.

position*ű*
Position of the rule within the firewall policy.

protocol*ű*
Protocol value for this firewall rule.

shared*ű*
Shared status of this firewall rule.

show*ű*
Detailed information about resource.

source_ip_address*ű*
Source ip_address for this firewall rule.

source_port*ű*
Source port range for this firewall rule.

tenant_id*ű*
Id of the tenant owning the firewall.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::FirewallRule
    properties:
      action: String
      description: String
      destination_ip_address: String
      destination_port: String
      enabled: Boolean
      ip_version: String
      name: String
      protocol: String
      shared: Boolean
      source_ip_address: String
      source_port: String
```

OS::Neutron::FloatingIP

A resource for managing Neutron floating ips.

Floating IP addresses can change their association between routers by action of the user. One of the most common use cases for floating IPs is to provide public IP addresses to a private cloud, where there are a limited number of IP addresses available. Another is for a public cloud user to have a static IP address that can be reassigned when an instance is upgraded or moved.

Required Properties

floating_network*ű*

Available since 2014.2 (Juno)

Network to allocate floating IP from.

String value expected.

Updates cause replacement.

Value must be of type neutron.network

Optional Properties

dns_domain*ű*

Available since 7.0.0 (Newton)

DNS domain associated with floating ip.

String value expected.

Can be updated without replacement.

Value must be of type dns_domain

dns_name*ű*

Available since 7.0.0 (Newton)

DNS name associated with floating ip.

String value expected.

Can be updated without replacement.

Value must be of type rel_dns_name

fixed_ip_address*ű*

IP address to use if the port has multiple addresses.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

floating_ip_address*ű*

Available since 5.0.0 (Liberty)

IP address of the floating IP. NOTE: The default policy setting in Neutron restricts usage of this property to administrative users only.

String value expected.

Updates cause replacement.

Value must be of type ip_addr

floating_subnet*ű*

Available since 9.0.0 (Pike)

Subnet to allocate floating IP from.

String value expected.

Updates cause replacement.

Value must be of type neutron.subnet

port_id*ű*

ID of an existing port with at least one IP address to associate with this floating IP.

String value expected.

Can be updated without replacement.

Value must be of type neutron.port

value_specs*ű*

Extra parameters to include in the floatingip object in the creation request. Parameters are often specific to installed hardware or extensions.

Map value expected.

Updates cause replacement.

Defaults to `{}`

Attributes

fixed_ip_address*ű*
IP address of the associated port, if specified.

floating_ip_address*ű*
The allocated address of this IP.

floating_network_id*ű*
ID of the network in which this IP is allocated.

port_id*ű*
ID of the port associated with this IP.

router_id*ű*
ID of the router used as gateway, set when associated with a port.

show*ű*
Detailed information about resource.

tenant_id*ű*
The tenant owning this floating IP.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::FloatingIP
    properties:
      dns_domain: String
      dns_name: String
      fixed_ip_address: String
      floating_ip_address: String
      floating_network: String
      floating_subnet: String
      port_id: String
      value_specs: {...}
```

OS::Neutron::FloatingIPAssociation

A resource for associating floating ips and ports.

This resource allows associating a floating IP to a port with at least one IP address to associate with this floating IP.

Required Properties

floatingip_id*ŕ*

ID of the floating IP to associate.

String value expected.

Can be updated without replacement.

port_id*ŕ*

ID of an existing port with at least one IP address to associate with this floating IP.

String value expected.

Can be updated without replacement.

Value must be of type neutron.port

Optional Properties

fixed_ip_address*ŕ*

IP address to use if the port has multiple addresses.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::FloatingIPAssociation
    properties:
      fixed_ip_address: String
      floatingip_id: String
      port_id: String
```

OS::Neutron::FloatingIPPortForward

Available since 19.0.0 (Zed)

A resource for creating port forwarding for floating IPs.

This resource creates port forwarding for floating IPs. These are sub-resource of exsisting Floating ips, which requires the service_plugin and extension port_forwarding enabled and that the floating ip is not associated with a neutron port.

Required Properties

external_port*ű*

External port address to port forward from.

Integer value expected.

Can be updated without replacement.

The value must be in the range 1 to 65535.

floating_ip*ű*

Name or ID of the floating IP create port forwarding on.

String value expected.

Updates cause replacement.

internal_ip_address*ű*

Internal IP address to port forwarded to.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

internal_port*ű*

Name or ID of the internal_ip_address port.

String value expected.

Can be updated without replacement.

Value must be of type neutron.port

protocol*ű*

Port protocol to forward.

String value expected.

Can be updated without replacement.

Allowed values: tcp, udp, icmp, icmp6, sctp, dccp

Optional Properties

internal_port_number*ű*

Internal port number to port forward to.

Integer value expected.

Can be updated without replacement.

The value must be in the range 1 to 65535.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::FloatingIPPortForward
    properties:
      external_port: Integer
      floating_ip: String
      internal_ip_address: String
      internal_port: String
      internal_port_number: Integer
      protocol: String
```

OS::Neutron::IKEPolicy

A resource for IKE policy in Neutron.

The Internet Key Exchange policy identifies the authentication and encryption algorithm used during phase one and phase two negotiation of a VPN connection.

Optional Properties

auth_algorithm*ű*

Authentication hash algorithm for the ike policy.

String value expected.

Can be updated without replacement.

Defaults to `"sha1"`

Allowed values: sha1, sha256, sha384, sha512

description*ű*

Description for the ike policy.

String value expected.

Can be updated without replacement.

encryption_algorithm*ű*

Encryption algorithm for the ike policy.

String value expected.

Can be updated without replacement.

Defaults to `"aes-128"`

Allowed values: 3des, aes-128, aes-192, aes-256

ike_version*ű*

Version for the ike policy.

String value expected.

Can be updated without replacement.

Defaults to `"v1"`

Allowed values: v1, v2

lifetime*ű*

Safety assessment lifetime configuration for the ike policy.

Map value expected.

Can be updated without replacement.

Map properties:

units*ű*

Optional.

Safety assessment lifetime units.

String value expected.

Can be updated without replacement.

Defaults to `"seconds"`

Allowed values: seconds, kilobytes

value*ří*

Optional.

Safety assessment lifetime value in specified units.

Integer value expected.

Can be updated without replacement.

Defaults to `3600`

name*ří*

Name for the ike policy.

String value expected.

Can be updated without replacement.

pfs*ří*

Perfect forward secrecy in lowercase for the ike policy.

String value expected.

Can be updated without replacement.

Defaults to `"group5"`

Allowed values: group2, group5, group14

phase1_negotiation_mode*ří*

Negotiation mode for the ike policy.

String value expected.

Updates cause replacement.

Defaults to `"main"`

Allowed values: main

Attributes

auth_algorithm*ří*
The authentication hash algorithm used by the ike policy.

description*ří*
The description of the ike policy.

encryption_algorithm*ří*
The encryption algorithm used by the ike policy.

ike_version*ű*
The version of the ike policy.

lifetime*ű*
The safety assessment lifetime configuration for the ike policy.

name*ű*
The name of the ike policy.

pfs*ű*
The perfect forward secrecy of the ike policy.

phase1_negotiation_mode*ű*
The negotiation mode of the ike policy.

show*ű*
Detailed information about resource.

tenant_id*ű*
The unique identifier of the tenant owning the ike policy.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::IKEPolicy
    properties:
      auth_algorithm: String
      description: String
      encryption_algorithm: String
      ike_version: String
      lifetime: {"units": String, "value": Integer}
      name: String
      pfs: String
      phase1_negotiation_mode: String
```

OS::Neutron::IPsecPolicy

A resource for IPsec policy in Neutron.

The IP security policy specifying the authentication and encryption algorithm, and encapsulation mode used for the established VPN connection.

Optional Properties

auth_algorithm*ű*

Authentication hash algorithm for the ipsec policy.

String value expected.

Updates cause replacement.

Defaults to "sha1"

Allowed values: sha1

description*ŕ*

Description for the ipsec policy.

String value expected.

Can be updated without replacement.

encapsulation_mode*ŕ*

Encapsulation mode for the ipsec policy.

String value expected.

Updates cause replacement.

Defaults to "tunnel"

Allowed values: tunnel, transport

encryption_algorithm*ŕ*

Encryption algorithm for the ipsec policy.

String value expected.

Updates cause replacement.

Defaults to "aes-128"

Allowed values: 3des, aes-128, aes-192, aes-256

lifetime*ŕ*

Safety assessment lifetime configuration for the ipsec policy.

Map value expected.

Updates cause replacement.

Map properties:

units*ŕ*

Optional.

Safety assessment lifetime units.

String value expected.

Updates cause replacement.

Defaults to "seconds"

Allowed values: seconds, kilobytes

value*ŕ*

Optional.

Safety assessment lifetime value in specified units.

Integer value expected.

Updates cause replacement.

Defaults to `3600`

name*ŕ*

Name for the ipsec policy.

String value expected.

Can be updated without replacement.

pfs*ŕ*

Perfect forward secrecy for the ipsec policy.

String value expected.

Updates cause replacement.

Defaults to `"group5"`

Allowed values: group2, group5, group14

transform_protocol*ŕ*

Transform protocol for the ipsec policy.

String value expected.

Updates cause replacement.

Defaults to `"esp"`

Allowed values: esp, ah, ah-esp

Attributes

auth_algorithm*ŕ*
The authentication hash algorithm of the ipsec policy.

description*ŕ*
The description of the ipsec policy.

encapsulation_mode*ŕ*
The encapsulation mode of the ipsec policy.

encryption_algorithm*ŕ*
The encryption algorithm of the ipsec policy.

lifetime*ŕ*
The safety assessment lifetime configuration of the ipsec policy.

name*ŕ*
The name of the ipsec policy.

pfs*ŕ*
The perfect forward secrecy of the ipsec policy.

show*ŕ*
Detailed information about resource.

tenant_id*ŕ*
The unique identifier of the tenant owning the ipsec policy.

transform_protocolǘ
The transform protocol of the ipsec policy.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::IPsecPolicy
    properties:
      auth_algorithm: String
      description: String
      encapsulation_mode: String
      encryption_algorithm: String
      lifetime: {"units": String, "value": Integer}
      name: String
      pfs: String
      transform_protocol: String
```

OS::Neutron::IPsecSiteConnection

A resource for IPsec site connection in Neutron.

This resource has details for the site-to-site IPsec connection, including the peer CIDRs, MTU, peer address, DPD settings and status.

Required Properties

ikepolicy_idǘ

Unique identifier for the ike policy associated with the ipsec site connection.

String value expected.

Updates cause replacement.

ipsecpolicy_idǘ

Unique identifier for the ipsec policy associated with the ipsec site connection.

String value expected.

Updates cause replacement.

peer_addressǘ

Remote branch router public IPv4 address or IPv6 address or FQDN.

String value expected.

Updates cause replacement.

peer_cidrsǘ

Remote subnet(s) in CIDR format.

List value expected.

Updates cause replacement.

List contents:

Optional.

String value expected.

Updates cause replacement.

Value must be of type net_cidr

peer_id*ű*

Remote branch router identity.

String value expected.

Updates cause replacement.

psk*ű*

Pre-shared key string for the ipsec site connection.

String value expected.

Updates cause replacement.

vpnservice_id*ű*

Unique identifier for the vpn service associated with the ipsec site connection.

String value expected.

Updates cause replacement.

Optional Properties

admin_state_up*ű*

Administrative state for the ipsec site connection.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

description*ű*

Description for the ipsec site connection.

String value expected.

Can be updated without replacement.

dpd*ű*

Dead Peer Detection protocol configuration for the ipsec site connection.

Map value expected.

Updates cause replacement.

Map properties:

actions*ű*

Optional.

Controls DPD protocol mode.

String value expected.

Updates cause replacement.

Defaults to `"hold"`

Allowed values: clear, disabled, hold, restart, restart-by-peer

interval*ű*

Optional.

Number of seconds for the DPD delay.

Integer value expected.

Updates cause replacement.

Defaults to `30`

timeout*ű*

Optional.

Number of seconds for the DPD timeout.

Integer value expected.

Updates cause replacement.

Defaults to `120`

initiator*ű*

Initiator state in lowercase for the ipsec site connection.

String value expected.

Updates cause replacement.

Defaults to `"bi-directional"`

Allowed values: bi-directional, response-only

mtu*ű*

Maximum transmission unit size (in bytes) for the ipsec site connection.

Integer value expected.

Updates cause replacement.

Defaults to `1500`

name*ű*

Name for the ipsec site connection.

String value expected.

Can be updated without replacement.

Attributes

admin_state_up*ű*
The administrative state of the ipsec site connection.

auth_mode*ű*
The authentication mode of the ipsec site connection.

description*ű*
The description of the ipsec site connection.

dpd*ű*
The dead peer detection protocol configuration of the ipsec site connection.

ikepolicy_id*ű*
The unique identifier of ike policy associated with the ipsec site connection.

initiator*ű*
The initiator of the ipsec site connection.

ipsecpolicy_id*ű*
The unique identifier of ipsec policy associated with the ipsec site connection.

mtu*ű*
The maximum transmission unit size (in bytes) of the ipsec site connection.

name*ű*
The name of the ipsec site connection.

peer_address*ű*
The remote branch router public IPv4 address or IPv6 address or FQDN.

peer_cidrs*ű*
The remote subnet(s) in CIDR format of the ipsec site connection.

peer_id*ű*
The remote branch router identity of the ipsec site connection.

psk*ű*
The pre-shared key string of the ipsec site connection.

route_mode*ű*
The route mode of the ipsec site connection.

show*ű*
Detailed information about resource.

status*ű*
The status of the ipsec site connection.

tenant_id*ű*
The unique identifier of the tenant owning the ipsec site connection.

vpnservice_id*ű*
The unique identifier of vpn service associated with the ipsec site connection.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::IPsecSiteConnection
    properties:
      admin_state_up: Boolean
      description: String
      dpd: {"actions": String, "interval": Integer, "timeout": Integer}
      ikepolicy_id: String
      initiator: String
      ipsecpolicy_id: String
      mtu: Integer
      name: String
      peer_address: String
      peer_cidrs: [String, String, ...]
      peer_id: String
      psk: String
      vpnservice_id: String
```

OS::Neutron::L2Gateway

Available since 12.0.0 (Stein)

A resource for managing Neutron L2 Gateways.

The are a number of use cases that can be addressed by an L2 Gateway API. Most notably in cloud computing environments, a typical use case is bridging the virtual with the physical. Translate this to Neutron and the OpenStack world, and this means relying on L2 Gateway capabilities to extend Neutron logical (overlay) networks into physical (provider) networks that are outside the OpenStack realm.

Required Properties

devices*ú*

List of gateway devices.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

device_name*ú*

Required.

The name of the gateway device.

String value expected.

Can be updated without replacement.

interfaces*ű*

List of gateway device interfaces.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

name*ű*

Required.

The name of the interface on the gateway device.

String value expected.

Can be updated without replacement.

segmentation_id*ű*

A list of segmentation ids of the interface.

List value expected.

Can be updated without replacement.

name*ű*

A symbolic name for the l2-gateway, which is not required to be unique.

String value expected.

Can be updated without replacement.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::L2Gateway
    properties:
```

(continues on next page)

(continued from previous page)

```
    devices: [{"device_name": String, "interfaces": [{"name": String,
↪"segmentation_id": [Value, Value, ...]}, {"name": String, "segmentation_id
↪": [Value, Value, ...]}, ...]}, {"device_name": String, "interfaces": [{
↪"name": String, "segmentation_id": [Value, Value, ...]}, {"name": String,
↪"segmentation_id": [Value, Value, ...]}, ...]}, ...]
    name: String
```

OS::Neutron::L2GatewayConnection

Available since 12.0.0 (Stein)

A resource for managing Neutron L2 Gateway Connections.

The L2 Gateway Connection provides a mapping to connect a Neutron network to a L2 Gateway on a particular segmentation ID.

Required Properties

l2_gateway_id*ű*

A string specifying a id of the l2gateway resource.

String value expected.

Updates cause replacement.

network_id*ű*

A string specifying a id of the network resource to connect to the l2gateway.

String value expected.

Updates cause replacement.

Value must be of type neutron.network

Optional Properties

segmentation_id*ű*

A string specifying a segmentation id for the interface on the l2gateway.

String value expected.

Updates cause replacement.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::L2GatewayConnection
    properties:
      l2_gateway_id: String
      network_id: String
      segmentation_id: String
```

OS::Neutron::MeteringLabel

Available since 2014.1 (Icehouse)

A resource for creating neutron metering label.

The idea is to meter this at the L3 routers levels. The point is to allow operators to configure IP ranges and to assign a label to them. For example we will be able to set two labels; one for the internal traffic, and the other one for the external traffic. Each label will measure the traffic for a specific set of IP range. Then, bandwidth measurement will be sent for each label to the Oslo notification system and could be collected by Ceilometer.

Optional Properties

description*ű*

Description of the metering label.

String value expected.

Updates cause replacement.

name*ű*

Name of the metering label.

String value expected.

Updates cause replacement.

shared*ű*

Available since 2015.1 (Kilo)

Whether the metering label should be shared across all tenants.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

Attributes

description*ű*
Description of the metering label.

name*ű*
Name of the metering label.

shared*ű*
Shared status of the metering label.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::MeteringLabel
    properties:
      description: String
      name: String
      shared: Boolean
```

OS::Neutron::MeteringRule

Available since 2014.1 (Icehouse)

A resource to create rule for some label.

Resource for allowing specified label to measure the traffic for a specific set of ip range.

Required Properties

metering_label_id*ű*

The metering label ID to associate with this metering rule.

String value expected.

Updates cause replacement.

remote_ip_prefix*ű*

Indicates remote IP prefix to be associated with this metering rule.

String value expected.

Updates cause replacement.

Optional Properties

direction*ű*

The direction in which metering rule is applied, either ingress or egress.

String value expected.

Updates cause replacement.

Defaults to `"ingress"`

Allowed values: ingress, egress

excluded*ű*

Specify whether the remote_ip_prefix will be excluded or not from traffic counters of the metering label. For example to not count the traffic of a specific IP address of a range.

Boolean value expected.

Updates cause replacement.

Defaults to `"False"`

Attributes

direction*ű*
The direction in which metering rule is applied.

excluded*ű*
Exclude state for cidr.

metering_label_id*ű*
The metering label ID to associate with this metering rule.

remote_ip_prefix*ű*
CIDR to be associated with this metering rule.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::MeteringRule
    properties:
      direction: String
      excluded: Boolean
      metering_label_id: String
      remote_ip_prefix: String
```

OS::Neutron::Net

A resource for managing Neutron net.

A network is a virtual isolated layer-2 broadcast domain which is typically reserved to the tenant who created it, unless the network has been explicitly configured to be shared.

Optional Properties

admin_state_up*ű*

A boolean value specifying the administrative status of the network.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

availability_zone_hints*ű*

Available since 19.0.0 (Zed)

Availability zone candidates for the network. It requires the availability_zone extension to be available.

List value expected.

Updates cause replacement.

dhcp_agent_ids*ű*

The IDs of the DHCP agent to schedule the network. Note that the default policy setting in Neutron restricts usage of this property to administrative users only.

List value expected.

Can be updated without replacement.

dns_domain*ű*

Available since 7.0.0 (Newton)

DNS domain associated with this network.

String value expected.

Can be updated without replacement.

Value must be of type dns_domain

name*ű*

A string specifying a symbolic name for the network, which is not required to be unique.

String value expected.

Can be updated without replacement.

port_security_enabled*ű*

Available since 5.0.0 (Liberty)

Flag to enable/disable port security on the network. It provides the default value for the attribute of the ports created on this network.

Boolean value expected.

Can be updated without replacement.

qos_policy*ű*

Available since 6.0.0 (Mitaka)

The name or ID of QoS policy to attach to this network.

String value expected.

Can be updated without replacement.

Value must be of type neutron.qos_policy

shared*ű*

Whether this network should be shared across all tenants. Note that the default policy setting restricts usage of this attribute to administrative users only.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

tags*ű*

Available since 9.0.0 (Pike)

The tags to be added to the network.

List value expected.

Can be updated without replacement.

List contents:

Optional.

String value expected.

Can be updated without replacement.

tenant_id*ű*

The ID of the tenant which will own the network. Only administrative users can set the tenant identifier; this cannot be changed using authorization policies.

String value expected.

Updates cause replacement.

value_specs*ű*

Extra parameters to include in the request. Parameters are often specific to installed hardware or extensions.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

Attributes

admin_state_up*ű*
The administrative status of the network.

l2_adjacency*ű*

Available since 9.0.0 (Pike)

A boolean value for L2 adjacency, True means that you can expect L2 connectivity throughout the Network.

mtu*ű*

Available since 5.0.0 (Liberty)

The maximum transmission unit size(in bytes) for the network.

name*ű*
The name of the network.

port_security_enabled*ű*

Available since 5.0.0 (Liberty)

Port security enabled of the network.

qos_policy_id*ű*

Available since 6.0.0 (Mitaka)

The QoS policy ID attached to this network.

segments*ű*

Available since 11.0.0 (Rocky)

The segments of this network.

show*ű*
Detailed information about resource.

status*ű*
The status of the network.

subnets*ű*
Subnets of this network.

tenant_id*ű*
The tenant owning this network.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::Net
    properties:
      admin_state_up: Boolean
      availability_zone_hints: [Value, Value, ...]
      dhcp_agent_ids: [Value, Value, ...]
      dns_domain: String
      name: String
      port_security_enabled: Boolean
      qos_policy: String
      shared: Boolean
      tags: [String, String, ...]
      tenant_id: String
      value_specs: {...}
```

OS::Neutron::NetworkGateway

Available since 2014.1 (Icehouse)

Network Gateway resource in Neutron Network Gateway.

Resource for connecting internal networks with specified devices.

Required Properties

devices*ű*

Device info for this network gateway.

List value expected.

Can be updated without replacement.

The length must be at least 1.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

id*ŕí*

Required.

The device id for the network gateway.

String value expected.

Can be updated without replacement.

interface_name*ŕí*

Required.

The interface name for the network gateway.

String value expected.

Can be updated without replacement.

Optional Properties

connections*ŕí*

Connection info for this network gateway.

List value expected.

Can be updated without replacement.

Defaults to `{}`

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

network*ŕí*

Available since 2014.2 (Juno)

Required.

The internal network to connect on the network gateway.

String value expected.

Can be updated without replacement.

Value must be of type neutron.network

segmentation_id*ŕí*

Optional.

The id for L2 segment on the external side of the network gateway. Must be specified when using vlan.

Integer value expected.

Can be updated without replacement.

The value must be in the range 0 to 4094.

segmentation_type*ű*

Optional.

L2 segmentation strategy on the external side of the network gateway.

String value expected.

Can be updated without replacement.

Defaults to `"flat"`

Allowed values: flat, vlan

name*ű*

The name of the network gateway.

String value expected.

Can be updated without replacement.

Attributes

default*ű*
A boolean value of default flag.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::NetworkGateway
    properties:
      connections: [{"network_id": String, "network": String, "segmentation_
↪type": String, "segmentation_id": Integer}, {"network_id": String, "network
↪": String, "segmentation_type": String, "segmentation_id": Integer}, ...]
      devices: [{"id": String, "interface_name": String}, {"id": String,
↪"interface_name": String}, ...]
      name: String
```

OS::Neutron::Port

A resource for managing Neutron ports.

A port represents a virtual switch port on a logical network switch. Virtual instances attach their interfaces into ports. The logical port also defines the MAC address and the IP address(es) to be assigned to the interfaces plugged into them. When IP addresses are associated to a port, this also implies the port is associated with a subnet, as the IP address was taken from the allocation pool for a specific subnet.

Required Properties

network*ú*

Available since 2014.2 (Juno)

Network this port belongs to. If you plan to use current port to assign Floating IP, you should specify fixed_ips with subnet. Note if this changes to a different network update, the port will be replaced.

String value expected.

Updates cause replacement.

Value must be of type neutron.network

Optional Properties

admin_state_up*ú*

The administrative state of this port.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

allowed_address_pairs*ú*

Additional MAC/IP address pairs allowed to pass through the port.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

ip_address*ú*

Required.

IP address to allow through this port.

String value expected.

Can be updated without replacement.

Value must be of type ip_or_cidr

mac_address*ú*

Optional.

MAC address to allow through this port.

String value expected.

Can be updated without replacement.

Value must be of type mac_addr

binding:vnic_type*ű*

Available since 2015.1 (Kilo)

The vnic type to be bound on the neutron port. To support SR-IOV PCI passthrough networking, you can request that the neutron port to be realized as normal (virtual nic), direct (pci passthrough), or macvtap (virtual interface with a tap-like software interface). Note that this only works for Neutron deployments that support the bindings extension.

String value expected.

Can be updated without replacement.

Defaults to `"normal"`

Allowed values: normal, direct, macvtap, direct-physical, baremetal, virtio-forwarder, smart-nic

device_id*ű*

Device ID of this port.

String value expected.

Can be updated without replacement.

Defaults to `""`

device_owner*ű*

Name of the network owning the port. The value is typically network:floatingip or network:router_interface or network:dhcp.

String value expected.

Can be updated without replacement.

Defaults to `""`

dns_name*ű*

Available since 7.0.0 (Newton)

DNS name associated with the port.

String value expected.

Can be updated without replacement.

Value must be of type dns_name

fixed_ips*ű*

Desired IPs for this port.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

ip_address*ű*

Optional.

IP address desired in the subnet for this port.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

subnet*ű*

Available since 2014.2 (Juno)

Optional.

Subnet in which to allocate the IP address for this port.

String value expected.

Can be updated without replacement.

Value must be of type neutron.subnet

mac_address*ű*

MAC address to give to this port. The default update policy of this property in neutron is that allow admin role only.

String value expected.

Updates cause replacement.

Value must be of type mac_addr

name*ű*

A symbolic name for this port.

String value expected.

Can be updated without replacement.

no_fixed_ips*ű*

Available since 16.0.0 (Wallaby)

Flag to disable all fixed ips on the port.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

port_security_enabled*ú*

Available since 5.0.0 (Liberty)

Flag to enable/disable port security on the port. When disable this feature(set it to False), there will be no packages filtering, like security-group and address-pairs.

Boolean value expected.

Can be updated without replacement.

propagate_uplink_status*ú*

Available since 15.0.0 (Victoria)

Flag to enable/disable propagate uplink status on the port.

Boolean value expected.

Can be updated without replacement.

qos_policy*ú*

Available since 6.0.0 (Mitaka)

The name or ID of QoS policy to attach to this port.

String value expected.

Can be updated without replacement.

Value must be of type neutron.qos_policy

security_groups*ú*

Security group IDs to associate with this port.

List value expected.

Can be updated without replacement.

tags*ú*

Available since 9.0.0 (Pike)

The tags to be added to the port.

List value expected.

Can be updated without replacement.

List contents:

Optional.

String value expected.

Can be updated without replacement.

value_specs*ŭ*

Extra parameters to include in the request.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

Attributes

admin_state_up*ŭ*
The administrative state of this port.

allowed_address_pairs*ŭ*
Additional MAC/IP address pairs allowed to pass through a port.

device_id*ŭ*
Unique identifier for the device.

device_owner*ŭ*
Name of the network owning the port.

dns_assignment*ŭ*

Available since 7.0.0 (Newton)

The DNS assigned to this port.

fixed_ips*ŭ*
Fixed IP addresses.

mac_address*ŭ*
MAC address of the port.

name*ŭ*
Friendly name of the port.

network*ŭ*

Available since 11.0.0 (Rocky)

The attributes of the network owning the port. (The full list of response parameters can be found in the 'Openstack Networking service API reference <https://docs.openstack.org/api-ref/network/>'_.) The following examples demonstrate some (not all) possible expressions. (Obtains the network, the MTU (Maximum transmission unit), the network tags and the l2_adjacency property): "{get_attr: [<port>, network]}", "{get_attr: [<port>, network, mtu]}", "{get_attr: [<port>, network, tags]}¿', "{get_attr: [<port>, network, l2_adjacency]}".

network_id*ŭ*
Unique identifier for the network owning the port.

port_security_enabled*ŭ*

Available since 5.0.0 (Liberty)

Port security enabled of the port.

propagate_uplink_status*ű*

Available since 15.0.0 (Victoria)

Enable/Disable propagate uplink status for the port.

qos_policy_id*ű*

Available since 6.0.0 (Mitaka)

The QoS policy ID attached to this port.

security_groups*ű*
A list of security groups for the port.

show*ű*
Detailed information about resource.

status*ű*
The status of the port.

subnets*ű*
A list of all subnet attributes for the port.

tenant_id*ű*
Tenant owning the port.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::Port
    properties:
      admin_state_up: Boolean
      allowed_address_pairs: [{"mac_address": String, "ip_address": String}, {
↪"mac_address": String, "ip_address": String}, ...]
      binding:vnic_type: String
      device_id: String
      device_owner: String
      dns_name: String
      fixed_ips: [{"subnet_id": String, "subnet": String, "ip_address":␣
↪String}, {"subnet_id": String, "subnet": String, "ip_address": String}, ...]
      mac_address: String
      name: String
      network: String
```

(continues on next page)

(continued from previous page)

```
    no_fixed_ips: Boolean
    port_security_enabled: Boolean
    propagate_uplink_status: Boolean
    qos_policy: String
    security_groups: [Value, Value, ...]
    tags: [String, String, ...]
    value_specs: {...}
```

OS::Neutron::ProviderNet

Available since 2014.1 (Icehouse)

A resource for managing Neutron provider networks.

Provider networks specify details of physical realisation of the existing network.

The default policy usage of this resource is limited to administrators only.

Required Properties

network_type*ű*

A string specifying the provider network type for the network.

String value expected.

Can be updated without replacement.

Allowed values: local, vlan, vxlan, gre, geneve, flat

Optional Properties

admin_state_up*ű*

A boolean value specifying the administrative status of the network.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

availability_zone_hints*ű*

Available since 19.0.0 (Zed)

Availability zone candidates for the network. It requires the availability_zone extension to be available.

List value expected.

Can be updated without replacement.

dns_domain*ű*

Available since 15.0.0 (Victoria)

DNS domain associated with this network.

String value expected.

Can be updated without replacement.

Value must be of type dns_domain

name*ŕ*

A string specifying a symbolic name for the network, which is not required to be unique.

String value expected.

Can be updated without replacement.

physical_network*ŕ*

A string specifying physical network mapping for the network.

String value expected.

Can be updated without replacement.

port_security_enabled*ŕ*

Available since 8.0.0 (Ocata)

Flag to enable/disable port security on the network. It provides the default value for the attribute of the ports created on this network.

Boolean value expected.

Can be updated without replacement.

router_external*ŕ*

Available since 6.0.0 (Mitaka)

Whether the network contains an external router.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

segmentation_id*ŕ*

A string specifying the segmentation id for the network.

String value expected.

Can be updated without replacement.

shared*ŕ*

Whether this network should be shared across all tenants.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

tags*ű*

Available since 12.0.0 (Stein)

The tags to be added to the provider network.

List value expected.

Can be updated without replacement.

List contents:

Optional.

String value expected.

Can be updated without replacement.

tenant_id*ű*

Available since 24.0.0

The ID of the tenant which will own the provider network. Only administrative users can set the tenant identifier; this cannot be changed using authorization policies.

String value expected.

Updates cause replacement.

Attributes

segments*ű*

Available since 16.0.0 (Wallaby)

The segments of this network.

show*ű*
Detailed information about resource.

status*ű*
The status of the network.

subnets*ű*
Subnets of this network.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
```

(continues on next page)

(continued from previous page)

```
the_resource:
  type: OS::Neutron::ProviderNet
  properties:
    admin_state_up: Boolean
    availability_zone_hints: [Value, Value, ...]
    dns_domain: String
    name: String
    network_type: String
    physical_network: String
    port_security_enabled: Boolean
    router_external: Boolean
    segmentation_id: String
    shared: Boolean
    tags: [String, String, ...]
    tenant_id: String
```

OS::Neutron::QoSBandwidthLimitRule

Available since 6.0.0 (Mitaka)

A resource for Neutron QoS bandwidth limit rule.

This rule can be associated with QoS policy, and then the policy can be used by neutron port and network, to provide bandwidth limit QoS capabilities.

The default policy usage of this resource is limited to administrators only.

Required Properties

max_kbps*ű*

Max bandwidth in kbps.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

policy*ű*

ID or name of the QoS policy.

String value expected.

Updates cause replacement.

Value must be of type neutron.qos_policy

Optional Properties

direction*ű*

Available since 13.0.0 (Train)

Traffic direction from the point of view of the port.

String value expected.

Can be updated without replacement.

Defaults to `"egress"`

Allowed values: egress, ingress

max_burst_kbps*ű*

Max burst bandwidth in kbps.

Integer value expected.

Can be updated without replacement.

Defaults to `0`

The value must be at least 0.

tenant_id*ű*

The owner tenant ID of this rule.

String value expected.

Updates cause replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::QoSBandwidthLimitRule
    properties:
      direction: String
      max_burst_kbps: Integer
      max_kbps: Integer
      policy: String
      tenant_id: String
```

OS::Neutron::QoSDscpMarkingRule

Available since 7.0.0 (Newton)

A resource for Neutron QoS DSCP marking rule.

This rule can be associated with QoS policy, and then the policy can be used by neutron port and network, to provide DSCP marking QoS capabilities.

The default policy usage of this resource is limited to administrators only.

Required Properties

dscp_mark*ű*

DSCP mark between 0 and 56, except 2-6, 42, 44, and 50-54.

Integer value expected.

Can be updated without replacement.

Allowed values: 0, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 46, 48, 56

policy*ű*

ID or name of the QoS policy.

String value expected.

Updates cause replacement.

Value must be of type neutron.qos_policy

Optional Properties

tenant_id*ű*

The owner tenant ID of this rule.

String value expected.

Updates cause replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::QoSDscpMarkingRule
    properties:
```

(continues on next page)

(continued from previous page)

```
    dscp_mark: Integer
    policy: String
    tenant_id: String
```

OS::Neutron::QoSMinimumBandwidthRule

Available since 14.0.0 (Ussuri)

A resource for guaranteeing bandwidth.

This rule can be associated with a QoS policy, and then the policy can be used by a neutron port to provide guaranteed bandwidth QoS capabilities.

Depending on drivers the guarantee may be enforced on two levels. First when a server is placed (scheduled) on physical infrastructure and/or second in the data plane of the physical hypervisor. For details please see Neutron documentation:

https://docs.openstack.org/neutron/latest/admin/config-qos-min-bw.html

The default policy usage of this resource is limited to administrators only.

Required Properties

min_kbps*ű*

Min bandwidth in kbps.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

policy*ű*

ID or name of the QoS policy.

String value expected.

Updates cause replacement.

Value must be of type neutron.qos_policy

Optional Properties

direction*ű*

Traffic direction from the point of view of the port.

String value expected.

Can be updated without replacement.

Defaults to `"egress"`

Allowed values: egress, ingress

tenant_id*ű*

The owner tenant ID of this rule.

String value expected.

Updates cause replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::QoSMinimumBandwidthRule
    properties:
      direction: String
      min_kbps: Integer
      policy: String
      tenant_id: String
```

OS::Neutron::QoSMinimumPacketRateRule

Available since 19.0.0 (Zed)

A resource for guaranteeing packet rate.

This rule can be associated with a QoS policy, and then the policy can be used by a neutron port to provide guaranteed packet rate QoS capabilities.

Depending on drivers the guarantee may be enforced on two levels. First when a server is placed (scheduled) on physical infrastructure and/or second in the data plane of the physical hypervisor. For details please see Neutron documentation:

https://docs.openstack.org/neutron/latest/admin/config-qos-min-pps.html

The default policy usage of this resource is limited to administrators only.

Required Properties

min_kpps*ű*

Min packet rate in kpps.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

policy*ű*

ID or name of the QoS policy.

String value expected.

Updates cause replacement.

Value must be of type neutron.qos_policy

Optional Properties

direction*ű*

Traffic direction from the point of view of the port.

String value expected.

Can be updated without replacement.

Defaults to `"egress"`

Allowed values: any, egress, ingress

tenant_id*ű*

The owner tenant ID of this rule.

String value expected.

Updates cause replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::QoSMinimumPacketRateRule
    properties:
      direction: String
      min_kpps: Integer
      policy: String
      tenant_id: String
```

OS::Neutron::QoSPolicy

Available since 6.0.0 (Mitaka)

A resource for Neutron QoS Policy.

This QoS policy can be associated with neutron resources, such as port and network, to provide QoS capabilities.

The default policy usage of this resource is limited to administrators only.

Optional Properties

description*ű*

The description for the QoS policy.

String value expected.

Can be updated without replacement.

name*ű*

The name for the QoS policy.

String value expected.

Can be updated without replacement.

shared*ű*

Whether this QoS policy should be shared to other tenants.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

tenant_id*ű*

The owner tenant ID of this QoS policy.

String value expected.

Updates cause replacement.

Attributes

rules*ű*
: A list of all rules for the QoS policy.

show*ű*
: Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::QoSPolicy
    properties:
      description: String
```

(continues on next page)

(continued from previous page)

```
    name: String
    shared: Boolean
    tenant_id: String
```

OS::Neutron::Quota

Available since 8.0.0 (Ocata)

A resource for managing neutron quotas.

Neutron Quota is used to manage operational limits for projects. Currently, this resource can manage Neutrons quotas for:

- subnet
- network
- floatingip
- security_group_rule
- security_group
- router
- port
- subnetpool
- rbac_policy

Note that default neutron security policy usage of this resource is limited to being used by administrators only. Administrators should be careful to create only one Neutron Quota resource per project, otherwise it will be hard for them to manage the quota properly.

Required Properties

project*ű*

Name or id of the project to set the quota for.

String value expected.

Updates cause replacement.

Value must be of type keystone.project

Optional Properties

floatingip*ű*

Quota for the number of floating IPs. Setting -1 means unlimited.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

network*ű*

Quota for the number of networks. Setting -1 means unlimited.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

port*ű*

Quota for the number of ports. Setting -1 means unlimited.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

rbac_policy*ű*

Available since 12.0.0 (Stein)

Quota for the number of rbac policies. Setting -1 means unlimited.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

router*ű*

Quota for the number of routers. Setting -1 means unlimited.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

security_group*ű*

Quota for the number of security groups. Setting -1 means unlimited.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

security_group_rule*ű*

Quota for the number of security group rules. Setting -1 means unlimited.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

subnet*ű*

Quota for the number of subnets. Setting -1 means unlimited.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

subnetpool*ű*

Available since 12.0.0 (Stein)

Quota for the number of subnet pools. Setting -1 means unlimited.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::Quota
    properties:
      floatingip: Integer
      network: Integer
      port: Integer
      project: String
      rbac_policy: Integer
      router: Integer
      security_group: Integer
      security_group_rule: Integer
      subnet: Integer
      subnetpool: Integer
```

OS::Neutron::RBACPolicy

Available since 6.0.0 (Mitaka)

A Resource for managing RBAC policy in Neutron.

This resource creates and manages Neutron RBAC policy, which allows to share Neutron networks and qos-policies to subsets of tenants.

Required Properties

action*ű*

Action for the RBAC policy. The allowed actions differ for different object types - only network objects can have an access_as_external action.

String value expected.

Updates cause replacement.

Allowed values: access_as_shared, access_as_external

object_id*ű*

ID or name of the RBAC object.

String value expected.

Updates cause replacement.

object_type*ű*

Type of the object that RBAC policy affects.

String value expected.

Updates cause replacement.

Allowed values: network, qos_policy

target_tenant*ű*

ID of the tenant to which the RBAC policy will be enforced.

String value expected.

Can be updated without replacement.

Optional Properties

tenant_id*ű*

The owner tenant ID. Only required if the caller has an administrative role and wants to create a RBAC for another tenant.

String value expected.

Updates cause replacement.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
```

(continues on next page)

(continued from previous page)

```
the_resource:
  type: OS::Neutron::RBACPolicy
  properties:
    action: String
    object_id: String
    object_type: String
    target_tenant: String
    tenant_id: String
```

OS::Neutron::Router

A resource that implements Neutron router.

Router is a physical or virtual network device that passes network traffic between different networks.

Optional Properties

admin_state_up*ű*

The administrative state of the router.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

availability_zone_hints*ű*

Available since 19.0.0 (Zed)

Availability zone candidates for the router. It requires the availability_zone extension to be available.

List value expected.

Can be updated without replacement.

distributed*ű*

Available since 2015.1 (Kilo)

Indicates whether or not to create a distributed router. NOTE: The default policy setting in Neutron restricts usage of this property to administrative users only. This property can not be used in conjunction with the L3 agent ID.

Boolean value expected.

Updates cause replacement.

external_gateway_info*ű*

External network gateway configuration for a router.

Map value expected.

Can be updated without replacement.

Map properties:

enable_snat*ű*

Optional.

Enables Source NAT on the router gateway. NOTE: The default policy setting in Neutron restricts usage of this property to administrative users only.

Boolean value expected.

Can be updated without replacement.

external_fixed_ips*ű*

Available since 6.0.0 (Mitaka)

External fixed IP addresses for the gateway.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

ip_address*ű*

Optional.

External fixed IP address.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

subnet*ű*

Optional.

Subnet of external fixed IP address.

String value expected.

Can be updated without replacement.

Value must be of type neutron.subnet

network*ű*

Required.

ID or name of the external network for the gateway.

String value expected.

Can be updated without replacement.

ha*ű*

Available since 2015.1 (Kilo)

Indicates whether or not to create a highly available router. NOTE: The default policy setting in Neutron restricts usage of this property to administrative users only. And now neutron do not support distributed and ha at the same time.

Boolean value expected.

Updates cause replacement.

l3_agent_ids*ű*

Available since 2015.1 (Kilo)

ID list of the L3 agent. User can specify multi-agents for highly available router. NOTE: The default policy setting in Neutron restricts usage of this property to administrative users only.

List value expected.

Can be updated without replacement.

List contents:

Optional.

String value expected.

Can be updated without replacement.

name*ű*

The name of the router.

String value expected.

Can be updated without replacement.

tags*ű*

Available since 9.0.0 (Pike)

The tags to be added to the router.

List value expected.

Can be updated without replacement.

List contents:

Optional.

String value expected.

Can be updated without replacement.

tenant_id*ű*

Available since 24.0.0

The ID of the tenant which will own the router. Only administrative users can set the tenant identifier; this cannot be changed using authorization policies.

String value expected.

Updates cause replacement.

value_specs*ú*

Extra parameters to include in the creation request.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

Attributes

admin_state_up*ú*
Administrative state of the router.

external_gateway_info*ú*
Gateway network for the router.

name*ú*
Friendly name of the router.

show*ú*
Detailed information about resource.

status*ú*
The status of the router.

tenant_id*ú*
Tenant owning the router.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::Router
    properties:
      admin_state_up: Boolean
      availability_zone_hints: [Value, Value, ...]
      distributed: Boolean
      external_gateway_info: {"network": String, "enable_snat": Boolean,
↪"external_fixed_ips": [{"ip_address": String, "subnet": String}, {"ip_
↪address": String, "subnet": String}, ...]}
      ha: Boolean
      l3_agent_ids: [String, String, ...]
```

(continues on next page)

(continued from previous page)

```
    name: String
    tags: [String, String, ...]
    tenant_id: String
    value_specs: {...}
```

OS::Neutron::RouterInterface

A resource for managing Neutron router interfaces.

Router interfaces associate routers with existing subnets or ports.

Required Properties

router*ű*

The router.

String value expected.

Updates cause replacement.

Value must be of type neutron.router

Optional Properties

port*ű*

Available since 2015.1 (Kilo)

The port, either subnet or port should be specified.

String value expected.

Updates cause replacement.

Value must be of type neutron.port

subnet*ű*

The subnet, either subnet or port should be specified.

String value expected.

Updates cause replacement.

Value must be of type neutron.subnet

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::RouterInterface
    properties:
      port: String
      router: String
      subnet: String
```

OS::Neutron::SecurityGroup

Available since 2014.1 (Icehouse)

A resource for managing Neutron security groups.

Security groups are sets of IP filter rules that are applied to an instances networking. They are project specific, and project members can edit the default rules for their group and add new rules sets. All projects have a default security group, which is applied to instances that have no other security group defined.

Optional Properties

description*ǐ*

Description of the security group.

String value expected.

Can be updated without replacement.

name*ǐ*

A string specifying a symbolic name for the security group, which is not required to be unique.

String value expected.

Can be updated without replacement.

rules*ǐ*

List of security group rules.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

direction*ű*

Optional.

The direction in which the security group rule is applied. For a compute instance, an ingress security group rule matches traffic that is incoming (ingress) for that instance. An egress rule is applied to traffic leaving the instance.

String value expected.

Can be updated without replacement.

Defaults to `"ingress"`

Allowed values: ingress, egress

ethertype*ű*

Optional.

Ethertype of the traffic.

String value expected.

Can be updated without replacement.

Defaults to `"IPv4"`

Allowed values: IPv4, IPv6

port_range_max*ű*

Optional.

The maximum port number in the range that is matched by the security group rule. The port_range_min attribute constrains the port_range_max attribute. If the protocol is ICMP, this value must be an ICMP type.

Integer value expected.

Can be updated without replacement.

The value must be in the range 0 to 65535.

port_range_min*ű*

Optional.

The minimum port number in the range that is matched by the security group rule. If the protocol is TCP or UDP, this value must be less than or equal to the value of the port_range_max attribute. If the protocol is ICMP, this value must be an ICMP type.

Integer value expected.

Can be updated without replacement.

The value must be in the range 0 to 65535.

protocol*ű*

Optional.

The protocol that is matched by the security group rule. Valid values include tcp, udp, and icmp.

String value expected.

Can be updated without replacement.

remote_group_id*ű*

Optional.

The remote group ID to be associated with this security group rule. If no value is specified then this rule will use this security group for the remote_group_id. The remote mode parameter must be set to remote_group_id.

String value expected.

Can be updated without replacement.

Value must be of type neutron.security_group

remote_ip_prefix*ű*

Optional.

The remote IP prefix (CIDR) to be associated with this security group rule.

String value expected.

Can be updated without replacement.

Value must be of type net_cidr

remote_mode*ű*

Optional.

Whether to specify a remote group or a remote IP prefix.

String value expected.

Can be updated without replacement.

Defaults to `"remote_ip_prefix"`

Allowed values: remote_ip_prefix, remote_group_id

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::SecurityGroup
    properties:
      description: String
      name: String
      rules: [{"direction": String, "ethertype": String, "port_range_min":␣
↪Integer, "port_range_max": Integer, "protocol": String, "remote_mode":␣
```

(continues on next page)

(continued from previous page)

```
↪String, "remote_group_id": String, "remote_ip_prefix": String}, {"direction
↪": String, "ethertype": String, "port_range_min": Integer, "port_range_max
↪": Integer, "protocol": String, "remote_mode": String, "remote_group_id":␣
↪String, "remote_ip_prefix": String}, ...]
```

OS::Neutron::SecurityGroupRule

Available since 7.0.0 (Newton)

A resource for managing Neutron security group rules.

Rules to use in security group resource.

Required Properties

security_group*ií*

Security group name or ID to add rule.

String value expected.

Updates cause replacement.

Value must be of type neutron.security_group

Optional Properties

description*ií*

Description of the security group rule.

String value expected.

Updates cause replacement.

direction*ií*

The direction in which the security group rule is applied. For a compute instance, an ingress security group rule matches traffic that is incoming (ingress) for that instance. An egress rule is applied to traffic leaving the instance.

String value expected.

Updates cause replacement.

Defaults to `"ingress"`

Allowed values: ingress, egress

ethertype*ií*

Ethertype of the traffic.

String value expected.

Updates cause replacement.

Defaults to `"IPv4"`

Allowed values: IPv4, IPv6

port_range_max*ű*

The maximum port number in the range that is matched by the security group rule. The port_range_min attribute constrains the port_range_max attribute. If the protocol is ICMP, this value must be an ICMP code.

Integer value expected.

Updates cause replacement.

The value must be in the range 0 to 65535.

port_range_min*ű*

The minimum port number in the range that is matched by the security group rule. If the protocol is TCP or UDP, this value must be less than or equal to the value of the port_range_max attribute. If the protocol is ICMP, this value must be an ICMP type.

Integer value expected.

Updates cause replacement.

The value must be in the range 0 to 65535.

protocol*ű*

The protocol that is matched by the security group rule. Allowed values are ah, dccp, egp, esp, gre, icmp, icmpv6, igmp, ipv6-encap, ipv6-frag, ipv6-icmp, ipv6-nonxt, ipv6-opts, ipv6-route, ospf, pgm, rsvp, sctp, tcp, udp, udplite, vrrp and integer representations [0-255].

String value expected.

Updates cause replacement.

Defaults to `"tcp"`

Allowed values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, ah, dccp, egp, esp, gre, icmp, icmpv6, igmp, ipv6-encap, ipv6-frag, ipv6-icmp, ipv6-nonxt, ipv6-opts, ipv6-route, ospf, pgm, rsvp, sctp, tcp, udp, udplite, vrrp

remote_group*ű*

The remote group name or ID to be associated with this security group rule.

String value expected.

Updates cause replacement.

Value must be of type neutron.security_group

remote_ip_prefixűű

The remote IP prefix (CIDR) to be associated with this security group rule.

String value expected.

Updates cause replacement.

Value must be of type net_cidr

Attributes

showűű
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::SecurityGroupRule
    properties:
      description: String
      direction: String
      ethertype: String
      port_range_max: Integer
      port_range_min: Integer
      protocol: String
      remote_group: String
      remote_ip_prefix: String
      security_group: String
```

OS::Neutron::Segment

Available since 9.0.0 (Pike)

A resource for Neutron Segment.

This requires enabling the segments service plug-in by appending segments to the list of service_plugins in the neutron.conf.

The default policy usage of this resource is limited to administrators only.

Required Properties

networkűű

The name/id of network to associate with this segment.

String value expected.

Updates cause replacement.

Value must be of type neutron.network

network_type*ű*

Type of network to associate with this segment.

String value expected.

Updates cause replacement.

Allowed values: local, vlan, vxlan, gre, geneve, flat

Optional Properties

description*ű*

Description of the segment.

String value expected.

Can be updated without replacement.

name*ű*

Name of the segment.

String value expected.

Can be updated without replacement.

physical_network*ű*

Name of physical network to associate with this segment.

String value expected.

Updates cause replacement.

segmentation_id*ű*

Segmentation ID for this segment.

Integer value expected.

Updates cause replacement.

The value must be at least 1.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::Segment
```

(continues on next page)

(continued from previous page)

```
    properties:
      description: String
      name: String
      network: String
      network_type: String
      physical_network: String
      segmentation_id: Integer
```

OS::Neutron::Subnet

A resource for managing Neutron subnets.

A subnet represents an IP address block that can be used for assigning IP addresses to virtual instances. Each subnet must have a CIDR and must be associated with a network. IPs can be either selected from the whole subnet CIDR, or from allocation pools that can be specified by the user.

Required Properties

network*ű*

Available since 2014.2 (Juno)

The ID of the attached network.

String value expected.

Updates cause replacement.

Value must be of type neutron.network

Optional Properties

allocation_pools*ű*

The start and end addresses for the allocation pools.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

end*ű*

Required.

End address for the allocation pool.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

start*ű*

Required.

Start address for the allocation pool.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

cidr*ű*

The CIDR.

String value expected.

Updates cause replacement.

Value must be of type net_cidr

dns_nameservers*ű*

A specified set of DNS name servers to be used.

List value expected.

Can be updated without replacement.

Defaults to `[]`

enable_dhcp*ű*

Set to true if DHCP is enabled and false if DHCP is disabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

gateway_ip*ű*

The gateway IP address. Set to any of [null | ~ |] to create/update a subnet without a gateway. If omitted when creation, neutron will assign the first free IP address within the subnet to the gateway automatically. If remove this from template when update, the old gateway IP address will be detached.

String value expected.

Can be updated without replacement.

host_routes*ű*

A list of host route dictionaries for the subnet.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

destination*ű*

Required.

The destination for static route.

String value expected.

Can be updated without replacement.

Value must be of type net_cidr

nexthop*ű*

Required.

The next hop for the destination.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

ip_version*ű*

The IP version, which is 4 or 6.

Integer value expected.

Updates cause replacement.

Defaults to 4

Allowed values: 4, 6

ipv6_address_mode*ű*

Available since 2015.1 (Kilo)

IPv6 address mode.

String value expected.

Updates cause replacement.

Allowed values: dhcpv6-stateful, dhcpv6-stateless, slaac

ipv6_ra_mode*ű*

Available since 2015.1 (Kilo)

IPv6 RA (Router Advertisement) mode.

String value expected.

Updates cause replacement.

Allowed values: dhcpv6-stateful, dhcpv6-stateless, slaac

name*ű*

The name of the subnet.

String value expected.

Can be updated without replacement.

prefixlen*ű*

Available since 6.0.0 (Mitaka)

Prefix length for subnet allocation from subnet pool.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

segment*ű*

Available since 11.0.0 (Rocky) - Update allowed since version 11.0.0.

Available since 9.0.0 (Pike)

The name/ID of the segment to associate.

String value expected.

Can be updated without replacement.

Value must be of type neutron.segment

subnetpool*ű*

Available since 6.0.0 (Mitaka)

The name or ID of the subnet pool.

String value expected.

Updates cause replacement.

Value must be of type neutron.subnetpool

tags*ű*

Available since 9.0.0 (Pike)

The tags to be added to the subnet.

List value expected.

Can be updated without replacement.

List contents:

Optional.

String value expected.

Can be updated without replacement.

tenant_id*ű*

The ID of the tenant who owns the network. Only administrative users can specify a tenant ID other than their own.

String value expected.

Updates cause replacement.

value_specs*ű*

Extra parameters to include in the request.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

Attributes

allocation_pools*ű*
Ip allocation pools and their ranges.

cidr*ű*
CIDR block notation for this subnet.

dns_nameservers*ű*
List of dns nameservers.

enable_dhcp*ű*
true if DHCP is enabled for this subnet; false otherwise.

gateway_ip*ű*
Ip of the subnets gateway.

host_routes*ű*
Additional routes for this subnet.

ip_version*ű*
Ip version for the subnet.

name*ű*
Friendly name of the subnet.

network_id*ű*
Parent network of the subnet.

show*ű*
Detailed information about resource.

tenant_id*ű*
Tenant owning the subnet.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::Subnet
    properties:
      allocation_pools: [{"start": String, "end": String}, {"start": String,
↪"end": String}, ...]
      cidr: String
      dns_nameservers: [Value, Value, ...]
      enable_dhcp: Boolean
      gateway_ip: String
      host_routes: [{"destination": String, "nexthop": String}, {"destination
↪": String, "nexthop": String}, ...]
      ip_version: Integer
      ipv6_address_mode: String
      ipv6_ra_mode: String
      name: String
      network: String
      prefixlen: Integer
      segment: String
      subnetpool: String
      tags: [String, String, ...]
      tenant_id: String
      value_specs: {...}
```

OS::Neutron::SubnetPool

Available since 6.0.0 (Mitaka)

A resource that implements neutron subnet pool.

This resource can be used to create a subnet pool with a large block of addresses and create subnets from it.

Required Properties

prefixes*ű*

List of subnet prefixes to assign.

List value expected.

Can be updated without replacement.

The length must be at least 1.

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type net_cidr

Optional Properties

address_scope*ú*

An address scope ID to assign to the subnet pool.

String value expected.

Can be updated without replacement.

Value must be of type neutron.address_scope

default_prefixlen*ú*

The size of the prefix to allocate when the cidr or prefixlen attributes are not specified while creating a subnet.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

default_quota*ú*

A per-tenant quota on the prefix space that can be allocated from the subnet pool for tenant subnets.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

is_default*ú*

Whether this is default IPv4/IPv6 subnet pool. There can only be one default subnet pool for each IP family. Note that the default policy setting restricts administrative users to set this to True.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

max_prefixlen*ú*

Maximum prefix size that can be allocated from the subnet pool.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

min_prefixlen*ú*

Smallest prefix size that can be allocated from the subnet pool.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

name*ű*

Name of the subnet pool.

String value expected.

Can be updated without replacement.

shared*ű*

Whether the subnet pool will be shared across all tenants. Note that the default policy setting restricts usage of this attribute to administrative users only.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

tags*ű*

> Available since 9.0.0 (Pike)

The tags to be added to the subnetpool.

List value expected.

Can be updated without replacement.

List contents:

> Optional.
>
> String value expected.
>
> Can be updated without replacement.

tenant_id*ű*

The ID of the tenant who owns the subnet pool. Only administrative users can specify a tenant ID other than their own.

String value expected.

Updates cause replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
```

(continues on next page)

(continued from previous page)

```
...
the_resource:
  type: OS::Neutron::SubnetPool
  properties:
    address_scope: String
    default_prefixlen: Integer
    default_quota: Integer
    is_default: Boolean
    max_prefixlen: Integer
    min_prefixlen: Integer
    name: String
    prefixes: [String, String, ...]
    shared: Boolean
    tags: [String, String, ...]
    tenant_id: String
```

OS::Neutron::TaaS::TapFlow

Available since 12.0.0 (Stein)

A resource for neutron tap-as-a-service tap-flow.

This plug-in requires neutron-taas. So to enable this plug-in, install this library and restart the heat-engine.

A Tap-Flow represents the port from which the traffic needs to be mirrored.

Required Properties

port*ű*

ID or name of the tap-flow neutron port.

String value expected.

Updates cause replacement.

Value must be of type neutron.port

tap_service*ű*

ID or name of the neutron tap-service.

String value expected.

Updates cause replacement.

Value must be of type neutron.taas.tap_service

Optional Properties

description*ű*

Description for the Tap-Flow.

String value expected.

Can be updated without replacement.

Defaults to ""

direction*ű*

The Direction to capture the traffic on.

String value expected.

Updates cause replacement.

Defaults to "BOTH"

Allowed values: IN, OUT, BOTH

name*ű*

Name for the Tap-Flow.

String value expected.

Can be updated without replacement.

Defaults to ""

vlan_filter*ű*

Comma separated list of VLANs, data for which needs to be captured on probe VM.

String value expected.

Updates cause replacement.

Value must match pattern: ^([0-9]+(-[0-9]+)?)(,([0-9]+(-[0-9]+)?))*$

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::TaaS::TapFlow
    properties:
      description: String
      direction: String
      name: String
```

(continues on next page)

(continued from previous page)

```
    port: String
    tap_service: String
    vlan_filter: String
```

OS::Neutron::TaaS::TapService

Available since 12.0.0 (Stein)

A resource for neutron tap-as-a-service tap-service.

This plug-in requires neutron-taas. So to enable this plug-in, install this library and restart the heat-engine.

A Tap-Service represents the port on which the mirrored traffic is delivered. Any VM that uses the mirrored data is attached to this port.

Required Properties

port*ű*

ID or name of the tap-service neutron port.

String value expected.

Updates cause replacement.

Value must be of type neutron.port

Optional Properties

description*ű*

Description for the Tap-Service.

String value expected.

Can be updated without replacement.

Defaults to ""

name*ű*

Name for the Tap-Service.

String value expected.

Can be updated without replacement.

Defaults to ""

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::TaaS::TapService
    properties:
      description: String
      name: String
      port: String
```

OS::Neutron::Trunk

Available since 9.0.0 (Pike)

A resource for managing Neutron trunks.

Requires Neutron Trunk Extension to be enabled:

```
$ openstack extension show trunk
```

The network trunk service allows multiple networks to be connected to an instance using a single virtual NIC (vNIC). Multiple networks can be presented to an instance by connecting the instance to a single port.

Users can create a port, associate it with a trunk (as the trunks parent) and launch an instance on that port. Users can dynamically attach and detach additional networks without disrupting operation of the instance.

Every trunk has a parent port and can have any number (0, 1,) of subports. The parent port is the port that the instance is directly associated with and its traffic is always untagged inside the instance. Users must specify the parent port of the trunk when launching an instance attached to a trunk.

A network presented by a subport is the network of the associated port. When creating a subport, a `segmentation_type` and `segmentation_id` may be required by the driver so the user can distinguish the networks inside the instance. As of release Pike only `segmentation_type vlan` is supported. `segmentation_id` defines the segmentation ID on which the subport network is presented to the instance.

Note that some Neutron backends (primarily Open vSwitch) only allow trunk creation before an instance is booted on the parent port. To avoid a possible race condition when booting an instance with a trunk it is strongly recommended to refer to the trunks parent port indirectly in the template via `get_attr`. For example:

```
trunk:
  type: OS::Neutron::Trunk
  properties:
    port: ...
instance:
  type: OS::Nova::Server
```

(continues on next page)

(continued from previous page)

```
  properties:
    networks:
      - { port: { get_attr: [trunk, port_id] } }
```

Though other Neutron backends may tolerate the direct port reference (and the possible reverse ordering of API requests implied) its a good idea to avoid writing Neutron backend specific templates.

Required Properties

port*ű*

ID or name of a port to be used as a parent port.

String value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

Value must be of type neutron.port

Optional Properties

admin_state_up*ű*

Enable/disable subport addition, removal and trunk delete.

Boolean value expected.

Can be updated without replacement.

description*ű*

Description for the trunk.

String value expected.

Can be updated without replacement.

name*ű*

A string specifying a symbolic name for the trunk, which is not required to be uniqe.

String value expected.

Can be updated without replacement.

sub_ports*ű*

List with 0 or more map elements containing subport details.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

port*ű*

Required.

ID or name of a port to be used as a subport.

String value expected.

Can be updated without replacement.

Value must be of type neutron.port

segmentation_id*ű*

Required.

The segmentation ID on which the subport network is presented to the instance.

Integer value expected.

Can be updated without replacement.

The value must be in the range 1 to 4094.

segmentation_type*ű*

Required.

Segmentation type to be used on the subport.

String value expected.

Can be updated without replacement.

Allowed values: vlan

Attributes

port_id*ű*
ID or name of a port used as a parent port.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::Trunk
    properties:
      admin_state_up: Boolean
      description: String
      name: String
      port: String
      sub_ports: [{"port": String, "segmentation_type": String, "segmentation_
↪id": Integer}, {"port": String, "segmentation_type": String, "segmentation_
↪id": Integer}, ...]
```

OS::Neutron::VPNService

A resource for VPN service in Neutron.

VPN service is a high level object that associates VPN with a specific subnet and router.

Required Properties

router*ű*

Available since 2015.1 (Kilo)

The router to which the vpn service will be inserted.

String value expected.

Updates cause replacement.

Value must be of type neutron.router

subnet*ű*

Available since 2014.2 (Juno)

Subnet in which the vpn service will be created.

String value expected.

Updates cause replacement.

Value must be of type neutron.subnet

Optional Properties

admin_state_up*ű*

Administrative state for the vpn service.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

description*ű*

Description for the vpn service.

String value expected.

Can be updated without replacement.

name*ű*

Name for the vpn service.

String value expected.

Can be updated without replacement.

Attributes

admin_state_up*ű*
The administrative state of the vpn service.

description*ű*
The description of the vpn service.

name*ű*
The name of the vpn service.

router_id*ű*
The unique identifier of the router to which the vpn service was inserted.

show*ű*
Detailed information about resource.

status*ű*
The status of the vpn service.

subnet_id*ű*
The unique identifier of the subnet in which the vpn service was created.

tenant_id*ű*
The unique identifier of the tenant owning the vpn service.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::VPNService
    properties:
      admin_state_up: Boolean
      description: String
      name: String
      router: String
      subnet: String
```

OS::Nova::Flavor

Available since 2014.2 (Juno)

A resource for creating OpenStack virtual hardware templates.

Due to default nova security policy usage of this resource is limited to being used by administrators only. The rights may also be delegated to other users by redefining the access controls on the nova-api server.

Note that the current implementation of the Nova Flavor resource does not allow specifying the name and flavorid properties for the resource. This is done to avoid potential naming collision upon flavor creation as all flavor have a global scope.

Required Properties

ram*ú*

Memory in MB for the flavor.

Integer value expected.

Updates cause replacement.

vcpus*ú*

Number of VCPUs for the flavor.

Integer value expected.

Updates cause replacement.

Optional Properties

disk*ú*

Size of local disk in GB. The 0 size is a special case that uses the native base image size as the size of the ephemeral root volume.

Integer value expected.

Updates cause replacement.

Defaults to `0`

ephemeral*ú*

Size of a secondary ephemeral data disk in GB.

Integer value expected.

Updates cause replacement.

Defaults to `0`

extra_specs*ú*

Key/Value pairs to extend the capabilities of the flavor.

Map value expected.

Can be updated without replacement.

flavorid*ú*

Available since 7.0.0 (Newton)

Unique ID of the flavor. If not specified, an UUID will be auto generated and used.

String value expected.

Updates cause replacement.

is_public*ú*

Available since 6.0.0 (Mitaka)

Scope of flavor accessibility. Public or private. Default value is True, means public, shared across all projects.

Boolean value expected.

Updates cause replacement.

Defaults to `true`

name*ű*

Available since 7.0.0 (Newton)

Name of the flavor.

String value expected.

Updates cause replacement.

rxtx_factor*ű*

RX/TX factor.

Number value expected.

Updates cause replacement.

Defaults to `1.0`

swap*ű*

Swap space in MB.

Integer value expected.

Updates cause replacement.

Defaults to `0`

tenants*ű*

Available since 8.0.0 (Ocata)

List of tenants.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type keystone.project

Attributes

extra_specs*ű*

Available since 7.0.0 (Newton)

Extra specs of the flavor in key-value pairs.

is_public*ű*

Available since 6.0.0 (Mitaka)

Whether the flavor is shared across all projects.

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Nova::Flavor
    properties:
      disk: Integer
      ephemeral: Integer
      extra_specs: {...}
      flavorid: String
      is_public: Boolean
      name: String
      ram: Integer
      rxtx_factor: Number
      swap: Integer
      tenants: [String, String, ...]
      vcpus: Integer
```

OS::Nova::HostAggregate

Available since 6.0.0 (Mitaka)

A resource for further partition an availability zone with hosts.

While availability zones are visible to users, host aggregates are only visible to administrators. Host aggregates started out as a way to use Xen hypervisor resource pools, but has been generalized to provide a mechanism to allow administrators to assign key-value pairs to groups of machines. Each node can have multiple aggregates, each aggregate can have multiple key-value pairs, and the same key-value pair can be assigned to multiple aggregate. This information can be used in the scheduler to enable advanced scheduling, to set up xen hypervisor resources pools or to define logical groups for migration.

Required Properties

name*ű*

Name for the aggregate.

String value expected.

Can be updated without replacement.

Optional Properties

availability_zone*ű*

Name for the availability zone.

String value expected.

Can be updated without replacement.

hosts*ű*

List of hosts to join aggregate.

List value expected.

Can be updated without replacement.

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type nova.host

metadata*ű*

Arbitrary key/value metadata to store information for aggregate.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Nova::HostAggregate
```

(continues on next page)

(continued from previous page)

```
  properties:
    availability_zone: String
    hosts: [String, String, ...]
    metadata: {...}
    name: String
```

OS::Nova::KeyPair

Available since 2014.1 (Icehouse)

A resource for creating Nova key pairs.

A keypair is a ssh key that can be injected into a server on launch.

Note that if a new key is generated setting *save_private_key* to *True* results in the system saving the private key which can then be retrieved via the *private_key* attribute of this resource.

Setting the *public_key* property means that the *private_key* attribute of this resource will always return an empty string regardless of the *save_private_key* setting since there will be no private key data to save.

Required Properties

name*ű*

The name of the key pair.

String value expected.

Updates cause replacement.

The length must be in the range 1 to 255.

Optional Properties

public_key*ű*

The public key. This allows users to supply the public key from a pre-existing key pair. In Nova api version < 2.92, if not supplied, a new key pair will be generated. This property is required since Nova api version 2.92.

String value expected.

Updates cause replacement.

save_private_key*ű*

True if the system should remember a generated private key; False otherwise.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

type*ű*

Available since 8.0.0 (Ocata)

Keypair type. Supported since Nova api version 2.2.

String value expected.

Updates cause replacement.

Allowed values: ssh, x509

user*ű*

Available since 9.0.0 (Pike)

ID or name of user to whom to add key-pair. The usage of this property is limited to being used by administrators only. Supported since Nova api version 2.10.

String value expected.

Updates cause replacement.

Value must be of type keystone.user

Attributes

private_key*ű*
The private key if it has been saved.

public_key*ű*
The public key.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Nova::KeyPair
    properties:
      name: String
      public_key: String
      save_private_key: Boolean
      type: String
      user: String
```

OS::Nova::Quota

Available since 8.0.0 (Ocata)

A resource for creating nova quotas.

Nova Quota is used to manage operational limits for projects. Currently, this resource can manage Novas quotas for:

- cores
- fixed_ips
- floating_ips
- instances
- injected_files
- injected_file_content_bytes
- injected_file_path_bytes
- key_pairs
- metadata_items
- ram
- security_groups
- security_group_rules
- server_groups
- server_group_members

Note that default nova security policy usage of this resource is limited to being used by administrators only. Administrators should be careful to create only one Nova Quota resource per project, otherwise it will be hard for them to manage the quota properly.

Required Properties

project*ú*

Name or id of the project to set the quota for.

String value expected.

Updates cause replacement.

Value must be of type keystone.project

Optional Properties

cores*ú*

Quota for the number of cores. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

fixed_ips*ű*

Quota for the number of fixed IPs. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

floating_ips*ű*

Quota for the number of floating IPs. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

instances*ű*

Quota for the number of instances. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

key_pairs*ű*

Quota for the number of key pairs. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

metadata_items*ű*

Quota for the number of metadata items. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

ram*ű*

Quota for the amount of ram (in megabytes). Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

security_group_rules*ű*

Quota for the number of security group rules. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

security_groups*ú*

Quota for the number of security groups. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

server_group_members*ú*

Quota for the number of server group members. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

server_groups*ú*

Quota for the number of server groups. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

injected_file_content_bytes*ú*

> DEPRECATED since 14.0.0 (Ussuri) - File injection is deprecated from compute REST API OS::Nova::Quota resource will not support it in the future.

Quota for the number of injected file content bytes. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

injected_file_path_bytes*ú*

> DEPRECATED since 14.0.0 (Ussuri) - File injection is deprecated from compute REST API OS::Nova::Quota resource will not support it in the future.

Quota for the number of injected file path bytes. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

injected_files*ú*

> DEPRECATED since 14.0.0 (Ussuri) - File injection is deprecated from compute REST API OS::Nova::Quota resource will not support it in the future.

Quota for the number of injected files. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Nova::Quota
    properties:
      cores: Integer
      fixed_ips: Integer
      floating_ips: Integer
      instances: Integer
      key_pairs: Integer
      metadata_items: Integer
      project: String
      ram: Integer
      security_group_rules: Integer
      security_groups: Integer
      server_group_members: Integer
      server_groups: Integer
```

OS::Nova::Server

A resource for managing Nova instances.

A Server resource manages the running virtual machine instance within an OpenStack cloud.

Required Properties

flavor*ű*

The ID or name of the flavor to boot onto.

String value expected.

Can be updated without replacement.

Value must be of type nova.flavor

Optional Properties

admin_pass*ű*

The administrator password for the server.

String value expected.

Can be updated without replacement.

availability_zone*ű*

Name of the availability zone for server placement.

String value expected.

Updates cause replacement.

block_device_mapping*ű*

Block device mappings for this server.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

delete_on_termination*ű*

Optional.

Indicate whether the volume should be deleted when the server is terminated.

Boolean value expected.

Updates cause replacement.

device_name*ű*

Required.

A device name where the volume will be attached in the system at /dev/device_name. This value is typically vda.

String value expected.

Updates cause replacement.

snapshot_id*ű*

Optional.

The ID of the snapshot to create a volume from.

String value expected.

Updates cause replacement.

Value must be of type cinder.snapshot

volume_id*ű*

Optional.

The ID of the volume to boot from. Only one of volume_id or snapshot_id should be provided.

String value expected.

Updates cause replacement.

Value must be of type cinder.volume

volume_size*ű*

Optional.

The size of the volume, in GB. It is safe to leave this blank and have the Compute service infer the size.

Integer value expected.

Updates cause replacement.

block_device_mapping_v2*ű*

Available since 2015.1 (Kilo)

Block device mappings v2 for this server.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

boot_index*ű*

Optional.

Integer used for ordering the boot disks. If it is not specified, value 0 will be set for bootable sources (volume, snapshot, image); value -1 will be set for non-bootable sources.

Integer value expected.

Updates cause replacement.

delete_on_termination*ű*

Optional.

Indicate whether the volume should be deleted when the server is terminated. Defaults to False in case of a volume, snapshot or image and to True in case of swap or ephemeral.

Boolean value expected.

Updates cause replacement.

device_name*ű*

Optional.

A device name where the volume will be attached in the system at /dev/device_name. This value is typically vda.

String value expected.

Updates cause replacement.

device_type*ű*

Optional.

Device type: at the moment we can make distinction only between disk and cdrom.

String value expected.

Updates cause replacement.

Allowed values: cdrom, disk

disk_bus*ű*

Optional.

Bus of the device: hypervisor driver chooses a suitable default if omitted.

String value expected.

Updates cause replacement.

Allowed values: ide, lame_bus, sata, scsi, usb, virtio

ephemeral_format*ű*

Available since 8.0.0 (Ocata)

Optional.

The format of the local ephemeral block device. If no format is specified, uses default value, defined in nova configuration file.

String value expected.

Updates cause replacement.

Allowed values: ext2, ext3, ext4, xfs, ntfs

ephemeral_size*ű*

Available since 8.0.0 (Ocata)

Optional.

The size of the local ephemeral block device, in GB.

Integer value expected.

Updates cause replacement.

The value must be at least 1.

image*ű*

Available since 7.0.0 (Newton)

Optional.

The ID or name of the image to create a volume from.

String value expected.

Updates cause replacement.

Value must be of type glance.image

snapshot_id*ű*

Optional.

The ID of the snapshot to create a volume from.

String value expected.

Updates cause replacement.

Value must be of type cinder.snapshot

swap_size*ű*

Optional.

The size of the swap, in MB.

Integer value expected.

Updates cause replacement.

volume_id*ű*

Optional.

The volume_id can be boot or non-boot device to the server.

String value expected.

Updates cause replacement.

Value must be of type cinder.volume

volume_size*ű*

Optional.

Size of the block device in GB. If it is omitted, hypervisor driver calculates size.

Integer value expected.

Updates cause replacement.

config_drive*ű*

If True, enable config drive on the server.

Boolean value expected.

Updates cause replacement.

deployment_swift_data*ű*

Available since 9.0.0 (Pike)

Swift container and object to use for storing deployment data for the server resource. The parameter is a map value with the keys container and object, and the values are the corresponding container and object names. The software_config_transport parameter must be set to POLL_TEMP_URL for swift to be used. If not specified, and software_config_transport is set to POLL_TEMP_URL, a container will be automatically created from the resource name, and the object name will be a generated uuid.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

Map properties:

container*ű*

Optional.

Name of the container.

String value expected.

Can be updated without replacement.

The length must be at least 1.

object*ű*

Optional.

Name of the object.

String value expected.

Can be updated without replacement.

The length must be at least 1.

diskConfig*ű*

Control how the disk is partitioned when the server is created.

String value expected.

Updates cause replacement.

Allowed values: AUTO, MANUAL

flavor_update_policy*ű*

Policy on how to apply a flavor update; either by requesting a server resize or by replacing the entire server.

String value expected.

Can be updated without replacement.

Defaults to `"RESIZE"`

Allowed values: RESIZE, REPLACE

image*ű*

The ID or name of the image to boot with.

String value expected.

Can be updated without replacement.

Value must be of type glance.image

image_update_policy*ű*

Policy on how to apply an image-id update; either by requesting a server rebuild or by replacing the entire server.

String value expected.

Can be updated without replacement.

Defaults to `"REBUILD"`

Allowed values: REBUILD, REPLACE, REBUILD_PRESERVE_EPHEMERAL

key_name*ű*

Name of keypair to inject into the server.

String value expected.

Updates cause replacement.

Value must be of type nova.keypair

metadata*ű*

Arbitrary key/value metadata to store for this server. Both keys and values must be 255 characters or less. Non-string values will be serialized to JSON (and the serialized string must be 255 characters or less).

Map value expected.

Can be updated without replacement.

Defaults to `{}`

name*ű*

Server name.

String value expected.

Can be updated without replacement.

networks*ű*

An ordered list of nics to be added to this server, with information about connected networks, fixed ips, port etc.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

allocate_network*ű*

Available since 9.0.0 (Pike)

Optional.

The special string values of network, auto: means either a network that is already available to the project will be used, or if one does not exist, will be automatically created for the project; none: means no networking will be allocated for the created server. Supported by Nova API since version 2.37. This property can not be used with other network keys.

String value expected.

Can be updated without replacement.

Allowed values: none, auto

fixed_ip*ű*

Optional.

Fixed IP address to specify for the port created on the requested network.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

floating_ip*ű*

Available since 6.0.0 (Mitaka)

Optional.

ID of the floating IP to associate.

String value expected.

Can be updated without replacement.

network*ű*

Optional.

Name or ID of network to create a port on.

String value expected.

Can be updated without replacement.

Value must be of type neutron.network

port*ű*

Optional.

ID of an existing port to associate with this server.

String value expected.

Can be updated without replacement.

Value must be of type neutron.port

port_extra_properties*ű*

Available since 6.0.0 (Mitaka)

Dict, which has expand properties for port. Used only if port property is not specified for creating port.

Map value expected.

Can be updated without replacement.

Map properties:

admin_state_up*ű*

Optional.

The administrative state of this port.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

allowed_address_pairs*ű*

Additional MAC/IP address pairs allowed to pass through the port.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

ip_address*ű*

Required.

IP address to allow through this port.

String value expected.

Can be updated without replacement.

Value must be of type ip_or_cidr

mac_address*ű*

Optional.

MAC address to allow through this port.

String value expected.

Can be updated without replacement.

Value must be of type mac_addr

binding:vnic_type*ű*

Available since 2015.1 (Kilo)

Optional.

The vnic type to be bound on the neutron port. To support SR-IOV PCI passthrough networking, you can request that the neutron port to be realized as normal (virtual nic), direct (pci passthrough), or macvtap (virtual interface with a tap-like software interface). Note that this only works for Neutron deployments that support the bindings extension.

String value expected.

Can be updated without replacement.

Defaults to `"normal"`

Allowed values: normal, direct, macvtap, direct-physical, baremetal, virtio-forwarder, smart-nic

mac_address*ű*

Optional.

MAC address to give to this port. The default update policy of this property in neutron is that allow admin role only.

String value expected.

Can be updated without replacement.

Value must be of type mac_addr

no_fixed_ips*ű*

Available since 16.0.0 (Wallaby)

Optional.

Flag to disable all fixed ips on the port.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

port_security_enabled*ű*

Available since 5.0.0 (Liberty)

Optional.

Flag to enable/disable port security on the port. When disable this feature(set it to False), there will be no packages filtering, like security-group and address-pairs.

Boolean value expected.

Can be updated without replacement.

propagate_uplink_status*ŕí*

Available since 15.0.0 (Victoria)

Optional.

Flag to enable/disable propagate uplink status on the port.

Boolean value expected.

Can be updated without replacement.

qos_policy*ŕí*

Available since 6.0.0 (Mitaka)

Optional.

The name or ID of QoS policy to attach to this port.

String value expected.

Can be updated without replacement.

Value must be of type neutron.qos_policy

value_specs*ŕí*

Extra parameters to include in the request.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

subnet*ŕí*

Available since 5.0.0 (Liberty)

Optional.

Subnet in which to allocate the IP address for port. Used for creating port, based on derived properties. If subnet is specified, network property becomes optional.

String value expected.

Can be updated without replacement.

tag*ŕí*

Available since 9.0.0 (Pike)

Optional.

Port tag. Heat ignores any update on this property as nova does not support it.

String value expected.

Can be updated without replacement.

reservation_id*ű*

A UUID for the set of servers being requested.

String value expected.

Updates cause replacement.

scheduler_hints*ű*

Arbitrary key-value pairs specified by the client to help boot a server.

Map value expected.

Updates cause replacement.

security_groups*ű*

List of security group names or IDs. Cannot be used if neutron ports are associated with this server; assign security groups to the ports instead.

List value expected.

Updates cause replacement.

Defaults to `[]`

software_config_transport*ű*

How the server should receive the metadata required for software configuration. POLL_SERVER_CFN will allow calls to the cfn API action DescribeStackResource authenticated with the provided keypair. POLL_SERVER_HEAT will allow calls to the Heat API resource-show using the provided keystone credentials. POLL_TEMP_URL will create and populate a Swift TempURL with metadata for polling. ZAQAR_MESSAGE will create a dedicated zaqar queue and post the metadata for polling.

String value expected.

Can be updated without replacement.

Defaults to `"POLL_SERVER_CFN"`

Allowed values: POLL_SERVER_CFN, POLL_SERVER_HEAT, POLL_TEMP_URL, ZAQAR_MESSAGE

tags*ű*

Available since 8.0.0 (Ocata)

Server tags. Supported since client version 2.26.

List value expected.

Can be updated without replacement.

List contents:

Optional.

String value expected.

Can be updated without replacement.

user_data*ű*

User data script to be executed by cloud-init or CoreOS ignition. Changes cause replacement of the resource by default, but can be ignored altogether by setting the 'user_data_update_policy' property.

String value expected.

Can be updated without replacement.

Defaults to `""`

user_data_format*ű*

How the user_data should be formatted for the server. For HEAT_CFNTOOLS, the user_data is bundled as part of the heat-cfntools cloud-init boot configuration data. For RAW the user_data is passed to Nova unmodified. For SOFTWARE_CONFIG user_data is bundled as part of the software config data, and metadata is derived from any associated SoftwareDeployment resources. And if the user_data is in CoreOS ignition(json) format, the metadata will be injected into the user_data automatically by Heat.

String value expected.

Updates cause replacement.

Defaults to `"HEAT_CFNTOOLS"`

Allowed values: HEAT_CFNTOOLS, RAW, SOFTWARE_CONFIG

user_data_update_policy*ű*

Available since 6.0.0 (Mitaka)

Policy on how to apply a user_data update; by ignoring it, by replacing the entire server, or rebuild the server.

String value expected.

Can be updated without replacement.

Defaults to `"REPLACE"`

Allowed values: REPLACE, IGNORE, REBUILD

personality*ű*

DEPRECATED since 12.0.0 (Stein) - This is not supported with nova api microversion 2.57 and above. OS::Nova::Server resource will not support it in the future. Please use user_data or metadata instead. However, you can set heat config option max_nova_api_microversion < 2.57 to use this property in the meantime.

A map of files to create/overwrite on the server upon boot. Keys are file names and values are the file contents.

Map value expected.

Updates cause replacement.

Defaults to `{}`

Attributes

accessIPv4*ií*

DEPRECATED since 14.0.0 (Ussuri)

Available since 2015.1 (Kilo)

The manually assigned alternative public IPv4 address of the server.

accessIPv6*ií*

DEPRECATED since 14.0.0 (Ussuri)

Available since 2015.1 (Kilo)

The manually assigned alternative public IPv6 address of the server.

addresses*ií*

Available since 11.0.0 (Rocky) - The attribute was extended to include subnets and network with version 11.0.0.

A dict of all network addresses with corresponding port_id and subnets. Each network will have two keys in dict, they are network name and network id. The port ID may be obtained through the following expression: "{get_attr: [<server>, addresses, <network name_or_id>, 0, port]}". The subnets may be obtained trough the following expression: "{get_attr: [<server>, addresses, <network name_or_id>, 0, subnets]}". The network may be obtained through the following expression: "{get_attr: [<server>, addresses, <network name_or_id>, 0, network]}".

console_urls*ií*

Available since 2015.1 (Kilo)

URLs of servers consoles. To get a specific console type, the requested type can be specified as parameter to the get_attr function, e.g. get_attr: [<server>, console_urls, novnc]. Currently supported types are novnc, xvpvnc, spice-html5, rdp-html5, serial and webmks.

instance_name*ií*

AWS compatible instance name.

name*ű*

Name of the server.

networks*ű*

A dict of assigned network addresses of the form: {public: [ip1, ip2], private: [ip3, ip4], public_uuid: [ip1, ip2], private_uuid: [ip3, ip4]}. Each network will have two keys in dict, they are network name and network id.

os_collect_config*ű*

Available since 9.0.0 (Pike)

The os-collect-config configuration for the servers local agent to be configured to connect to Heat to retrieve deployment data.

show*ű*

Detailed information about resource.

tags*ű*

Available since 8.0.0 (Ocata)

Tags from the server. Supported since client version 2.26.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Nova::Server
    properties:
      admin_pass: String
      availability_zone: String
      block_device_mapping: [{"device_name": String, "volume_id": String,
↪"snapshot_id": String, "volume_size": Integer, "delete_on_termination":␣
↪Boolean}, {"device_name": String, "volume_id": String, "snapshot_id":␣
↪String, "volume_size": Integer, "delete_on_termination": Boolean}, ...]
      block_device_mapping_v2: [{"device_name": String, "volume_id": String,
↪"image_id": String, "image": String, "snapshot_id": String, "swap_size":␣
↪Integer, "ephemeral_size": Integer, "ephemeral_format": String, "device_type
↪": String, "disk_bus": String, "boot_index": Integer, "volume_size":␣
↪Integer, "delete_on_termination": Boolean}, {"device_name": String, "volume_
↪id": String, "image_id": String, "image": String, "snapshot_id": String,
↪"swap_size": Integer, "ephemeral_size": Integer, "ephemeral_format": String,
↪ "device_type": String, "disk_bus": String, "boot_index": Integer, "volume_
↪size": Integer, "delete_on_termination": Boolean}, ...]
      config_drive: Boolean
      deployment_swift_data: {"container": String, "object": String}
      diskConfig: String
```

(continues on next page)

(continued from previous page)

```
      flavor: String
      flavor_update_policy: String
      image: String
      image_update_policy: String
      key_name: String
      metadata: {...}
      name: String
      networks: [{"uuid": String, "network": String, "allocate_network":␣
↪String, "fixed_ip": String, "port": String, "port_extra_properties": {
↪"value_specs": {...}, "admin_state_up": Boolean, "mac_address": String,
↪"allowed_address_pairs": [{"mac_address": String, "ip_address": String}, {
↪"mac_address": String, "ip_address": String}, ...], "binding:vnic_type":␣
↪String, "port_security_enabled": Boolean, "qos_policy": String, "propagate_
↪uplink_status": Boolean, "no_fixed_ips": Boolean}, "subnet": String,
↪"floating_ip": String, "tag": String}, {"uuid": String, "network": String,
↪"allocate_network": String, "fixed_ip": String, "port": String, "port_extra_
↪properties": {"value_specs": {...}, "admin_state_up": Boolean, "mac_address
↪": String, "allowed_address_pairs": [{"mac_address": String, "ip_address":␣
↪String}, {"mac_address": String, "ip_address": String}, ...], "binding:vnic_
↪type": String, "port_security_enabled": Boolean, "qos_policy": String,
↪"propagate_uplink_status": Boolean, "no_fixed_ips": Boolean}, "subnet":␣
↪String, "floating_ip": String, "tag": String}, ...]
      reservation_id: String
      scheduler_hints: {...}
      security_groups: [Value, Value, ...]
      software_config_transport: String
      tags: [String, String, ...]
      user_data: String
      user_data_format: String
      user_data_update_policy: String
```

OS::Nova::ServerGroup

Available since 2014.2 (Juno)

A resource for managing a Nova server group.

Server groups allow you to make sure instances (VM/VPS) are on the same hypervisor host or on a different one.

Optional Properties

name*ű*

Server Group name.

String value expected.

Updates cause replacement.

policies*ű*

A list of exactly one policy to apply. Defaults to anti-affinity.

List value expected.

Updates cause replacement.

Defaults to `["anti-affinity"]`

Allowed values: anti-affinity, affinity, soft-anti-affinity, soft-affinity

List contents:

Optional.

String value expected.

Updates cause replacement.

rules*ű*

Available since 17.0.0 (Xena)

Rules for a policy.

Map value expected.

Updates cause replacement.

Map properties:

max_server_per_host*ű*

Optional.

Maximum servers in a group on a given host. Rule for anti-affinity policy.

Number value expected.

Updates cause replacement.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Nova::ServerGroup
    properties:
      name: String
      policies: [String, String, ...]
      rules: {"max_server_per_host": Number}
```

OS::Octavia::AvailabilityZone

Available since 24.0.0

A resource for creating octavia Availability Zones.

This resource creates and manages octavia Availability Zones, which allows to tune Load Balancers capabilities.

Required Properties

availability_zone_profile*ű*

The ID or the name of the Availability Zone Profile.

String value expected.

Updates cause replacement.

Value must be of type octavia.availabilityzoneprofile

Optional Properties

description*ű*

Description of this Availability Zone.

String value expected.

Can be updated without replacement.

Defaults to `""`

enabled*ű*

If the resource if available for use.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

name*ű*

Name of this Availability Zone.

String value expected.

Can be updated without replacement.

Attributes

availability_zone_profile_id*ű*
The ID of the availability zone profile.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::AvailabilityZone
    properties:
      availability_zone_profile: String
      description: String
      enabled: Boolean
      name: String
```

OS::Octavia::AvailabilityZoneProfile

Available since 24.0.0

A resource for creating octavia Availability Zone Profiles.

This resource creates and manages octavia Availability Zone Profiles, which allows to tune Load Balancers capabilities.

Required Properties

availability_zone_data*ří*

JSON string containing the availability zone metadata.

String value expected.

Can be updated without replacement.

Value must be of type json_string

Optional Properties

name*ří*

Name of this Availability Zone Profile.

String value expected.

Can be updated without replacement.

provider_name*ří*

Provider name of this Availability Zone.

String value expected.

Can be updated without replacement.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::AvailabilityZoneProfile
    properties:
      availability_zone_data: String
      name: String
      provider_name: String
```

OS::Octavia::Flavor

Available since 14.0.0 (Ussuri)

A resource for creating octavia Flavors.

This resource creates and manages octavia Flavors, which allows to tune Load Balancers capabilities.

Required Properties

flavor_profile*ű*

The ID or the name of the Flavor Profile.

String value expected.

Updates cause replacement.

Value must be of type octavia.flavorprofile

Optional Properties

description*ű*

Description of this Flavor.

String value expected.

Can be updated without replacement.

Defaults to ""

enabled*ű*

If the resource if available for use.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

name*ű*

Name of this Flavor.

String value expected.

Can be updated without replacement.

Attributes

flavor_profile_id*ű*
The ID of the flavor profile.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::Flavor
    properties:
      description: String
      enabled: Boolean
      flavor_profile: String
      name: String
```

OS::Octavia::FlavorProfile

Available since 14.0.0 (Ussuri)

A resource for creating octavia Flavor Profiles.

This resource creates and manages octavia Flavor Profiles, which allows to tune Load Balancers capabilities.

Required Properties

flavor_data*ű*

JSON string containing the flavor metadata.

String value expected.

Can be updated without replacement.

Value must be of type json_string

Optional Properties

name*ű*

Name of this Flavor Profile.

String value expected.

Can be updated without replacement.

provider_name*ű*

Provider name of this Flavor Profile.

String value expected.

Can be updated without replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::FlavorProfile
    properties:
      flavor_data: String
      name: String
      provider_name: String
```

OS::Octavia::HealthMonitor

Available since 10.0.0 (Queens)

A resource to handle load balancer health monitors.

This resource creates and manages octavia healthmonitors, which watches status of the load balanced servers.

Required Properties

delay*ű*

The minimum time in seconds between regular connections of the member.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

max_retries*ŕ*

Number of permissible connection failures before changing the member status to INACTIVE.

Integer value expected.

Can be updated without replacement.

The value must be in the range 1 to 10.

pool*ŕ*

ID or name of the load balancing pool.

String value expected.

Updates cause replacement.

Value must be of type octavia.pool

timeout*ŕ*

Maximum number of seconds for a monitor to wait for a connection to be established before it times out.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

type*ŕ*

One of predefined health monitor types.

String value expected.

Updates cause replacement.

Allowed values: PING, TCP, HTTP, HTTPS, UDP-CONNECT

Optional Properties

admin_state_up*ŕ*

The administrative state of the health monitor.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

expected_codes*ŕ*

The HTTP status codes expected in response from the member to declare it healthy. Specify one of the following values: a single value, such as 200. a list, such as 200, 202. a range, such as 200-204.

String value expected.

Can be updated without replacement.

http_method*ŕ*

The HTTP method used for requests by the monitor of type HTTP.

String value expected.

Can be updated without replacement.

Allowed values: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH

tenant_id*ű*

ID of the tenant who owns the health monitor.

String value expected.

Updates cause replacement.

url_path*ű*

The HTTP path used in the HTTP request used by the monitor to test a member health. A valid value is a string the begins with a forward slash (/).

String value expected.

Can be updated without replacement.

Attributes

pools*ű*
The list of Pools related to this monitor.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::HealthMonitor
    properties:
      admin_state_up: Boolean
      delay: Integer
      expected_codes: String
      http_method: String
      max_retries: Integer
      pool: String
      tenant_id: String
      timeout: Integer
      type: String
      url_path: String
```

OS::Octavia::L7Policy

Available since 10.0.0 (Queens)

A resource for managing octavia L7Policies.

This resource manages L7Policies, which represent a collection of L7Rules. L7Policy holds the action that should be performed when the rules are matched (Redirect to Pool, Redirect to URL, Reject). L7Policy holds a Listener id, so a Listener can evaluate a collection of L7Policies. L7Policy will return True when all of the L7Rules that belong to this L7Policy are matched. L7Policies under a specific Listener are ordered and the first l7Policy that returns a match will be executed. When none of the policies match the request gets forwarded to listener.default_pool_id.

Required Properties

action*ű*

Action type of the policy.

String value expected.

Can be updated without replacement.

Allowed values: REJECT, REDIRECT_TO_POOL, REDIRECT_TO_URL

listener*ű*

ID or name of the listener this policy belongs to.

String value expected.

Updates cause replacement.

Value must be of type octavia.listener

Optional Properties

admin_state_up*ű*

The administrative state of the policy.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

description*ű*

Description of the policy.

String value expected.

Can be updated without replacement.

name*ű*

Name of the policy.

String value expected.

Can be updated without replacement.

position*ű*

L7 policy position in ordered policies list. This must be an integer starting from 1. If not specified, policy will be placed at the tail of existing policies list.

Number value expected.

Can be updated without replacement.

The value must be at least 1.

redirect_pool*ű*

ID or name of the pool for REDIRECT_TO_POOL action type.

String value expected.

Can be updated without replacement.

Value must be of type octavia.pool

redirect_url*ű*

URL for REDIRECT_TO_URL action type. This should be a valid URL string.

String value expected.

Can be updated without replacement.

Attributes

rules*ű*
L7Rules associated with this policy.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::L7Policy
    properties:
      action: String
      admin_state_up: Boolean
      description: String
      listener: String
      name: String
      position: Number
      redirect_pool: String
      redirect_url: String
```

OS::Octavia::L7Rule

Available since 10.0.0 (Queens)

A resource for managing octavia L7Rules.

This resource manages L7Rules, which represent a set of attributes that defines which part of the request should be matched and how it should be matched.

Required Properties

compare_type*ű*

Rule compare type.

String value expected.

Can be updated without replacement.

Allowed values: REGEX, STARTS_WITH, ENDS_WITH, CONTAINS, EQUAL_TO

l7policy*ű*

ID or name of L7 policy this rule belongs to.

String value expected.

Updates cause replacement.

Value must be of type octavia.l7policy

type*ű*

Rule type.

String value expected.

Can be updated without replacement.

Allowed values: HOST_NAME, PATH, FILE_TYPE, HEADER, COOKIE

value*ű*

Value to compare.

String value expected.

Can be updated without replacement.

Optional Properties

admin_state_up*ű*

The administrative state of the rule.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

invert*ű*

Invert the compare type.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

key*ű*

Key to compare. Relevant for HEADER and COOKIE types only.

String value expected.

Can be updated without replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::L7Rule
    properties:
      admin_state_up: Boolean
      compare_type: String
      invert: Boolean
      key: String
      l7policy: String
      type: String
      value: String
```

OS::Octavia::Listener

Available since 10.0.0 (Queens)

A resource for managing octavia Listeners.

This resource creates and manages Neutron octavia Listeners, which represent a listening endpoint for the vip.

Required Properties

loadbalancer*ű*

ID or name of the load balancer with which listener is associated.

String value expected.

Updates cause replacement.

Value must be of type octavia.loadbalancer

protocol*ú*

Protocol on which to listen for the client traffic.

String value expected.

Updates cause replacement.

Allowed values: TCP, HTTP, HTTPS, TERMINATED_HTTPS, PROXY, UDP

protocol_port*ú*

TCP or UDP port on which to listen for client traffic.

Integer value expected.

Updates cause replacement.

The value must be in the range 1 to 65535.

Optional Properties

admin_state_up*ú*

The administrative state of this listener.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

allowed_cidrs*ú*

Available since 14.0.0 (Ussuri)

A list of IPv4, IPv6 or mix of both CIDRs. The default is all allowed. When a list of CIDRs is provided, the default switches to deny all.

List value expected.

Can be updated without replacement.

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type net_cidr

connection_limit*ú*

The maximum number of connections permitted for this load balancer. Defaults to -1, which is infinite.

Integer value expected.

Can be updated without replacement.

Defaults to `-1`

The value must be at least -1.

default_pool*ű*

ID or name of the default pool for the listener.

String value expected.

Can be updated without replacement.

Value must be of type octavia.pool

default_tls_container_ref*ű*

Default TLS container reference to retrieve TLS information.

String value expected.

Can be updated without replacement.

description*ű*

Description of this listener.

String value expected.

Can be updated without replacement.

Defaults to `""`

name*ű*

Name of this listener.

String value expected.

Can be updated without replacement.

sni_container_refs*ű*

List of TLS container references for SNI.

List value expected.

Can be updated without replacement.

tenant_id*ű*

The ID of the tenant who owns the listener.

String value expected.

Updates cause replacement.

Attributes

default_pool_id*ű*
: ID of the default pool this listener is associated to.

loadbalancers*ű*
: ID of the load balancer this listener is associated to.

show*ű*
: Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::Listener
    properties:
      admin_state_up: Boolean
      allowed_cidrs: [String, String, ...]
      connection_limit: Integer
      default_pool: String
      default_tls_container_ref: String
      description: String
      loadbalancer: String
      name: String
      protocol: String
      protocol_port: Integer
      sni_container_refs: [Value, Value, ...]
      tenant_id: String
```

OS::Octavia::LoadBalancer

Available since 10.0.0 (Queens)

A resource for creating octavia Load Balancers.

This resource creates and manages octavia Load Balancers, which allows traffic to be directed between servers.

Required Properties

vip_subnet*ú*

The name or ID of the subnet on which to allocate the VIP address.

String value expected.

Updates cause replacement.

Value must be of type neutron.subnet

Optional Properties

admin_state_up*ú*

The administrative state of this Load Balancer.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

availability_zone*ű*

Available since 17.0.0 (Xena)

The availability zone of the Load Balancer.

String value expected.

Updates cause replacement.

Value must be of type octavia.availabilityzone

description*ű*

Description of this Load Balancer.

String value expected.

Can be updated without replacement.

Defaults to ""

flavor*ű*

Available since 14.0.0 (Ussuri)

The name or ID of the flavor of the Load Balancer.

String value expected.

Updates cause replacement.

Value must be of type octavia.flavor

name*ű*

Name of this Load Balancer.

String value expected.

Can be updated without replacement.

provider*ű*

Provider for this Load Balancer.

String value expected.

Updates cause replacement.

tenant_id*ű*

The ID of the tenant who owns the Load Balancer. Only administrative users can specify a tenant ID other than their own.

String value expected.

Updates cause replacement.

Value must be of type keystone.project

vip_address*ű*

IP address for the VIP.

String value expected.

Updates cause replacement.

Value must be of type ip_addr

Attributes

flavor_id*ű*
The flavor ID of the LoadBalancer.

pools*ű*
Pools this LoadBalancer is associated with.

show*ű*
Detailed information about resource.

vip_address*ű*
The VIP address of the LoadBalancer.

vip_port_id*ű*
The VIP port of the LoadBalancer.

vip_subnet_id*ű*
The VIP subnet of the LoadBalancer.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::LoadBalancer
    properties:
      admin_state_up: Boolean
      availability_zone: String
      description: String
      flavor: String
      name: String
      provider: String
      tenant_id: String
      vip_address: String
      vip_subnet: String
```

OS::Octavia::Pool

Available since 10.0.0 (Queens)

A resource for managing Octavia Pools.

This resources manages octavia LBaaS Pools, which represent a group of nodes. Pools define the subnet where nodes reside, balancing algorithm, and the nodes themselves.

Required Properties

lb_algorithm*ŕ*

The algorithm used to distribute load between the members of the pool.

String value expected.

Can be updated without replacement.

Allowed values: ROUND_ROBIN, LEAST_CONNECTIONS, SOURCE_IP, SOURCE_IP_PORT

protocol*ŕ*

Protocol of the pool.

String value expected.

Updates cause replacement.

Allowed values: TCP, HTTP, HTTPS, TERMINATED_HTTPS, PROXY, UDP

Optional Properties

admin_state_up*ŕ*

The administrative state of this pool.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

description*ŕ*

Description of this pool.

String value expected.

Can be updated without replacement.

Defaults to `""`

listener*ŕ*

Listener name or ID to be associated with this pool.

String value expected.

Updates cause replacement.

Value must be of type octavia.listener

loadbalancer*ŕ*

Loadbalancer name or ID to be associated with this pool.

String value expected.

Updates cause replacement.

Value must be of type octavia.loadbalancer

name*ű*

Name of this pool.

String value expected.

Can be updated without replacement.

session_persistence*ű*

Configuration of session persistence.

Map value expected.

Can be updated without replacement.

Map properties:

cookie_name*ű*

Optional.

Name of the cookie, required if type is APP_COOKIE.

String value expected.

Can be updated without replacement.

type*ű*

Required.

Method of implementation of session persistence feature.

String value expected.

Can be updated without replacement.

Allowed values: SOURCE_IP, HTTP_COOKIE, APP_COOKIE

tls_enabled*ű*

Available since 14.0.0 (Ussuri)

Enable backend member re-encryption.

Boolean value expected.

Can be updated without replacement.

Defaults to `false`

Attributes

healthmonitor_id*ű*
ID of the health monitor associated with this pool.

listeners*ű*
Listener associated with this pool.

members*ű*
Members associated with this pool.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::Pool
    properties:
      admin_state_up: Boolean
      description: String
      lb_algorithm: String
      listener: String
      loadbalancer: String
      name: String
      protocol: String
      session_persistence: {"type": String, "cookie_name": String}
      tls_enabled: Boolean
```

OS::Octavia::PoolMember

Available since 10.0.0 (Queens)

A resource for managing Octavia Pool Members.

A pool member represents a single backend node.

Required Properties

address*ű*

IP address of the pool member on the pool network.

String value expected.

Updates cause replacement.

Value must be of type ip_addr

pool*ű*

Name or ID of the load balancing pool.

String value expected.

Updates cause replacement.

Value must be of type octavia.pool

protocol_port*ű*

Port on which the pool member listens for requests or connections.

Integer value expected.

Updates cause replacement.

The value must be in the range 1 to 65535.

Optional Properties

admin_state_up*ű*

The administrative state of the pool member.

Boolean value expected.

Can be updated without replacement.

Defaults to `true`

monitor_address*ű*

Alternate IP address which health monitor can use for health check.

String value expected.

Updates cause replacement.

Value must be of type ip_addr

monitor_port*ű*

Alternate Port which health monitor can use for health check.

Integer value expected.

Updates cause replacement.

The value must be in the range 1 to 65535.

subnet*ű*

Subnet name or ID of this member.

String value expected.

Updates cause replacement.

Value must be of type neutron.subnet

tags*ű*

Available since 13.0.0 (Train)

A list of simple strings assigned to the member. The property is supported with Stein Octavia or newer version.

List value expected.

Can be updated without replacement.

weight*ű*

Weight of pool member in the pool (default to 1).

Integer value expected.

Can be updated without replacement.

Defaults to 1

The value must be in the range 0 to 256.

Attributes

show*ú*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::PoolMember
    properties:
      address: String
      admin_state_up: Boolean
      monitor_address: String
      monitor_port: Integer
      pool: String
      protocol_port: Integer
      subnet: String
      tags: [Value, Value, ...]
      weight: Integer
```

OS::Octavia::Quota

Available since 14.0.0 (Ussuri)

A resource for creating Octavia quotas.

Ocatavia Quota is used to manage operational limits for Octavia. Currently, this resource can manage Octavias quotas for:

- healthmonitor
- listener
- loadbalancer
- pool
- member

Note that default octavia security policy usage of this resource is limited to being used by administrators only. Administrators should be careful to create only one Octavia Quota resource per project, otherwise it will be hard for them to manage the quota properly.

Required Properties

project*ű*

Name or id of the project to set the quota for.

String value expected.

Updates cause replacement.

Value must be of type keystone.project

Optional Properties

healthmonitor*ű*

Quota for the number of healthmonitors. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

listener*ű*

Quota for the number of listeners. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

loadbalancer*ű*

Quota for the number of load balancers. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

member*ű*

Quota for the number of m. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

pool*ű*

Quota for the number of pools. Setting the value to -1 removes the limit.

Integer value expected.

Can be updated without replacement.

The value must be at least -1.

Attributes

show*ú*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Octavia::Quota
    properties:
      healthmonitor: Integer
      listener: Integer
      loadbalancer: Integer
      member: Integer
      pool: Integer
      project: String
```

OS::Swift::Container

A resource for managing Swift containers.

A container defines a namespace for objects. An object with the same name in two different containers represents two different objects.

Optional Properties

PurgeOnDelete*ú*

Available since 2015.1 (Kilo)

If True, delete any objects in the container when the container is deleted. Otherwise, deleting a non-empty container will result in an error.

Boolean value expected.

Updates cause replacement.

Defaults to `false`

X-Account-Meta*ú*

A map of user-defined meta data to associate with the account. Each key in the map will set the header X-Account-Meta-{key} with the corresponding value.

Map value expected.

Updates cause replacement.

Defaults to `{}`

X-Container-Meta*ú*

A map of user-defined meta data to associate with the container. Each key in the map will set the header X-Container-Meta-{key} with the corresponding value.

Map value expected.

Updates cause replacement.

Defaults to {}

X-Container-Read*ű*

Specify the ACL permissions on who can read objects in the container.

String value expected.

Updates cause replacement.

X-Container-Write*ű*

Specify the ACL permissions on who can write objects to the container.

String value expected.

Updates cause replacement.

name*ű*

Name for the container. If not specified, a unique name will be generated.

String value expected.

Updates cause replacement.

Attributes

BytesUsed*ű*
The number of bytes stored in the container.

DomainName*ű*
The host from the container URL.

HeadContainer*ű*
A map containing all headers for the container.

ObjectCount*ű*
The number of objects stored in the container.

RootURL*ű*
The parent URL of the container.

WebsiteURL*ű*
The URL of the container.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
```

(continues on next page)

(continued from previous page)

```
...
the_resource:
  type: OS::Swift::Container
  properties:
    PurgeOnDelete: Boolean
    X-Account-Meta: {...}
    X-Container-Meta: {...}
    X-Container-Read: String
    X-Container-Write: String
    name: String
```

OS::Trove::Cluster

Available since 2015.1 (Kilo)

A resource for managing Trove clusters.

A Cluster is an opaque cluster used to store Database clusters.

Required Properties

datastore_type*ű*

Name of registered datastore type.

String value expected.

Updates cause replacement.

The length must be no greater than 255.

datastore_version*ű*

Name of the registered datastore version. It must exist for provided datastore type. Defaults to using single active version. If several active versions exist for provided datastore type, explicit value for this parameter must be specified.

String value expected.

Updates cause replacement.

The length must be no greater than 255.

instances*ű*

List of database instances.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

availability_zone*ű*

Available since 14.0.0 (Ussuri)

Optional.

Name of the availability zone for DB instance.

String value expected.

Updates cause replacement.

flavor*ű*

Required.

Flavor of the instance.

String value expected.

Updates cause replacement.

Value must be of type trove.flavor

networks*ű*

Available since 10.0.0 (Queens)

List of network interfaces to create on instance.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

fixed_ip*ű*

Optional.

Fixed IPv4 address for this NIC.

String value expected.

Updates cause replacement.

Value must be of type ip_addr

network*ű*

Optional.

Name or UUID of the network to attach this NIC to. Either port or network must be specified.

String value expected.

Updates cause replacement.

Value must be of type neutron.network

port*ű*

Optional.

Name or UUID of Neutron port to attach this NIC to. Either port or network must be specified.

String value expected.

Updates cause replacement.

Value must be of type neutron.port

volume_size*ű*

Required.

Size of the instance disk volume in GB.

Integer value expected.

Updates cause replacement.

The value must be in the range 1 to 150.

Optional Properties

name*ű*

Name of the cluster to create.

String value expected.

Updates cause replacement.

The length must be no greater than 255.

Attributes

instances*ű*
A list of instances ids.

ip*ű*
A list of cluster instance IPs.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
```

(continues on next page)

(continued from previous page)

```
  the_resource:
    type: OS::Trove::Cluster
    properties:
      datastore_type: String
      datastore_version: String
      instances: [{"flavor": String, "volume_size": Integer, "networks": [{
↪"network": String, "port": String, "fixed_ip": String}, {"network": String,
↪"port": String, "fixed_ip": String}, ...], "availability_zone": String}, {
↪"flavor": String, "volume_size": Integer, "networks": [{"network": String,
↪"port": String, "fixed_ip": String}, {"network": String, "port": String,
↪"fixed_ip": String}, ...], "availability_zone": String}, ...]
      name: String
```

OS::Trove::Instance

Available since 2014.1 (Icehouse)

OpenStack cloud database instance resource.

Trove is Database as a Service for OpenStack. Its designed to run entirely on OpenStack, with the goal of allowing users to quickly and easily utilize the features of a relational or non-relational database without the burden of handling complex administrative tasks.

Required Properties

flavor*ű*

Reference to a flavor for creating DB instance.

String value expected.

Can be updated without replacement.

Value must be of type trove.flavor

size*ű*

Database volume size in GB.

Integer value expected.

Can be updated without replacement.

The value must be in the range 1 to 150.

Optional Properties

availability_zone*ű*

Name of the availability zone for DB instance.

String value expected.

Updates cause replacement.

databases*ű*

List of databases to be created on DB instance creation.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

character_set*ří*

Optional.

Set of symbols and encodings.

String value expected.

Can be updated without replacement.

Defaults to `"utf8"`

collate*ří*

Optional.

Set of rules for comparing characters in a character set.

String value expected.

Can be updated without replacement.

Defaults to `"utf8_general_ci"`

name*ří*

Required.

Specifies database names for creating databases on instance creation.

String value expected.

Can be updated without replacement.

The length must be no greater than 64.

Value must match pattern: [a-zA-Z0-9_-]+[a-zA-Z0-9_@?#\s-]*[a-zA-Z0-9_-]+

datastore_type*ří*

Name of registered datastore type.

String value expected.

Updates cause replacement.

The length must be no greater than 255.

datastore_version*ří*

Name of the registered datastore version. It must exist for provided datastore type. Defaults to using single active version. If several active versions exist for provided datastore type, explicit value for this parameter must be specified.

String value expected.

Updates cause replacement.

The length must be no greater than 255.

name*ű*

Name of the DB instance to create.

String value expected.

Can be updated without replacement.

The length must be no greater than 255.

networks*ű*

List of network interfaces to create on instance.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

fixed_ip*ű*

Optional.

Fixed IPv4 address for this NIC.

String value expected.

Updates cause replacement.

Value must be of type ip_addr

network*ű*

Optional.

Name or UUID of the network to attach this NIC to. Either port or network must be specified.

String value expected.

Updates cause replacement.

Value must be of type neutron.network

port*ű*

Optional.

Name or UUID of Neutron port to attach this NIC to. Either port or network must be specified.

String value expected.

Updates cause replacement.

Value must be of type neutron.port

replica_count*ŭ*

Available since 5.0.0 (Liberty)

The number of replicas to be created.

Integer value expected.

Updates cause replacement.

replica_of*ŭ*

Available since 5.0.0 (Liberty)

Identifier of the source instance to replicate.

String value expected.

Updates cause replacement.

restore_point*ŭ*

DB instance restore point.

String value expected.

Updates cause replacement.

users*ŭ*

List of users to be created on DB instance creation.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

databases*ŭ*

Names of databases that those users can access on instance creation.

List value expected.

Can be updated without replacement.

The length must be at least 1.

List contents:

Optional.

String value expected.

Can be updated without replacement.

host*ű*

Optional.

The host from which a user is allowed to connect to the database.

String value expected.

Can be updated without replacement.

Defaults to `"%"`

name*ű*

Required.

User name to create a user on instance creation.

String value expected.

Can be updated without replacement.

The length must be no greater than 16.

Value must match pattern: [a-zA-Z0-9_]+[a-zA-Z0-9_@?#\s]*[a-zA-Z0-9_]+

password*ű*

Required.

Password for those users on instance creation.

String value expected.

Can be updated without replacement.

Value must match pattern: [a-zA-Z0-9_]+[a-zA-Z0-9_@?#\s]*[a-zA-Z0-9_]+

Attributes

hostname*ű*
Hostname of the instance.

ref*ű*
Api endpoint reference of the instance.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Trove::Instance
    properties:
      availability_zone: String
      databases: [{"character_set": String, "collate": String, "name": String}
↪, {"character_set": String, "collate": String, "name": String}, ...]
      datastore_type: String
      datastore_version: String
      flavor: String
      name: String
      networks: [{"network": String, "port": String, "fixed_ip": String}, {
↪"network": String, "port": String, "fixed_ip": String}, ...]
      replica_count: Integer
      replica_of: String
      restore_point: String
      size: Integer
      users: [{"name": String, "password": String, "host": String, "databases
↪": [String, String, ...]}, {"name": String, "password": String, "host":␣
↪String, "databases": [String, String, ...]}, ...]
```

OS::Zaqar::MistralTrigger

Available since 8.0.0 (Ocata)

A Zaqar subscription for triggering Mistral workflows.

This Zaqar subscription type listens for messages in a queue and triggers a Mistral workflow execution each time one is received.

The content of the Zaqar message is passed to the workflow in the environment with the name notification, and thus is accessible from within the workflow as:

<% env().notification %>

Other environment variables can be set using the env key in the params property.

Required Properties

queue_name*ří*

Name of the queue to subscribe to.

String value expected.

Updates cause replacement.

Value must be of type zaqar.queue

workflow_id*ű*

UUID of the Mistral workflow to trigger.

String value expected.

Can be updated without replacement.

Value must be of type mistral.workflow

Optional Properties

input*ű*

Input values to pass to the Mistral workflow.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

params*ű*

Parameters to pass to the Mistral workflow execution. The parameters depend on the workflow type.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

ttl*ű*

Time to live of the subscription in seconds.

Integer value expected.

Can be updated without replacement.

Defaults to `220367260800`

The value must be at least 60.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Zaqar::MistralTrigger
    properties:
      input: {...}
      params: {...}
```

(continues on next page)

(continued from previous page)

```
      queue_name: String
      ttl: Integer
      workflow_id: String
```

OS::Zaqar::Queue

Available since 2014.2 (Juno)

A resource for managing Zaqar queues.

Queue is a logical entity that groups messages. Ideally a queue is created per work type. For example, if you want to compress files, you would create a queue dedicated for this job. Any application that reads from this queue would only compress files.

Optional Properties

metadata*ű*

Arbitrary key/value metadata to store contextual information about this queue.

Map value expected.

Can be updated without replacement.

name*ű*

Name of the queue instance to create.

String value expected.

Updates cause replacement.

The length must be no greater than 64.

Attributes

href*ű*
: The resource href of the queue.

show*ű*
: Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Zaqar::Queue
    properties:
      metadata: {...}
      name: String
```

OS::Zaqar::SignedQueueURL

Available since 8.0.0 (Ocata)

A resource for managing signed URLs of Zaqar queues.

Signed URLs allow to give specific access to queues, for example to be used as alarm notifications. To supply a signed queue URL to Aodh as an action URL, pass zaqar://? followed by the query_str attribute of the signed queue URL resource.

Required Properties

queue*ú*

Name of the queue instance to create a URL for.

String value expected.

Updates cause replacement.

Optional Properties

methods*ú*

List of allowed HTTP methods to be used. Default to allow GET.

List value expected.

Updates cause replacement.

List contents:

Optional.

String value expected.

Updates cause replacement.

Allowed values: GET, DELETE, PATCH, POST, PUT

paths*ú*

List of allowed paths to be accessed. Default to allow queue messages URL.

List value expected.

Updates cause replacement.

ttl*ú*

Time validity of the URL, in seconds. Default to one day.

Integer value expected.

Updates cause replacement.

Attributes

expires*ű*
Expiration date of the URL.

methods*ű*
Comma-delimited list of methods for convenience.

paths*ű*
Comma-delimited list of paths for convenience.

project*ű*
The ID of the Keystone project containing the queue.

query_str*ű*
An HTTP URI query fragment.

show*ű*
Detailed information about resource.

signature*ű*
Signature of the URL built by Zaqar.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Zaqar::SignedQueueURL
    properties:
      methods: [String, String, ...]
      paths: [Value, Value, ...]
      queue: String
      ttl: Integer
```

OS::Zaqar::Subscription

Available since 8.0.0 (Ocata)

A resource for managing Zaqar subscriptions.

A Zaqar subscription listens for messages in a queue and sends a notification over email or webhook.

Required Properties

queue_name*ű*

Name of the queue to subscribe to.

String value expected.

Updates cause replacement.

Value must be of type zaqar.queue

subscriber*ű*

URI of the subscriber which will be notified. Must be in the format: <TYPE>:<VALUE>.

String value expected.

Can be updated without replacement.

Optional Properties

options*ű*

Options used to configure this subscription.

Map value expected.

Can be updated without replacement.

ttl*ű*

Time to live of the subscription in seconds.

Integer value expected.

Can be updated without replacement.

Defaults to `220367260800`

The value must be at least 60.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Zaqar::Subscription
    properties:
      options: {...}
      queue_name: String
      subscriber: String
      ttl: Integer
```

OS::Zun::Container

Available since 9.0.0 (Pike)

A resource that creates a Zun Container.

This resource creates a Zun container.

Required Properties

image*ű*

Name or ID of the image.

String value expected.

Updates cause replacement.

Optional Properties

command*ű*

Send command to the container.

String value expected.

Updates cause replacement.

cpu*ű*

The number of virtual cpus.

Number value expected.

Can be updated without replacement.

environment*ű*

The environment variables.

Map value expected.

Updates cause replacement.

hints*ű*

Available since 10.0.0 (Queens)

Arbitrary key-value pairs for scheduler to select host.

Map value expected.

Updates cause replacement.

hostname*ű*

Available since 10.0.0 (Queens)

The hostname of the container.

String value expected.

Updates cause replacement.

image_driver*ű*

The image driver to use to pull container image.

String value expected.

Updates cause replacement.

Allowed values: docker, glance

image_pull_policy*ŕ*

The policy which determines if the image should be pulled prior to starting the container.

String value expected.

Updates cause replacement.

Allowed values: ifnotpresent, always, never

interactive*ŕ*

Keep STDIN open even if not attached.

Boolean value expected.

Updates cause replacement.

labels*ŕ*

Adds a map of labels to a container. May be used multiple times.

Map value expected.

Updates cause replacement.

memory*ŕ*

The container memory size in MiB.

Integer value expected.

Can be updated without replacement.

mounts*ŕ*

A list of volumes mounted inside the container.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

mount_path*ŕ*

Required.

The filesystem path inside the container.

String value expected.

Updates cause replacement.

volume_id*ű*

Optional.

The ID or name of the cinder volume mount to the container.

String value expected.

Updates cause replacement.

Value must be of type cinder.volume

volume_size*ű*

Optional.

The size of the cinder volume to create.

Integer value expected.

Updates cause replacement.

name*ű*

Name of the container.

String value expected.

Can be updated without replacement.

networks*ű*

Available since 11.0.0 (Rocky)

An ordered list of nics to be added to this server, with information about connected networks, fixed ips, port etc.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

fixed_ip*ű*

Optional.

Fixed IP address to specify for the port created on the requested network.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

network*ű*

Optional.

Name or ID of network to create a port on.

String value expected.

Can be updated without replacement.

Value must be of type neutron.network

port*ű*

Optional.

ID of an existing port to associate with this container.

String value expected.

Can be updated without replacement.

Value must be of type neutron.port

restart_policy*ű*

Restart policy to apply when a container exits. Possible values are no, on-failure[:max-retry], always, and unless-stopped.

String value expected.

Updates cause replacement.

security_groups*ű*

Available since 10.0.0 (Queens)

List of security group names or IDs.

List value expected.

Updates cause replacement.

Defaults to `[]`

tty*ű*

Available since 14.0.0 (Ussuri)

Whether the container allocates a TTY for itself.

Boolean value expected.

Updates cause replacement.

workdir*ű*

The working directory for commands to run in.

String value expected.

Updates cause replacement.

Attributes

addresses*ű*

A dict of all network addresses with corresponding port_id. Each network will have two keys in dict, they are network name and network id. The port ID may be obtained through the following expression: {get_attr: [<container>, addresses, <network name_or_id>, 0, port]}.

name*ű*

Name of the container.

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Zun::Container
    properties:
      command: String
      cpu: Number
      environment: {...}
      hints: {...}
      hostname: String
      image: String
      image_driver: String
      image_pull_policy: String
      interactive: Boolean
      labels: {...}
      memory: Integer
      mounts: [{"volume_id": String, "volume_size": Integer, "mount_path":␣
↪String}, {"volume_id": String, "volume_size": Integer, "mount_path": String}
↪, ...]
      name: String
      networks: [{"network": String, "fixed_ip": String, "port": String}, {
↪"network": String, "fixed_ip": String, "port": String}, ...]
      restart_policy: String
      security_groups: [Value, Value, ...]
      tty: Boolean
      workdir: String
```

CloudFormation Compatible Resource Types

AWS::AutoScaling::AutoScalingGroup

Available since 2014.1 (Icehouse)

Required Properties

AvailabilityZones*ű*

Not Implemented.

List value expected.

Updates cause replacement.

MaxSize*ű*

Maximum number of instances in the group.

Integer value expected.

Can be updated without replacement.

MinSize*ű*

Minimum number of instances in the group.

Integer value expected.

Can be updated without replacement.

Optional Properties

Cooldown*ű*

Cooldown period, in seconds.

Integer value expected.

Can be updated without replacement.

DesiredCapacity*ű*

Desired initial number of instances.

Integer value expected.

Can be updated without replacement.

HealthCheckGracePeriod*ű*

> **Note**
>
> Not implemented.

HealthCheckType*ű*

> **Note**
>
> Not implemented.

InstanceId*ű*

The ID of an existing instance to use to create the Auto Scaling group. If specify this property, will create the group use an existing instance instead of a launch configuration.

String value expected.

Updates cause replacement.

Value must be of type nova.server

LaunchConfigurationName*ű*

The reference to a LaunchConfiguration resource.

String value expected.

Can be updated without replacement.

LoadBalancerNames*ű*

List of LoadBalancer resources.

List value expected.

Updates cause replacement.

Tags*ű*

Tags to attach to this group.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

Key*ű*

Required.

String value expected.

Updates cause replacement.

Value*ű*

Required.

String value expected.

Updates cause replacement.

VPCZoneIdentifier*ű*

Use only with Neutron, to list the internal subnet to which the instance will be attached; needed only if multiple exist; list length must be exactly 1.

List value expected.

Updates cause replacement.

List contents:

Optional.

UUID of the internal subnet to which the instance will be attached.

String value expected.

Updates cause replacement.

Attributes

InstanceList*ří*

A comma-delimited list of server ip addresses. (Heat extension).

show*ří*

Detailed information about resource.

update_policy

AutoScalingRollingUpdate*ří*

Map value expected.

Updates cause replacement.

Map properties:

MaxBatchSize*ří*

Optional.

Integer value expected.

Updates cause replacement.

Defaults to `1`

MinInstancesInService*ří*

Optional.

Integer value expected.

Updates cause replacement.

Defaults to `0`

PauseTime*ří*

Optional.

String value expected.

Updates cause replacement.

Defaults to `"PT0S"`

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::AutoScaling::AutoScalingGroup
```

(continues on next page)

(continued from previous page)

```
  properties:
    AvailabilityZones: [Value, Value, ...]
    Cooldown: Integer
    DesiredCapacity: Integer
    InstanceId: String
    LaunchConfigurationName: String
    LoadBalancerNames: [Value, Value, ...]
    MaxSize: Integer
    MinSize: Integer
    Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
    VPCZoneIdentifier: [String, String, ...]
```

AWS::AutoScaling::LaunchConfiguration

Optional Properties

BlockDeviceMappings*ří*

Block device mappings to attach to instance.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

DeviceName*ří*

Required.

A device name where the volume will be attached in the system at /dev/device_name.e.g. vdb

String value expected.

Updates cause replacement.

Ebs*ří*

The ebs volume to attach to the instance.

Map value expected.

Updates cause replacement.

Map properties:

DeleteOnTermination*ří*

Optional.

Indicate whether the volume should be deleted when the instance is terminated.

Boolean value expected.

Updates cause replacement.

Defaults to `true`

Iops*ű*

Not implemented.

SnapshotId*ű*

Optional.

The ID of the snapshot to create a volume from.

String value expected.

Updates cause replacement.

Value must be of type cinder.snapshot

VolumeSize*ű*

Optional.

The size of the volume, in GB. Must be equal or greater than the size of the snapshot. It is safe to leave this blank and have the Compute service infer the size.

String value expected.

Updates cause replacement.

VolumeType*ű*

Not implemented.

NoDevice*ű*

Not implemented.

VirtualName*ű*

Not implemented.

ImageId*ű*

Glance image ID or name.

String value expected.

Updates cause replacement.

Value must be of type glance.image

InstanceId*ű*

The ID of an existing instance you want to use to create the launch configuration. All properties are derived from the instance with the exception of BlockDeviceMapping.

String value expected.

Updates cause replacement.

Value must be of type nova.server

InstanceType*ű*

Nova instance type (flavor).

String value expected.

Updates cause replacement.

Value must be of type nova.flavor

KernelId*ű*

> **Note**
>
> Not implemented.

KeyName*ű*

Optional Nova keypair name.

String value expected.

Updates cause replacement.

Value must be of type nova.keypair

NovaSchedulerHints*ű*

Scheduler hints to pass to Nova (Heat extension).

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

Key*ű*

Required.

String value expected.

Updates cause replacement.

Value*ű*

Required.

String value expected.

Updates cause replacement.

RamDiskId*ű*

> **Note**
>
> Not implemented.

SecurityGroups*ű*

Security group names to assign.

List value expected.

Updates cause replacement.

UserData*ű*

User data to pass to instance.

String value expected.

Updates cause replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::AutoScaling::LaunchConfiguration
    properties:
      BlockDeviceMappings: [{"DeviceName": String, "Ebs": {
↪"DeleteOnTermination": Boolean, "SnapshotId": String, "VolumeSize": String}}
↪, {"DeviceName": String, "Ebs": {"DeleteOnTermination": Boolean, "SnapshotId
↪": String, "VolumeSize": String}}, ...]
      ImageId: String
      InstanceId: String
      InstanceType: String
      KeyName: String
      NovaSchedulerHints: [{"Key": String, "Value": String}, {"Key": String,
↪"Value": String}, ...]
      SecurityGroups: [Value, Value, ...]
      UserData: String
```

AWS::AutoScaling::ScalingPolicy

Required Properties

AdjustmentType*ű*

Type of adjustment (absolute or percentage).

String value expected.

Can be updated without replacement.

Allowed values: ChangeInCapacity, ExactCapacity, PercentChangeInCapacity

AutoScalingGroupName*ű*

AutoScaling group name to apply policy to.

String value expected.

Updates cause replacement.

ScalingAdjustment*ű*

Size of adjustment.

Integer value expected.

Can be updated without replacement.

Optional Properties

Cooldown*ű*

Cooldown period, in seconds.

Integer value expected.

Can be updated without replacement.

MinAdjustmentStep*ű*

Minimum number of resources that are added or removed when the AutoScaling group scales up or down. This can be used only when specifying PercentChangeInCapacity for the AdjustmentType property.

Integer value expected.

Can be updated without replacement.

The value must be at least 0.

Attributes

AlarmUrl*ű*
: A signed url to handle the alarm. (Heat extension).

show*ű*
: Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::AutoScaling::ScalingPolicy
    properties:
      AdjustmentType: String
      AutoScalingGroupName: String
      Cooldown: Integer
```

(continues on next page)

(continued from previous page)

```
      MinAdjustmentStep: Integer
      ScalingAdjustment: Integer
```

AWS::CloudFormation::Stack

Represents a child stack to allow composition of templates.

Required Properties

TemplateURL*ű*

The URL of a template that specifies the stack to be created as a resource.

String value expected.

Can be updated without replacement.

Optional Properties

Parameters*ű*

The set of parameters passed to this nested stack.

Map value expected.

Can be updated without replacement.

TimeoutInMinutes*ű*

The length of time, in minutes, to wait for the nested stack creation.

Integer value expected.

Can be updated without replacement.

Attributes

show*ű*
: Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::CloudFormation::Stack
    properties:
      Parameters: {...}
      TemplateURL: String
      TimeoutInMinutes: Integer
```

AWS::CloudFormation::WaitCondition

Available since 2014.1 (Icehouse)

Required Properties

Handle*ŕ*

A reference to the wait condition handle used to signal this wait condition.

String value expected.

Updates cause replacement.

Timeout*ŕ*

The number of seconds to wait for the correct number of signals to arrive.

Integer value expected.

Updates cause replacement.

The value must be in the range 1 to 43200.

Optional Properties

Count*ŕ*

The number of success signals that must be received before the stack creation process continues.

Integer value expected.

Can be updated without replacement.

Defaults to 1

The value must be at least 1.

Attributes

Data*ŕ*
: JSON string containing data associated with wait condition signals sent to the handle.

show*ŕ*
: Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::CloudFormation::WaitCondition
    properties:
      Count: Integer
```

(continues on next page)

(continued from previous page)

```
    Handle: String
    Timeout: Integer
```

AWS::CloudFormation::WaitConditionHandle

Available since 2014.1 (Icehouse)

AWS WaitConditionHandle resource.

the main point of this class is to : have no dependencies (so the instance can reference it) generate a unique url (to be returned in the reference) then the cfn-signal will use this url to post to and WaitCondition will poll it to see if has been written to.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::CloudFormation::WaitConditionHandle
```

AWS::EC2::EIP

Optional Properties

InstanceId*ű*

Instance ID to associate with EIP.

String value expected.

Can be updated without replacement.

Value must be of type nova.server

Domain*ű*

DEPRECATED since 9.0.0 (Pike) - Now we only allow vpc here, so no need to set up this tag anymore.

Set to vpc to have IP address allocation associated to your VPC.

String value expected.

Updates cause replacement.

Allowed values: vpc

Attributes

AllocationId*ű*

ID that AWS assigns to represent the allocation of the address for use with Amazon VPC. Returned only for VPC elastic IP addresses.

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::EIP
    properties:
      InstanceId: String
```

AWS::EC2::EIPAssociation

Optional Properties

AllocationId*ű*

Allocation ID for VPC EIP address.

String value expected.

Can be updated without replacement.

EIP*ű*

EIP address to associate with instance.

String value expected.

Can be updated without replacement.

Value must be of type ip_addr

InstanceId*ű*

Instance ID to associate with EIP specified by EIP property.

String value expected.

Can be updated without replacement.

Value must be of type nova.server

NetworkInterfaceId*ű*

Network interface ID to associate with EIP.

String value expected.

Can be updated without replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::EIPAssociation
    properties:
      AllocationId: String
      EIP: String
      InstanceId: String
      NetworkInterfaceId: String
```

AWS::EC2::Instance

Required Properties

ImageId*ű*

Glance image ID or name.

String value expected.

Updates cause replacement.

Value must be of type glance.image

InstanceType*ű*

Nova instance type (flavor).

String value expected.

Can be updated without replacement.

Value must be of type nova.flavor

Optional Properties

AvailabilityZone*ű*

Availability zone to launch the instance in.

String value expected.

Updates cause replacement.

BlockDeviceMappings*ű*

Block device mappings to attach to instance.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

DeviceName*ří*

Required.

A device name where the volume will be attached in the system at /dev/device_name.e.g. vdb

String value expected.

Updates cause replacement.

Ebs*ří*

The ebs volume to attach to the instance.

Map value expected.

Updates cause replacement.

Map properties:

DeleteOnTermination*ří*

Optional.

Indicate whether the volume should be deleted when the instance is terminated.

Boolean value expected.

Updates cause replacement.

Defaults to `true`

Iops*ří*

Not implemented.

SnapshotId*ří*

Optional.

The ID of the snapshot to create a volume from.

String value expected.

Updates cause replacement.

Value must be of type cinder.snapshot

VolumeSize*ří*

Optional.

The size of the volume, in GB. Must be equal or greater than the size of the snapshot. It is safe to leave this blank and have the Compute service infer the size.

String value expected.

Updates cause replacement.

VolumeType*ű*

Not implemented.

NoDevice*ű*

Not implemented.

VirtualName*ű*

Not implemented.

DisableApiTermination*ű*

> **Note**
>
> Not implemented.

KernelId*ű*

> **Note**
>
> Not implemented.

KeyName*ű*

Optional Nova keypair name.

String value expected.

Updates cause replacement.

Value must be of type nova.keypair

Monitoring*ű*

> **Note**
>
> Not implemented.

NetworkInterfaces*ű*

Network interfaces to associate with instance.

List value expected.

Can be updated without replacement.

NovaSchedulerHints*ű*

Scheduler hints to pass to Nova (Heat extension).

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

Key*ű*

Required.

String value expected.

Updates cause replacement.

Value*ű*

Required.

String value expected.

Updates cause replacement.

PlacementGroupName*ű*

> **Note**
>
> Not implemented.

PrivateIpAddress*ű*

> **Note**
>
> Not implemented.

RamDiskId*ű*

> **Note**
>
> Not implemented.

SecurityGroupIds*ű*

Security group IDs to assign.

List value expected.

Updates cause replacement.

SecurityGroups*ű*

Security group names to assign.

List value expected.

Updates cause replacement.

SourceDestCheck*ű*

> **Note**
>
> Not implemented.

SubnetId*ű*

Subnet ID to launch instance in.

String value expected.

Can be updated without replacement.

Tags*ű*

Tags to attach to instance.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

Key*ű*

Required.

String value expected.

Can be updated without replacement.

Value*ű*

Required.

String value expected.

Can be updated without replacement.

Tenancy*ű*

> **Note**
>
> Not implemented.

UserData*ű*

User data to pass to instance.

String value expected.

Updates cause replacement.

Volumes*ű*

Volumes to attach to instance.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

Device*ǘ*

Required.

The device where the volume is exposed on the instance. This assignment may not be honored and it is advised that the path /dev/disk/by-id/virtio-<VolumeId> be used instead.

String value expected.

Updates cause replacement.

VolumeId*ǘ*

Required.

The ID of the volume to be attached.

String value expected.

Updates cause replacement.

Value must be of type cinder.volume

Attributes

AvailabilityZone*ǘ*
The Availability Zone where the specified instance is launched.

PrivateDnsName*ǘ*
Private DNS name of the specified instance.

PrivateIp*ǘ*
Private IP address of the specified instance.

PublicDnsName*ǘ*
Public DNS name of the specified instance.

PublicIp*ǘ*
Public IP address of the specified instance.

show*ǘ*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::Instance
    properties:
      AvailabilityZone: String
      BlockDeviceMappings: [{"DeviceName": String, "Ebs": {
↪"DeleteOnTermination": Boolean, "SnapshotId": String, "VolumeSize": String}}
↪, {"DeviceName": String, "Ebs": {"DeleteOnTermination": Boolean, "SnapshotId
↪": String, "VolumeSize": String}}, ...]
      ImageId: String
      InstanceType: String
      KeyName: String
      NetworkInterfaces: [Value, Value, ...]
      NovaSchedulerHints: [{"Key": String, "Value": String}, {"Key": String,
↪"Value": String}, ...]
      SecurityGroupIds: [Value, Value, ...]
      SecurityGroups: [Value, Value, ...]
      SubnetId: String
      Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
      UserData: String
      Volumes: [{"Device": String, "VolumeId": String}, {"Device": String,
↪"VolumeId": String}, ...]
```

AWS::EC2::InternetGateway

Optional Properties

Tags*ű*

List value expected.

Updates cause replacement.

List contents:

Not implemented.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
```

(continues on next page)

(continued from previous page)

```
resources:
  ...
  the_resource:
    type: AWS::EC2::InternetGateway
    properties:
      Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
```

AWS::EC2::NetworkInterface

Required Properties

SubnetId*ú*

Subnet ID to associate with this interface.

String value expected.

Updates cause replacement.

Value must be of type neutron.subnet

Optional Properties

Description*ú*

Description for this interface.

String value expected.

Updates cause replacement.

GroupSet*ú*

List of security group IDs associated with this interface.

List value expected.

Can be updated without replacement.

PrivateIpAddress*ú*

String value expected.

Updates cause replacement.

SourceDestCheck*ú*

> **Note**
>
> Not implemented.

Tags*ú*

List value expected.

Updates cause replacement.

List contents:

Not implemented.

Attributes

PrivateIpAddressű
Private IP address of the network interface.

showű
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::NetworkInterface
    properties:
      Description: String
      GroupSet: [Value, Value, ...]
      PrivateIpAddress: String
      SubnetId: String
      Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
```

AWS::EC2::RouteTable

Available since 2014.1 (Icehouse)

Required Properties

VpcIdű

VPC ID for where the route table is created.

String value expected.

Updates cause replacement.

Optional Properties

Tagsű

List value expected.

Updates cause replacement.

List contents:

Not implemented.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::RouteTable
    properties:
      Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
      VpcId: String
```

AWS::EC2::SecurityGroup

Required Properties

GroupDescription*ű*

Description of the security group.

String value expected.

Updates cause replacement.

Optional Properties

SecurityGroupEgress*ű*

List value expected.

Can be updated without replacement.

List contents:

List of security group egress rules.

Map value expected.

Can be updated without replacement.

Map properties:

CidrIp*ű*

Optional.

String value expected.

Can be updated without replacement.

FromPort*ű*

Optional.

String value expected.

Can be updated without replacement.

IpProtocol*ű*

Optional.

String value expected.

Can be updated without replacement.

SourceSecurityGroupId*ű*

Optional.

String value expected.

Can be updated without replacement.

SourceSecurityGroupName*ű*

Optional.

String value expected.

Can be updated without replacement.

SourceSecurityGroupOwnerId*ű*

Not implemented.

ToPort*ű*

Optional.

String value expected.

Can be updated without replacement.

SecurityGroupIngress*ű*

List value expected.

Can be updated without replacement.

List contents:

List of security group ingress rules.

Map value expected.

Can be updated without replacement.

Map properties:

CidrIp*ű*

Optional.

String value expected.

Can be updated without replacement.

FromPort*ű*

Optional.

String value expected.

Can be updated without replacement.

IpProtocolűű

Optional.

String value expected.

Can be updated without replacement.

SourceSecurityGroupIdűű

Optional.

String value expected.

Can be updated without replacement.

SourceSecurityGroupNameűű

Optional.

String value expected.

Can be updated without replacement.

SourceSecurityGroupOwnerIdűű

Not implemented.

ToPortűű

Optional.

String value expected.

Can be updated without replacement.

VpcIdűű

Physical ID of the VPC. Not implemented.

String value expected.

Updates cause replacement.

Attributes

showűű

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::SecurityGroup
    properties:
```

(continues on next page)

(continued from previous page)

```
    GroupDescription: String
    SecurityGroupEgress: [{"CidrIp": String, "FromPort": String, "ToPort":␣
↪String, "IpProtocol": String, "SourceSecurityGroupId": String,
↪"SourceSecurityGroupName": String}, {"CidrIp": String, "FromPort": String,
↪"ToPort": String, "IpProtocol": String, "SourceSecurityGroupId": String,
↪"SourceSecurityGroupName": String}, ...]
    SecurityGroupIngress: [{"CidrIp": String, "FromPort": String, "ToPort":␣
↪String, "IpProtocol": String, "SourceSecurityGroupId": String,
↪"SourceSecurityGroupName": String}, {"CidrIp": String, "FromPort": String,
↪"ToPort": String, "IpProtocol": String, "SourceSecurityGroupId": String,
↪"SourceSecurityGroupName": String}, ...]
    VpcId: String
```

AWS::EC2::Subnet

Required Properties

CidrBlock*ű*

CIDR block to apply to subnet.

String value expected.

Updates cause replacement.

VpcId*ű*

Ref structure that contains the ID of the VPC on which you want to create the subnet.

String value expected.

Updates cause replacement.

Optional Properties

AvailabilityZone*ű*

Availability zone in which you want the subnet.

String value expected.

Updates cause replacement.

Tags*ű*

List value expected.

Updates cause replacement.

List contents:

Not implemented.

Attributes

AvailabilityZone*ű*
Availability Zone of the subnet.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::Subnet
    properties:
      AvailabilityZone: String
      CidrBlock: String
      Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
      VpcId: String
```

AWS::EC2::SubnetRouteTableAssociation

Required Properties

RouteTableId*ű*

Route table ID.

String value expected.

Updates cause replacement.

SubnetId*ű*

Subnet ID.

String value expected.

Updates cause replacement.

Value must be of type neutron.subnet

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
```

(continues on next page)

(continued from previous page)

```
  ...
  the_resource:
    type: AWS::EC2::SubnetRouteTableAssociation
    properties:
      RouteTableId: String
      SubnetId: String
```

AWS::EC2::VPC

Optional Properties

CidrBlock*ű*

CIDR block to apply to the VPC.

String value expected.

Updates cause replacement.

InstanceTenancy*ű*

> **Note**
>
> Not implemented.

Tags*ű*

List value expected.

Updates cause replacement.

List contents:

Not implemented.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::VPC
    properties:
      CidrBlock: String
      Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
```

AWS::EC2::VPCGatewayAttachment

Required Properties

VpcId*ŕ*

VPC ID for this gateway association.

String value expected.

Updates cause replacement.

Optional Properties

InternetGatewayId*ŕ*

ID of the InternetGateway.

String value expected.

Updates cause replacement.

VpnGatewayId*ŕ*

> **Note**
>
> Not implemented.

Attributes

show*ŕ*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::VPCGatewayAttachment
    properties:
      InternetGatewayId: String
      VpcId: String
```

AWS::EC2::Volume

Required Properties

AvailabilityZone*ŕ*

The availability zone in which the volume will be created.

String value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

Optional Properties

Size*ű*

The size of the volume in GB.

Integer value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

The value must be at least 1.

SnapshotId*ű*

If specified, the backup used as the source to create the volume.

String value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

Value must be of type cinder.backup

Tags*ű*

The list of tags to associate with the volume.

List value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

List contents:

Map value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

Map properties:

Key*ű*

Required.

String value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

Value*ű*

Required.

String value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::Volume
    properties:
      AvailabilityZone: String
      Size: Integer
      SnapshotId: String
      Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
```

AWS::EC2::VolumeAttachment

Required Properties

Device*ŕ*

The device where the volume is exposed on the instance. This assignment may not be honored and it is advised that the path /dev/disk/by-id/virtio-<VolumeId> be used instead.

String value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

Value must match pattern: /dev/vd[b-z]

InstanceId*ŕ*

The ID of the instance to which the volume attaches.

String value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

Value must be of type nova.server

VolumeId*ŕ*

The ID of the volume to be attached.

String value expected.

Updates are not supported. Resource update will fail on any attempt to update this property.

Value must be of type cinder.volume

Attributes

show*ŕ*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::EC2::VolumeAttachment
    properties:
      Device: String
      InstanceId: String
      VolumeId: String
```

AWS::ElasticLoadBalancing::LoadBalancer

Implements a HAProxy-bearing instance as a nested stack.

The template for the nested stack can be redefined with `loadbalancer_template` option in `heat.conf`.

Generally the image used for the instance must have the following packages installed or available for installation at runtime:

```
- heat-cfntools and its dependencies like python-psutil
- cronie
- socat
- haproxy
```

Current default builtin template uses Fedora 21 x86_64 base cloud image (https://getfedora.org/cloud/download/) and apart from installing packages goes through some hoops around SELinux due to peculiarities of heat-cfntools.

Required Properties

AvailabilityZones*ű*

The Availability Zones in which to create the load balancer.

List value expected.

Updates cause replacement.

Listeners*ű*

One or more listeners for this load balancer.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

InstancePort*ű*

Required.

TCP port on which the instance server is listening.

Integer value expected.

Updates cause replacement.

LoadBalancerPort*ú*

Required.

The external load balancer port number.

Integer value expected.

Updates cause replacement.

PolicyNames*ú*

Not implemented.

Protocol*ú*

Required.

The load balancer transport protocol to use.

String value expected.

Updates cause replacement.

Allowed values: TCP, HTTP

SSLCertificateId*ú*

Not implemented.

Optional Properties

AppCookieStickinessPolicy*ú*

> **Note**
>
> Not implemented.

HealthCheck*ú*

An application health check for the instances.

Map value expected.

Updates cause replacement.

Map properties:

HealthyThreshold*ú*

Required.

The number of consecutive health probe successes required before moving the instance to the healthy state.

Integer value expected.

Updates cause replacement.

Interval*ú*

Required.

The approximate interval, in seconds, between health checks of an individual instance.

Integer value expected.

Updates cause replacement.

Target*ú*

Required.

The port being checked.

String value expected.

Updates cause replacement.

Timeout*ú*

Required.

Health probe timeout, in seconds.

Integer value expected.

Updates cause replacement.

UnhealthyThreshold*ú*

Required.

The number of consecutive health probe failures required before moving the instance to the unhealthy state

Integer value expected.

Updates cause replacement.

Instances*ú*

The list of instance IDs load balanced.

List value expected.

Can be updated without replacement.

LBCookieStickinessPolicy*ú*

> **Note**
>
> Not implemented.

SecurityGroups*ú*

List of Security Groups assigned on current LB.

List value expected.

Can be updated without replacement.

Subnets*ű*

> **Note**
>
> Not implemented.

Attributes

CanonicalHostedZoneName*ű*
The name of the hosted zone that is associated with the LoadBalancer.

CanonicalHostedZoneNameID*ű*
The ID of the hosted zone name that is associated with the LoadBalancer.

DNSName*ű*
The DNS name for the LoadBalancer.

SourceSecurityGroup.GroupName*ű*
The security group that you can use as part of your inbound rules for your LoadBalancers back-end instances.

SourceSecurityGroup.OwnerAlias*ű*
Owner of the source security group.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::ElasticLoadBalancing::LoadBalancer
    properties:
      AvailabilityZones: [Value, Value, ...]
      HealthCheck: {"HealthyThreshold": Integer, "Interval": Integer, "Target
↪": String, "Timeout": Integer, "UnhealthyThreshold": Integer}
      Instances: [Value, Value, ...]
      Listeners: [{"InstancePort": Integer, "LoadBalancerPort": Integer,
↪"Protocol": String}, {"InstancePort": Integer, "LoadBalancerPort": Integer,
↪"Protocol": String}, ...]
      SecurityGroups: [Value, Value, ...]
```

AWS::IAM::AccessKey

Required Properties

UserName*ű*

The name of the user that the new key will belong to.

String value expected.

Updates cause replacement.

Optional Properties

Serial*ű*

> **Note**
>
> Not implemented.

Status*ű*

> **Note**
>
> Not implemented.

Attributes

SecretAccessKey*ű*
Keypair secret key.

UserName*ű*
Username associated with the AccessKey.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::IAM::AccessKey
    properties:
      UserName: String
```

AWS::IAM::User

Optional Properties

Groups*ű*

Not Implemented.

List value expected.

Updates cause replacement.

LoginProfile*ű*

A login profile for the user.

Map value expected.

Updates cause replacement.

Map properties:

Password*ű*

Optional.

String value expected.

Updates cause replacement.

Path*ű*

Not Implemented.

String value expected.

Updates cause replacement.

Policies*ű*

Access policies to apply to the user.

List value expected.

Updates cause replacement.

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::IAM::User
    properties:
      Groups: [Value, Value, ...]
      LoginProfile: {"Password": String}
      Path: String
      Policies: [Value, Value, ...]
```

AWS::S3::Bucket

Optional Properties

AccessControl*ű*

A predefined access control list (ACL) that grants permissions on the bucket.

String value expected.

Updates cause replacement.

Allowed values: Private, PublicRead, PublicReadWrite, AuthenticatedRead, BucketOwnerRead, BucketOwnerFullControl

Tags*ű*

Tags to attach to the bucket.

List value expected.

Updates cause replacement.

List contents:

Map value expected.

Updates cause replacement.

Map properties:

Key*ű*

Required.

The tag key name.

String value expected.

Updates cause replacement.

Value*ű*

Required.

The tag value.

String value expected.

Updates cause replacement.

WebsiteConfiguration*ű*

Information used to configure the bucket as a static website.

Map value expected.

Updates cause replacement.

Map properties:

ErrorDocument*ű*

Optional.

The name of the error document.

String value expected.

Updates cause replacement.

IndexDocument*ű*

Optional.

The name of the index document.

String value expected.

Updates cause replacement.

Attributes

DomainName*ű*
The DNS name of the specified bucket.

WebsiteURL*ű*
The website endpoint for the specified bucket.

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: AWS::S3::Bucket
    properties:
      AccessControl: String
      Tags: [{"Key": String, "Value": String}, {"Key": String, "Value":␣
↪String}, ...]
      WebsiteConfiguration: {"IndexDocument": String, "ErrorDocument": String}
```

Unsupported Heat Resource Types

These resources are enabled, but are not officially supported.

OS::Aodh::Alarm

DEPRECATED since 10.0.0 (Queens) - Theshold alarm relies on ceilometer-api and has been deprecated in aodh since Ocata. Use OS::Aodh::GnocchiAggregationByResourcesAlarm instead.

Available since 2014.1 (Icehouse)

A resource that implements alarming service of Aodh.

A resource that allows for the setting alarms based on threshold evaluation for a collection of samples. Also, you can define actions to take if state of watched resource will be satisfied specified conditions. For example, it can watch for the memory consumption and when it reaches 70% on a given instance if the instance has been up for more than 10 min, some action will be called.

Required Properties

meter_name*ű*

Meter name watched by the alarm.

String value expected.

Updates cause replacement.

threshold*ú*

Threshold to evaluate against.

Number value expected.

Can be updated without replacement.

Optional Properties

alarm_actions*ú*

A list of URLs (webhooks) to invoke when state transitions to alarm.

List value expected.

Can be updated without replacement.

alarm_queues*ú*

> Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to alarm.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

> Optional.
>
> String value expected.
>
> Can be updated without replacement.
>
> Value must be of type zaqar.queue

comparison_operator*ú*

Operator used to compare specified statistic with threshold.

String value expected.

Can be updated without replacement.

Allowed values: le, ge, eq, lt, gt, ne

description*ú*

Description for the alarm.

String value expected.

Can be updated without replacement.

enabled*ú*

True if alarm evaluation/actioning is enabled.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

evaluation_periods*ří*

Number of periods to evaluate over.

Integer value expected.

Can be updated without replacement.

insufficient_data_actions*ří*

A list of URLs (webhooks) to invoke when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

insufficient_data_queues*ří*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to insufficient-data.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

matching_metadata*ří*

Meter should match this resource metadata (key=value) additionally to the meter_name.

Map value expected.

Can be updated without replacement.

Defaults to `{}`

ok_actions*ří*

A list of URLs (webhooks) to invoke when state transitions to ok.

List value expected.

Can be updated without replacement.

ok_queues*ří*

Available since 8.0.0 (Ocata)

A list of Zaqar queues to post to when state transitions to ok.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

String value expected.

Can be updated without replacement.

Value must be of type zaqar.queue

period*ű*

Period (seconds) to evaluate over.

Integer value expected.

Can be updated without replacement.

query*ű*

Available since 2015.1 (Kilo)

A list of query factors, each comparing a Sample attribute with a value. Implicitly combined with matching_metadata, if any.

List value expected.

Can be updated without replacement.

List contents:

Map value expected.

Can be updated without replacement.

Map properties:

field*ű*

Optional.

Name of attribute to compare. Names of the form metadata.user_metadata.X or metadata.metering.X are equivalent to what you can address through matching_metadata; the former for Nova meters, the latter for all others. To see the attributes of your Samples, use ‘ceilometer debug sample-list‘.

String value expected.

Can be updated without replacement.

op*ű*

Optional.

Comparison operator.

String value expected.

Can be updated without replacement.

Allowed values: le, ge, eq, lt, gt, ne

type*ŕ*

Available since 8.0.0 (Ocata)

Optional.

The type of the attribute.

String value expected.

Can be updated without replacement.

Defaults to `"string"`

Allowed values: integer, float, string, boolean, datetime

value*ŕ*

Optional.

String value with which to compare.

String value expected.

Can be updated without replacement.

repeat_actions*ŕ*

False to trigger actions when the threshold is reached AND the alarms state has changed. By default, actions are called each time the threshold is reached.

Boolean value expected.

Can be updated without replacement.

Defaults to `"true"`

severity*ŕ*

Available since 5.0.0 (Liberty)

Severity of the alarm.

String value expected.

Can be updated without replacement.

Defaults to `"low"`

Allowed values: low, moderate, critical

statistic*ŕ*

Meter statistic to evaluate.

String value expected.

Can be updated without replacement.

Allowed values: count, avg, sum, min, max

time_constraints*ű*

Available since 5.0.0 (Liberty)

Describe time constraints for the alarm. Only evaluate the alarm if the time at evaluation is within this time constraint. Start point(s) of the constraint are specified with a cron expression, whereas its duration is given in seconds.

List value expected.

Updates cause replacement.

Defaults to `[]`

List contents:

Map value expected.

Updates cause replacement.

Map properties:

description*ű*

Optional.

Description for the time constraint.

String value expected.

Updates cause replacement.

duration*ű*

Required.

Duration for the time constraint.

Integer value expected.

Updates cause replacement.

The value must be at least 0.

name*ű*

Required.

Name for the time constraint.

String value expected.

Updates cause replacement.

start*ű*

Required.

Start time for the time constraint. A CRON expression property.

String value expected.

Updates cause replacement.

Value must be of type cron_expression

timezone*ű*

Optional.

Timezone for the time constraint (eg. Asia/Taipei, Europe/Amsterdam).

String value expected.

Updates cause replacement.

Value must be of type timezone

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Aodh::Alarm
    properties:
      alarm_actions: [Value, Value, ...]
      alarm_queues: [String, String, ...]
      comparison_operator: String
      description: String
      enabled: Boolean
      evaluation_periods: Integer
      insufficient_data_actions: [Value, Value, ...]
      insufficient_data_queues: [String, String, ...]
      matching_metadata: {...}
      meter_name: String
      ok_actions: [Value, Value, ...]
      ok_queues: [String, String, ...]
      period: Integer
      query: [{"field": String, "type": String, "op": String, "value": String}
↪, {"field": String, "type": String, "op": String, "value": String}, ...]
      repeat_actions: Boolean
      severity: String
      statistic: String
      threshold: Number
      time_constraints: [{"name": String, "start": String, "description":␣
↪String, "duration": Integer, "timezone": String}, {"name": String, "start":␣
↪String, "description": String, "duration": Integer, "timezone": String}, ...
↪]
```

OS::Neutron::ExtraRoute

UNSUPPORTED - Use this resource at your own risk.

Resource for specifying extra routes for Neutron router.

Resource allows to specify nexthop IP and destination network for router.

Required Properties

destination*ű*

Network in CIDR notation.

String value expected.

Updates cause replacement.

nexthop*ű*

Nexthop IP address.

String value expected.

Updates cause replacement.

router_id*ű*

The router id.

String value expected.

Updates cause replacement.

Value must be of type neutron.router

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::ExtraRoute
    properties:
      destination: String
      nexthop: String
      router_id: String
```

OS::Neutron::FlowClassifier

UNSUPPORTED since 8.0.0 (Ocata)

Heat Template Resource for networking-sfc flow-classifier.

This resource used to select the traffic that can access the service chain. Traffic that matches any flow classifier will be directed to the first port in the chain.

Optional Properties

description*ří*

Description for the Flow Classifier.

String value expected.

Can be updated without replacement.

destination_ip_prefix*ří*

Destination IP prefix or subnet.

String value expected.

Updates cause replacement.

Value must be of type net_cidr

destination_port_range_max*ří*

Destination protocol port maximum.

Integer value expected.

Updates cause replacement.

The value must be in the range 1 to 65535.

destination_port_range_min*ří*

Destination protocol port minimum.

Integer value expected.

Updates cause replacement.

The value must be in the range 1 to 65535.

ethertype*ří*

L2 ethertype.

String value expected.

Updates cause replacement.

Defaults to `"IPv4"`

Allowed values: IPv4, IPv6

logical_destination_port*ří*

ID or name of the neutron destination port.

String value expected.

Updates cause replacement.

Value must be of type neutron.port

logical_source_port*ű*

ID or name of the neutron source port.

String value expected.

Updates cause replacement.

Value must be of type neutron.port

name*ű*

Name of the Flow Classifier.

String value expected.

Can be updated without replacement.

protocol*ű*

IP Protocol for the Flow Classifier.

String value expected.

Updates cause replacement.

Allowed values: tcp, udp, icmp

source_ip_prefix*ű*

Source IP prefix or subnet.

String value expected.

Updates cause replacement.

Value must be of type net_cidr

source_port_range_max*ű*

Source protocol port Maximum.

Integer value expected.

Updates cause replacement.

The value must be in the range 1 to 65535.

source_port_range_min*ű*

Source protocol port Minimum.

Integer value expected.

Updates cause replacement.

The value must be in the range 1 to 65535.

l7_parameters*ű*

UNSUPPORTED - Currently, no value is supported for this option.

Dictionary of L7-parameters.

Map value expected.

Updates cause replacement.

Attributes

show*ú*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::FlowClassifier
    properties:
      description: String
      destination_ip_prefix: String
      destination_port_range_max: Integer
      destination_port_range_min: Integer
      ethertype: String
      logical_destination_port: String
      logical_source_port: String
      name: String
      protocol: String
      source_ip_prefix: String
      source_port_range_max: Integer
      source_port_range_min: Integer
```

OS::Neutron::PortChain

UNSUPPORTED since 8.0.0 (Ocata)

A resource for neutron networking-sfc.

This resource used to define the service function path by arranging networking-sfc port-pair-groups and set of flow classifiers, to specify the classified traffic flows to enter the chain.

Required Properties

port_pair_groups*ú*

A list of port pair groups to apply to the Port Chain.

List value expected.

Can be updated without replacement.

List contents:

Optional.

Port Pair Group ID or Name .

String value expected.

Can be updated without replacement.

Value must be of type neutron.port_pair_group

Optional Properties

chain_parameters*ű*

Dictionary of chain parameters. Currently, only correlation=mpls is supported by default.

Map value expected.

Updates cause replacement.

Defaults to `{"correlation": "mpls"}`

description*ű*

Description for the Port Chain.

String value expected.

Can be updated without replacement.

flow_classifiers*ű*

A list of flow classifiers to apply to the Port Chain.

List value expected.

Can be updated without replacement.

Defaults to `[]`

List contents:

Optional.

Flow Classifier ID or Name .

String value expected.

Can be updated without replacement.

Value must be of type neutron.flow_classifier

name*ű*

Name of the Port Chain.

String value expected.

Can be updated without replacement.

Attributes

show*ú*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::PortChain
    properties:
      chain_parameters: {...}
      description: String
      flow_classifiers: [String, String, ...]
      name: String
      port_pair_groups: [String, String, ...]
```

OS::Neutron::PortPair

UNSUPPORTED since 7.0.0 (Newton)

A resource for neutron networking-sfc port-pair.

This plug-in requires networking-sfc>=1.0.0. So to enable this plug-in, install this library and restart the heat-engine.

A Port Pair represents a service function instance. The ingress port and the egress port of the service function may be specified. If a service function has one bidirectional port, the ingress port has the same value as the egress port.

Required Properties

egress*ú*

ID or name of the egress neutron port.

String value expected.

Updates cause replacement.

Value must be of type neutron.port

ingress*ú*

ID or name of the ingress neutron port.

String value expected.

Updates cause replacement.

Value must be of type neutron.port

Optional Properties

description*ű*

Description for the Port Pair.

String value expected.

Can be updated without replacement.

name*ű*

Name for the Port Pair.

String value expected.

Can be updated without replacement.

service_function_parameters*ű*

Dictionary of service function parameter. Currently only correlation=None is supported.

Map value expected.

Updates cause replacement.

Defaults to `{"correlation": null}`

Attributes

show*ű*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::PortPair
    properties:
      description: String
      egress: String
      ingress: String
      name: String
      service_function_parameters: {...}
```

OS::Neutron::PortPairGroup

UNSUPPORTED since 8.0.0 (Ocata)

Heat Template Resource for networking-sfc port-pair-group.

Multiple port-pairs may be included in a port-pair-group to allow the specification of a set of functionally equivalent Service Functions that can be used for load distribution.

Required Properties

port_pairs*ű*

A list of Port Pair IDs or names to apply.

List value expected.

Can be updated without replacement.

List contents:

Optional.

Port Pair ID or name .

String value expected.

Can be updated without replacement.

Value must be of type neutron.port_pair

Optional Properties

description*ű*

Description for the Port Pair Group.

String value expected.

Can be updated without replacement.

name*ű*

Name for the Port Pair Group.

String value expected.

Can be updated without replacement.

Attributes

show*ű*

Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Neutron::PortPairGroup
    properties:
      description: String
      name: String
      port_pairs: [String, String, ...]
```

OS::Vitrage::Template

DEPRECATED since 26.0.0 - Vitrage project was marked inactive

Available since 16.0.0 (Wallaby)

A resource for managing Vitrage templates.

A Vitrage template defines conditions and actions, based on the Vitrage topology graph. For example, if there is an instance down alarm on an instance, then execute a Mistral healing workflow.

The VitrageTemplate resource generates and adds to Vitrage a template based on the input parameters.

Required Properties

template_file*ú*

Path of the Vitrage template to use.

String value expected.

Updates cause replacement.

template_params*ú*

Input parameters for the Vitrage template.

Map value expected.

Updates cause replacement.

Attributes

show*ú*
Detailed information about resource.

HOT Syntax

```
heat_template_version: 2015-04-30
...
resources:
  ...
  the_resource:
    type: OS::Vitrage::Template
    properties:
      template_file: String
      template_params: {...}
```

Contributed Heat Resource Types

These resources are not enabled by default.

DockerInc Resource

This resource is not enabled by default.

This plugin enables the use of Docker containers in a Heat template and requires the docker-py package. You can find more information in the DOCKER_README.

CloudFormation Compatible Functions

There are a number of functions that you can use to help you write CloudFormation compatible templates. While most CloudFormation functions are supported in HOT version 2013-05-23, *Fn::Select* is the only CloudFormation function supported in HOT templates since version 2014-10-16 which is introduced in Juno.

All of these functions (except *Ref*) start with *Fn::*.

Ref

Returns the value of the named parameter or resource.

Parameters

name
: [String] The name of the resource or parameter.

Usage

```
{Ref: my_server}
```

Returns the nova instance ID. For example, `d8093de0-850f-4513-b202-7979de6c0d55`.

Fn::Base64

This is a placeholder for a function to convert an input string to Base64. This function in Heat actually performs no conversion. It is included for the benefit of CFN templates that convert UserData to Base64. Heat only accepts UserData in plain text.

Parameters

value
: [String] The string to convert.

Usage

```
{"Fn::Base64": "convert this string please."}
```

Returns the original input string.

Fn::FindInMap

Returns the value corresponding to keys into a two-level map declared in the Mappings section.

Parameters

map_name
[String] The logical name of a mapping declared in the Mappings section that contains the keys and values.

top_level_key
[String] The top-level key name. Its value is a list of key-value pairs.

second_level_key
[String] The second-level key name, which is set to one of the keys from the list assigned to top_level_key.

Usage

```
Mapping:
  MyContacts:
    jone: {phone: 337, email: a@b.com}
    jim: {phone: 908, email: g@b.com}

{"Fn::FindInMap": ["MyContacts", "jim", "phone" ] }
```

Returns `908`.

Fn::GetAtt

Returns an attribute of a resource within the template.

Parameters

resource
[String] The name of the resource.

attribute
[String] The name of the attribute.

Usage

```
{Fn::GetAtt: [my_server, PublicIp]}
```

Returns an IP address such as `10.0.0.2`.

Fn::GetAZs

Returns the Availability Zones within the given region.

Note: AZs and regions are not fully implemented in Heat.

Parameters

region
[String] The name of the region.

Usage

```
{Fn::GetAZs: ""}
```

Returns the list provided by `nova availability-zone-list`.

Fn::Join

Like python join, it joins a list of strings with the given delimiter.

Parameters

delimiter
: [String] The string to join the list with.

list
: [list] The list to join.

Usage

```
{Fn::Join: [",", ["beer", "wine", "more beer"]]}
```

Returns `beer, wine, more beer`.

Fn::Select

Select an item from a list.

Heat extension: Select an item from a map

Parameters

selector
: [string or integer] The number of item in the list or the name of the item in the map.

collection
: [map or list] The collection to select the item from.

Usage

For a list lookup:

```
{ "Fn::Select" : [ "2", [ "apples", "grapes", "mangoes" ] ] }
```

Returns `mangoes`.

For a map lookup:

```
{ "Fn::Select" : [ "red", {"red": "a", "flu": "b"} ] }
```

Returns `a`.

Fn::Split

This is the reverse of Join. Convert a string into a list based on the delimiter.

Parameters

delimiter
[string] Matching string to split on.

string
[String] The string to split.

Usage

```
{ "Fn::Split" : [ ",", "str1,str2,str3,str4"]}
```

Returns `{["str1", "str2", "str3", "str4"]}`.

Fn::Replace

Find and replace one string with another.

Parameters

substitutions
[map] A map of substitutions.

string: String
The string to do the substitutions in.

Usage

```
{"Fn::Replace": [
 {'$var1': 'foo', '%var2%': 'bar'},
  '$var1 is %var2%'
]}
```

Returns `"foo is bar"`.

Fn::ResourceFacade

When writing a Template Resource:

- user writes a template that will fill in for a resource (the resource is the facade).
- when they are writing their template they need to access the metadata from the facade.

Parameters

attribute_name
[String] One of `Metadata`, `DeletionPolicy` or `UpdatePolicy`.

Usage

```
{'Fn::ResourceFacade': 'Metadata'}
{'Fn::ResourceFacade': 'DeletionPolicy'}
{'Fn::ResourceFacade': 'UpdatePolicy'}
```

Example

Here is a top level template `top.yaml`

```
resources:
  my_server:
    type: OS::Nova::Server
    metadata:
      key: value
      some: more stuff
```

Here is a resource template `my_actual_server.yaml`

```
resources:
  _actual_server_:
    type: OS::Nova::Server
    metadata: {'Fn::ResourceFacade': Metadata}
```

The environment file `env.yaml`

```
resource_registry:
  resources:
    my_server:
      "OS::Nova::Server": my_actual_server.yaml
```

To use it

```
$ openstack stack create -t top.yaml -e env.yaml mystack
```

What happened is the metadata in `top.yaml` (key: value, some: more stuff) gets passed into the resource template via the *Fn::ResourceFacade* function.

Fn::MemberListToMap

Convert an AWS style member list into a map.

Parameters

key name: string
The name of the key (normally Name or Key).

value name: string
The name of the value (normally Value).

list: A list of strings
The string to convert.

Usage

```
{'Fn::MemberListToMap': ['Name', 'Value', ['.member.0.Name=key',
                                           '.member.0.Value=door',
                                           '.member.1.Name=colour',
                                           '.member.1.Value=green']]}
```

Returns `{'key': 'door', 'colour': 'green'}`.

Fn::Equals

Compares whether two values are equal. And returns true if the two values are equal or false if they arent.

Parameters

value1:
A value of any type that you want to compare.

value2:
A value of any type that you want to compare.

Usage

```
{'Fn::Equals': [{'Ref': 'env_type'}, 'prod']}
```

Returns true if the param env_type equals to prod, otherwise returns false.

Fn::If

Returns one value if the specified condition evaluates to true and another value if the specified condition evaluates to false.

Parameters

condition_name:
A reference to a condition in the `Conditions` section.

value_if_true:
A value to be returned if the specified condition evaluates to true.

value_if_false:
A value to be returned if the specified condition evaluates to false.

Usage

```
{'Fn::If': ['create_prod', 'value_true', 'value_false']}
```

Returns value_true if the condition create_prod evaluates to true, otherwise returns value_false.

Fn::Not

Acts as a NOT operator.

The syntax of the `Fn::Not` function is

```
{'Fn::Not': [condition]}
```

Returns true for a condition that evaluates to false or returns false for a condition that evaluates to true.

Parameters

condition:
A condition such as `Fn::Equals` that evaluates to true or false can be defined in this function, also we can set a boolean value as a condition.

Usage

```
{'Fn::Not': [{'Fn::Equals': [{'Ref': env_type'}, 'prod']}]}
```

Returns false if the param env_type equals to prod, otherwise returns true.

Fn::And

Acts as an AND operator to evaluate all the specified conditions. Returns true if all the specified conditions evaluate to true, or returns false if any one of the conditions evaluates to false.

Parameters

condition:
A condition such as Fn::Equals that evaluates to true or false.

Usage

```
{'Fn::And': [{'Fn::Equals': [{'Ref': env_type}, 'prod']},
             {'Fn::Not': [{'Fn::Equals': [{'Ref': zone}, 'beijing']}]}]
```

Returns true if the param env_type equals to prod and the param zone is not equal to beijing, otherwise returns false.

Fn::Or

Acts as an OR operator to evaluate all the specified conditions. Returns true if any one of the specified conditions evaluate to true, or returns false if all of the conditions evaluates to false.

Parameters

condition:
A condition such as Fn::Equals that evaluates to true or false.

Usage

```
{'Fn::Or': [{'Fn::Equals': [{'Ref': zone}, 'shanghai']},
            {'Fn::Equals': [{'Ref': zone}, 'beijing']}]}
```

Returns true if the param zone equals to shanghai or beijing, otherwise returns false.

3.3.2 Example Templates

This page documents the templates at https://opendev.org/openstack/heat-templates/

Example HOT Templates

Hello World HOT Template

https://opendev.org/openstack/heat-templates/src/branch/master/hot/hello_world.yaml

Description

Hello world HOT template that just defines a single compute instance. Contains just base features to verify base HOT support.

Parameters

key_name **(required)**

type
string

description
Name of an existing key pair to use for the instance

flavor **(optional)**

type
string

description
Flavor for the instance to be created

image **(required)**

type
string

description
Image *ID* or image name to use for the instance

admin_pass **(required)**

type
string

description
The admin password for the instance

db_port **(optional)**

type
number

description
The database port number

Example CFN Templates

AWS Wordpress Single Instance Template

https://opendev.org/openstack/heat-templates/src/branch/master/cfn/F18/WordPress_Single_Instance.template

Description

AWS CloudFormation Sample Template WordPress_Single_Instance: WordPress is web software you can use to create a beautiful website or blog. This template installs a single-instance WordPress deployment using a local MySQL database to store the data.

Parameters

KeyName `(required)`

type
string

description
Name of an existing EC2 KeyPair to enable SSH access to the instance

InstanceType `(optional)`

type
string

description
The EC2 instance type

DBName `(optional)`

type
string

description
The WordPress database name

DBUsernameName `(optional)`

type
string

description
The WordPress database admin account username

DBPassword `(optional)`

type
string

description
The WordPress database admin account password

DBRootPassword `(optional)`

type
string

description
Root password for MySQL

LinuxDistribution `(optional)`

type
string

description
Linux distribution of choice

3.4 Using the Heat Service

- OpenStack Orchestration API v1 Reference
- Python and CLI client

CHAPTER

FOUR

DEVELOPING HEAT

4.1 Heat Developer Guidelines

In the developer guide, you will find documented policies for developing heat. This includes the processes we use for stories (for bugs and features), contributor onboarding, core reviewer memberships, and other procedural items.

Note

This guideline also includes documentation for developers.

4.1.1 Heat and DevStack

Heat is fully integrated into DevStack. This is a convenient way to try out or develop heat alongside the current development state of all the other OpenStack projects. Heat on DevStack works on both Ubuntu and Fedora.

These instructions assume you already have a working DevStack installation which can launch basic instances.

Configure DevStack to enable heat

Heat is configured by default on devstack for Icehouse and Juno releases.

Newer versions of OpenStack require enabling heat services in devstack *local.conf*. Add the following to *[[local|localrc]]* section of *local.conf*:

```
[[local|localrc]]

#Enable heat services
enable_service h-eng h-api h-api-cfn
```

Since Newton release, heat is available as a devstack plugin. To enable the plugin add the following to the *[[local|localrc]]* section of *local.conf*:

```
[[local|localrc]]

#Enable heat plugin
enable_plugin heat https://opendev.org/openstack/heat
```

To use stable branches, make sure devstack is on that branch, and specify the branch name to enable_plugin, for example:

```
enable_plugin heat https://opendev.org/openstack/heat stable/newton
```

It would also be useful to automatically download and register a VM image that heat can launch. To do that add the following to *[[local|localrc]]* section of *local.conf*:

```
IMAGE_URL_SITE="https://download.fedoraproject.org"
IMAGE_URL_PATH="/pub/fedora/linux/releases/37/Cloud/x86_64/images/"
IMAGE_URL_FILE="Fedora-Cloud-Base-37-1.7.x86_64.qcow2"
IMAGE_URLS+=","$IMAGE_URL_SITE$IMAGE_URL_PATH$IMAGE_URL_FILE
```

URLs for any cloud image may be specified, but fedora images from F20 contain the heat-cfntools package which is required for some heat functionality.

That is all the configuration that is required. When you run *./stack.sh* the heat processes will be launched in *screen* with the labels prefixed with *h-*.

Configure DevStack to enable ceilometer and aodh (if using alarms)

To use aodh alarms you need to enable ceilometer and aodh in devstack. Adding the following lines to *[[local|localrc]]* section of *local.conf* will enable the services:

```
CEILOMETER_BACKENDS=gnocchi
enable_plugin ceilometer https://opendev.org/openstack/ceilometer
enable_plugin aodh https://opendev.org/openstack/aodh
```

Configure DevStack to enable OSprofiler

Adding the following line to *[[local|localrc]]* section of *local.conf* will add the profiler notifier to your ceilometer:

```
CEILOMETER_NOTIFICATION_TOPICS=notifications,profiler
```

Enable the profiler in /etc/heat/heat.conf:

```
$ echo -e "[profiler]\nenabled = True\n"\
"trace_sqlalchemy = True\n"\
"hmac_keys = SECRET_KEY\n"\
>> /etc/heat/heat.conf
```

Run any command with profile SECRET_KEY:

```
$ heat --profile SECRET_KEY stack-list
# it will print <Trace ID>
```

Get pretty HTML with traces:

```
$ osprofiler trace show --html <Trace ID>
```

Note that osprofiler should be run with the admin user name & tenant.

Create a stack

Now that you have a working heat environment you can go to *Creating your first stack*.

4.1.2 Blueprints and Specs

You have to create a Story in StoryBoard heat storyboard. And create tasks that fit with the plan to implement this spec (A task to link to a patch in gerrit).

> **Note**
>
> heat-spacs is no longer active, theres no requirement for any feature to summit spac on it.

Spec from existing stories

If theres an already existing story that describes feature suitable to the story. There is no need to create a new story. The comments and history of the existing story are important for its review.

4.1.3 Heat architecture

Heat is a service to orchestrate multiple composite cloud applications using the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.

Detailed description

What is the purpose of the project and vision for it?

Heat provides an AWS CloudFormation implementation for OpenStack that orchestrates an AWS CloudFormation template describing a cloud application by executing appropriate OpenStack API calls to generate running cloud applications.

Describe the relevance of the project to other OpenStack projects and the OpenStack mission to provide a ubiquitous cloud computing platform:

The software integrates other core components of OpenStack into a one-file template system. The templates allow creation of most OpenStack resource types (such as instances, floating IPs, volumes, security groups and users), as well as some more advanced functionality such as instance high availability, instance autoscaling, and nested stacks. By providing very tight integration with other OpenStack core projects, all OpenStack core projects could receive a larger user base.

Currently no other CloudFormation implementation exists for OpenStack. The developers believe cloud developers have a strong desire to move workloads from AWS to OpenStack deployments. Given the missing gap of a well-implemented and integrated CloudFormation API in OpenStack, we provide a high quality implementation of this gap improving the ubiquity of OpenStack.

Heat services

The developers are focused on creating an OpenStack style project using OpenStack design tenets, implemented in Python. We have started with full integration with keystone. We have a number of components.

As the developers have only started development in March 2012, the architecture is evolving rapidly.

heat

The heat tool is a CLI which communicates with the heat-api to execute AWS CloudFormation APIs. End developers could also use the heat REST API directly.

heat-api

The heat-api component provides an OpenStack-native REST API that processes API requests by sending them to the heat-engine over RPC.

heat-api-cfn

The heat-api-cfn component provides an AWS Query API that is compatible with AWS CloudFormation and processes API requests by sending them to the heat-engine over RPC.

heat-engine

The heat-engines main responsibility is to orchestrate the launching of templates and provide events back to the API consumer.

The templates integrate well with Puppet and Chef.

4.1.4 Heat Resource Plug-in Development Guide

Heat allows service providers to extend the capabilities of the orchestration service by writing their own resource plug-ins. These plug-ins are written in Python and included in a directory configured by the service provider. This guide describes a resource plug-in structure and life cycle in order to assist developers in writing their own resource plug-ins.

Resource Plug-in Life Cycle

A resource plug-in is relatively simple in that it needs to extend a base `Resource` class and implement some relevant life cycle handler methods. The basic life cycle methods of a resource are:

create
: The plug-in should create a new physical resource.

update
: The plug-in should update an existing resource with new configuration or tell the engine that the resource must be destroyed and re-created. This method is optional; the default behavior is to create a replacement resource and then delete the old resource.

suspend
: The plug-in should suspend operation of the physical resource; this is an optional operation.

resume
: The plug-in should resume operation of the physical resource; this is an optional operation.

delete
: The plug-in should delete the physical resource.

The base class `Resource` implements each of these life cycle methods and defines one or more handler methods that plug-ins can implement in order to manifest and manage the actual physical resource abstracted by the plug-in. These handler methods will be described in detail in the following sections.

Heat Resource Base Class

Plug-ins must extend the class `heat.engine.resource.Resource`.

This class is responsible for managing the overall life cycle of the plug-in. It defines methods corresponding to the life cycle as well as the basic hooks for plug-ins to handle the work of communicating with specific down-stream services. For example, when the engine determines it is time to create a resource, it calls the `create` method of the applicable plug-in. This method is implemented in the `Resource` base class and handles most of the bookkeeping and interaction with the engine. This method then calls a `handle_create` method defined in the plug-in class (if implemented) which is responsible for using specific service calls or other methods needed to instantiate the desired physical resource (server, network, volume, etc).

Resource Status and Action

The base class handles reporting state of the resource back to the engine. A resources state is the combination of the life cycle action and the status of that action. For example, if a resource is created successfully, the state of that resource will be `CREATE_COMPLETE`. Alternatively, if the plug-in encounters an error when attempting to create the physical resource, the state would be `CREATE_FAILED`. The base class handles the reporting and persisting of resource state, so a plug-ins handler methods only need to return data or raise exceptions as appropriate.

Resource Support Status

New resource should be marked from which OpenStack release it will be available with *support_status* option. For more details, see *Heat Support Status usage Guide*.

Resource description

An important part of future resources is a concisely written description. It should be in class docstring and contain information about the resource and how it could be useful to the end-user. The docstring description is used in documentation generation and should be always defined, if resource is designed for public use. Docstring should follows PEP 257.

```
class CustomResource(resource.Resource):
    """This custom resource has description.

    Now end-users could understand the meaning of the resource existing
    and will use it correctly without any additional questions.
    """
```

Properties and Attributes

A resources *properties* define the settings the template author can manipulate when including that resource in a template. Some examples would be:

- Which flavor and image to use for a Nova server
- The port to listen to on Neutron LBaaS nodes
- The size of a Cinder volume

> **Note**
>
> Properties should normally be accessed through self.properties. This resolves intrinsic functions, provides default values when required and performs property translation for backward compatible schema changes. The self.properties.data dict provides access to the raw data supplied by the user in the template without any of those transformations.

Attributes describe runtime state data of the physical resource that the plug-in can expose to other resources in a Stack. Generally, these arent available until the physical resource has been created and is in a usable state. Some examples would be:

- The host id of a Nova server
- The status of a Neutron network
- The creation time of a Cinder volume

Defining Resource Properties

Each property that a resource supports must be defined in a schema that informs the engine and validation logic what the properties are, what type each is, and validation constraints. The schema is a dictionary whose keys define property names and whose values describe the constraints on that property. This dictionary must be assigned to the `properties_schema` attribute of the plug-in.

```
from heat.common.i18n import _
from heat.engine import constraints
from heat.engine import properties

    nested_schema = {
        "foo": properties.Schema(
            properties.Schema.STRING,
            _('description of foo field'),
            constraints=[
                constraints.AllowedPattern('(Ba[rc]?)+'),
                constraints.Length(max=10,
                                   description="don't go crazy")
            ]
        )
    }
    properties_schema = {
        "property_name": properties.Schema(
            properties.Schema.MAP,
            _('Internationalized description of property'),
            required=True,
            default={"Foo": "Bar"},
            schema=nested_schema
        )
    }
```

As shown above, some properties may themselves be complex and reference nested schema definitions. Following are the parameters to the `Schema` constructor; all but the first have defaults.

data_type:

Defines the type of the propertys value. The valid types are the members of the list `properties.Schema.TYPES`, currently `INTEGER`, `STRING`, `NUMBER`, `BOOLEAN`, `MAP`, `LIST` and `ANY`; please use those symbolic names rather than the literals to which they are equated. For `LIST` and `MAP` type properties, the `schema` referenced constrains the format of complex items in the list or map.

***description*:**
A description of the property and its function; also used in documentation generation. Default is `None` but you should always provide a description.

***default*:**
The default value to assign to this property if none was supplied in the template. Default is `None`.

***schema*:**
This propertys value is complex and its members must conform to this referenced schema in order to be valid. The referenced schema dictionary has the same format as the `properties_schema`. Default is `None`.

***required*:**
`True` if the property must have a value for the template to be valid; `False` otherwise. The default is `False`

***constraints*:**
A list of constraints that apply to the propertys value. See *Property Constraints*.

***update_allowed*:**
`True` if an existing resource can be updated, `False` means update is accomplished by delete and re-create. Default is `False`.

***immutable*:**
`True` means updates are not supported, resource update will fail on every change of this property. `False` otherwise. Default is `False`.

***support_status*:**
Defines current status of the property. Read *Heat Support Status usage Guide* for details.

Accessing property values of the plug-in at runtime is then a simple call to:

```
self.properties['PropertyName']
```

Based on the property type, properties without a set value will return the default empty value for that type:

Type	Empty Value
String	
Number	0
Integer	0
List	[]
Map	{}
Boolean	False

Property Constraints

Following are the available kinds of constraints. The description is optional and, if given, states the constraint in plain language for the end user.

***AllowedPattern(regex, description)*:**
Constrains the value to match the given regular expression; applicable to STRING.

***AllowedValues(allowed, description)*:**
Lists the allowed values. `allowed` must be a `collections.abc.Sequence` or `string`. Applicable to all types of value except MAP.

***Length(min, max, description)*:**
Constrains the length of the value. Applicable to STRING, LIST, MAP. Both `min` and `max` default to `None`.

***Range(min, max, description)*:**
Constrains a numerical value. Applicable to INTEGER and NUMBER. Both `min` and `max` default to `None`.

***Modulo(step, offset, description)*:**
Starting with the specified `offset`, every multiple of `step` is a valid value. Applicable to INTEGER and NUMBER.

Available from template version 2017-02-24.

***CustomConstraint(name, description, environment)*:**
This constructor brings in a named constraint class from an environment. If the given environment is `None` (its default) then the environment used is the global one.

Defining Resource Attributes

Attributes communicate runtime state of the physical resource. Note that some plug-ins do not define any attributes and doing so is optional. If the plug-in needs to expose attributes, it will define an `attributes_schema` similar to the properties schema described above. Each item in the schema dictionary consists of an attribute name and an attribute Schema object.

```
attributes_schema = {
    "foo": attributes.Schema(
        _("The foo attribute"),
        type=attribute.Schema.STRING
    ),
    "bar": attributes.Schema(
        _("The bar attribute"),
        type=attribute.Schema.STRING
    ),
    "baz": attributes.Schema(
        _("The baz attribute"),
        type=attribute.Schema.STRING
    )
}
```

Following are the parameters to the Schema.

description
A description of the attribute; also used in documentation generation. Default is `None` but you

should always provide a description.

type
Defines the type of attribute value. The valid types are the members of the list `attributes.Schema.TYPES`, currently `STRING`, `NUMBER`, `BOOLEAN`, `MAP`, and `LIST`; please use those symbolic names rather than the literals to which they are equated.

support_status
Defines current status of the attribute. Read *Heat Support Status usage Guide* for details.

If attributes are defined, their values must also be resolved by the plug-in. The simplest way to do this is to override the `_resolve_attribute` method from the `Resource` class:

```
def _resolve_attribute(self, name):
    # _example_get_physical_resource is just an example and is not
    # defined in the Resource class
    phys_resource = self._example_get_physical_resource()
    if phys_resource:
        if not hasattr(phys_resource, name):
                # this is usually not needed, but this is a simple
                # example
                raise exception.InvalidTemplateAttribute(name)
        return getattr(phys_resource, name)
    return None
```

If the plug-in needs to be more sophisticated in its attribute resolution, the plug-in may instead choose to override `FnGetAtt`. However, if this method is chosen, validation and accessibility of the attribute would be the plug-ins responsibility.

Also, each resource has `show` attribute by default. The attribute uses default implementation from `heat.engine.resource.Resource` class, but if resource has different way of resolving `show` attribute, the `_show_resource` method from the `Resource` class will need to be overridden:

```
def _show_resource(self):
    """Default implementation; should be overridden by resources.

    :returns: the map of resource information or None
     """
    if self.entity:
        try:
            obj = getattr(self.client(), self.entity)
            resource = obj.get(self.resource_id)
            if isinstance(resource, dict):
                return resource
            else:
                return resource.to_dict()
        except AttributeError as ex:
            LOG.warning("Resolving 'show' attribute has failed : %s",
                        ex)
            return None
```

Property and Attribute Example

Assume the following simple property and attribute definition:

```
properties_schema = {
    'foo': properties.Schema(
        properties.Schema.STRING,
        _('foo prop description'),
        default='foo',
        required=True
    ),
    'bar': properties.Schema(
        properties.Schema.INTEGER,
        _('bar prop description'),
        required=True,
        constraints=[
            constraints.Range(5, 10)
        ]
    )
}

attributes_schema = {
    'Attr_1': attributes.Schema(
        _('The first attribute'),
        support_status=support.Status('5.0.0'),
        type=attributes.Schema.STRING
    ),
    'Attr_2': attributes.Schema(
        _('The second attribute'),
        type=attributes.Schema.MAP
    )
}
```

Also assume the plug-in defining the above has been registered under the template reference name Resource::Foo (see *Registering Resource Plug-ins*). A template author could then use this plug-in in a stack by simply making following declarations in a template:

```
# ... other sections omitted for brevity ...

resources:
  resource-1:
    type: Resource::Foo
    properties:
      foo: Value of the foo property
      bar: 7

outputs:
  foo-attrib-1:
    value: { get_attr: [resource-1, Attr_1] }
    description: The first attribute of the foo resource
  foo-attrib-2:
```

(continues on next page)

(continued from previous page)

```
    value: { get_attr: [resource-1, Attr_2] }
    description: The second attribute of the foo resource
```

Life Cycle Handler Methods

To do the work of managing the physical resource the plug-in supports, the following life cycle handler methods should be implemented. Note that the plug-in need not implement *all* of these methods; optional handlers will be documented as such.

Generally, the handler methods follow a basic pattern. The basic handler method for any life cycle step follows the format `handle_<life cycle step>`. So for the create step, the handler method would be `handle_create`. Once a handler is called, an optional `check_<life cycle step>_complete` may also be implemented so that the plug-in may return immediately from the basic handler and then take advantage of cooperative multi-threading built in to the base class and periodically poll a down-stream service for completion; the check method is polled until it returns `True`. Again, for the create step, this method would be `check_create_complete`.

Create

handle_create(*self*)

Create a new physical resource. This function should make the required calls to create the physical resource and return as soon as there is enough information to identify the resource. The function should return this identifying information and implement `check_create_complete` which will take this information in as a parameter and then periodically be polled. This allows for cooperative multi-threading between multiple resources that have had their dependencies satisfied.

Note once the native identifier of the physical resource is known, this function should call `self.resource_id_set` passing the native identifier of the physical resource. This will persist the identifier and make it available to the plug-in by accessing `self.resource_id`.

Returns
A representation of the created physical resource

Raise
any `Exception` if the create failed

check_create_complete(*self*, *token*)

If defined, will be called with the return value of `handle_create`

Parameters
token the return value of `handle_create`; used to poll the physical resources status.

Returns
`True` if the physical resource is active and ready for use; `False` otherwise.

Raise
any `Exception` if the create failed.

Update (Optional)

Note that there is a default implementation of `handle_update` in `heat.engine.resource.Resource` that simply raises an exception indicating that updates require the engine to delete and re-create the resource (this is the default behavior) so implementing this is optional.

handle_update(*self*, *json_snippet*, *tmpl_diff*, *prop_diff*)

Update the physical resources using updated information.

Parameters

- **json_snippet** (*collections.abc.Mapping*) the resource definition from the updated template
- **tmpl_diff** (*collections.abc.Mapping*) values in the updated definition that have changed with respect to the original template definition.
- **prop_diff** (*collections.abc.Mapping*) property values that are different between the original definition and the updated definition; keys are property names and values are the new values. Deleted or properties that were originally present but now absent have values of `None`

Note Before calling `handle_update` we check whether need to replace the resource, especially for resource in `*_FAILED` state, there is a default implementation of `needs_replace_failed` in `heat.engine.resource.Resource` that simply returns `True` indicating that updates require replacement. And we override the implementation for `OS::Nova::Server`, `OS::Cinder::Volume` and all of neutron resources. The base principle is that to check whether the resource exists underlying and whether the real status is available. So override the method `needs_replace_failed` for your resource plug-ins if needed.

check_update_complete(*self*, *token*)

If defined, will be called with the return value of `handle_update`

Parameters

token the return value of `handle_update`; used to poll the physical resources status.

Returns

`True` if the update has finished; `False` otherwise.

Raise

any `Exception` if the update failed.

Suspend (Optional)

These handler functions are optional and only need to be implemented if the physical resource supports suspending

handle_suspend(*self*)

If the physical resource supports it, this function should call the native API and suspend the resources operation. This function should return information sufficient for `check_suspend_complete` to poll the native API to verify the operations status.

Returns

a token containing enough information for `check_suspend_complete` to verify operation status.

Raise
any `Exception` if the suspend operation fails.

check_suspend_complete(*self*, *token*)

Verify the suspend operation completed successfully.

Parameters
token the return value of `handle_suspend`

Returns
`True` if the suspend operation completed and the physical resource is now suspended; `False` otherwise.

Raise
any `Exception` if the suspend operation failed.

Resume (Optional)

These handler functions are optional and only need to be implemented if the physical resource supports resuming from a suspended state

handle_resume(*self*)

If the physical resource supports it, this function should call the native API and resume a suspended resources operation. This function should return information sufficient for `check_resume_complete` to poll the native API to verify the operations status.

Returns
a token containing enough information for `check_resume_complete` to verify operation status.

Raise
any `Exception` if the resume operation fails.

check_resume_complete(*self*, *token*)

Verify the resume operation completed successfully.

Parameters
token the return value of `handle_resume`

Returns
`True` if the resume operation completed and the physical resource is now active; `False` otherwise.

Raise
any Exception if the resume operation failed.

Delete

handle_delete(*self*)

Delete the physical resource.

Returns
a token containing sufficient data to verify the operations status

Raise
any `Exception` if the delete operation failed

Note

As of the Liberty release, implementing handle_delete is optional. The parent resource class can handle the most common pattern for deleting resources:

```
def handle_delete(self):
    if self.resource_id is not None:
        try:
            self.client().<entity>.delete(self.resource_id)
        except Exception as ex:
            self.client_plugin().ignore_not_found(ex)
            return None
        return self.resource_id
```

For this to work for a particular resource, the *entity* and *default_client_name* attributes must be overridden in the resource implementation. For example, *entity* of Aodh Alarm should equals to alarm and *default_client_name* to aodh.

handle_delete_snapshot(*self*, *snapshot*)

Delete resource snapshot.

Parameters

snapshot dictionary describing current snapshot.

Returns

a token containing sufficient data to verify the operations status

Raise

any `Exception` if the delete operation failed

handle_snapshot_delete(*self*, *state*)

Called instead of `handle_delete` when the deletion policy is SNAPSHOT. Create backup of resource and then delete resource.

Parameters

state the (action, status) tuple of the resource to make sure that backup may be created for the current resource

Returns

a token containing sufficient data to verify the operations status

Raise

any `Exception` if the delete operation failed

check_delete_complete(*self*, *token*)

Verify the delete operation completed successfully.

Parameters

token the return value of `handle_delete` or `handle_snapshot_delete` (for deletion policy - Snapshot) used to verify the status of the operation

Returns

`True` if the delete operation completed and the physical resource is deleted; `False` otherwise.

Raise
any `Exception` if the delete operation failed.

`check_delete_snapshot_complete`(*self*, *token*)

Verify the delete snapshot operation completed successfully.

Parameters
token the return value of `handle_delete_snapshot` used to verify the status of the operation

Returns
`True` if the delete operation completed and the snapshot is deleted; `False` otherwise.

Raise
any `Exception` if the delete operation failed.

Resource Dependencies

Ideally, your resource should not have any hidden dependencies, i.e. Heat should be able to infer any inbound or outbound dependencies of your resource instances from resource properties and the other resources/resource attributes they reference. This is handled by `heat.engine.resource.Resource.add_dependencies()`.

If this is not possible, please do not simply override *add_dependencies()* in your resource plugin! This has previously caused problems for multiple operations, usually due to uncaught exceptions, If you feel you need to override *add_dependencies()*, please reach out to Heat developers on the *#heat* IRC channel on OFTC or on the openstack-discuss mailing list to discuss the possibility of a better solution.

Registering Resource Plug-ins

To make your plug-in available for use in stack templates, the plug-in must register a reference name with the engine. This is done by defining a `resource_mapping` function in your plug-in module that returns a map of template resource type names and their corresponding implementation classes:

```
def resource_mapping():
    return { 'My::Custom::Plugin': MyResourceClass }
```

This would allow a template author to define a resource as:

```
resources:
  my_resource:
    type: My::Custom::Plugin
    properties:
    # ... your plug-in's properties ...
```

Note that you can define multiple plug-ins per module by simply returning a map containing a unique template type name for each. You may also use this to register a single resource plug-in under multiple template type names (which you would only want to do when constrained by backwards compatibility).

Configuring the Engine

In order to use your plug-in, Heat must be configured to read your resources from a particular directory. The `plugin_dirs` configuration option lists the directories on the local file system where the engine will search for plug-ins. Simply place the file containing your resource in one of these directories and the engine will make them available next time the service starts.

See *Configuring Heat* for more information on configuring the orchestration service.

Testing

Tests can live inside the plug-in under the `tests` namespace/directory. The Heat plug-in loader will implicitly not load anything under that directory. This is useful when your plug-in tests have dependencies you dont want installed in production.

Putting It All Together

You can find the plugin classes in `heat/engine/resources`. An exceptionally simple one to start with is `random_string.py`; it is unusual in that it does not manipulate anything in the cloud!

Resource Contributions

The Heat team is interested in adding new resources that give Heat access to additional OpenStack or StackForge projects. The following checklist defines the requirements for a candidate resource to be considered for inclusion:

- Must wrap an OpenStack or StackForge project, or a third party project that is relevant to OpenStack users.
- Must have its dependencies listed in OpenStacks `global-requirements.txt` file, or else it should be able to conditionally disable itself when there are missing dependencies, without crashing or otherwise affecting the normal operation of the heat-engine service.
- The resources support status flag must be set to `UNSUPPORTED`, to indicate that the Heat team is not responsible for supporting this resource.
- The code must be of comparable quality to official resources. The Heat team can help with this during the review phase.

If you have a resource that is a good fit, you are welcome to contact the Heat team. If for any reason your resource does not meet the above requirements, but you still think it can be useful to other users, you are encouraged to host it on your own repository and share it as a regular Python installable package. You can find example resource plug-ins that have all the required packaging files in the `contrib` directory of the official Heat git repository.

4.1.5 Heat Stack Lifecycle Scheduler Hints

This is a mechanism whereby when heat processes a stack with Server or Volume resources, the stack id, root stack id, stack resource uuid, stack resource name and the path in the stack can be passed by heat to nova and cinder as scheduler hints.

Enabling the scheduler hints

By default, passing the lifecycle scheduler hints is disabled. To enable it, set stack_scheduler_hints to True in heat.conf.

The hints

When heat processes a stack, and the feature is enabled, the stack id, root stack id, stack resource uuid, stack resource name, and the path in the stack (as a list of comma delimited strings of stackresourcename and stackname) will be passed by heat to nova and cinder as scheduler hints.

Purpose

A heat provider may have a need for custom code to examine stack requests prior to performing the operations to create or update a stack. After the custom code completes, the provider may want to provide hints to the nova or cinder schedulers with stack related identifiers, for processing by any custom scheduler plug-ins configured for nova or cinder.

4.1.6 Guru Meditation Reports

Heat contains a mechanism whereby developers and system administrators can generate a report about the state of a running Heat executable. This report is called a *Guru Meditation Report* (*GMR* for short).

Generating a GMR

A *GMR* can be generated by sending the *USR2* signal to any Heat process with support (see below). The *GMR* will then be outputted standard error for that particular process.

For example, suppose that `heat-api` has process id `10172`, and was run with `2>/var/log/heat/heat-api-err.log`. Then, `kill -USR2 10172` will trigger the Guru Meditation report to be printed to `/var/log/heat/heat-api-err.log`.

Structure of a GMR

The *GMR* is designed to be extensible; any particular executable may add its own sections. However, the base *GMR* consists of several sections:

Package
: Shows information about the package to which this process belongs, including version information

Threads
: Shows stack traces and thread ids for each of the threads within this process

Green Threads
: Shows stack traces for each of the green threads within this process (green threads dont have thread ids)

Configuration
: Lists all the configuration options currently accessible via the CONF object for the current process

Adding support for GMRs to new executable

Adding support for a *GMR* to a given executable is fairly easy.

First import the module, as well as the Heat version module:

```
from oslo_reports import guru_meditation_report as gmr
from heat import version
```

Then, register any additional sections (optional):

```
TextGuruMeditation.register_section('Some Special Section',
                                    some_section_generator)
```

Finally (under main), before running the main loop of the executable (usually `server.start()` or something similar), register the *GMR* hook:

```
TextGuruMeditation.setup_autorun(version)
```

Extending the GMR

As mentioned above, additional sections can be added to the GMR for a particular executable. For more information, see the documentation about oslo.reports.

4.1.7 Heat Support Status usage Guide

Heat allows to use for each resource, property, attribute special option named *support_status*, which describes current state of object: current status, since what time this status is actual, any additional information about objects state. This guide describes a detailed state life cycle of resources, properties and attributes.

Support Status option and its parameters

Support status of object may be specified by using class `SupportStatus`, which has follow options:

***status*:**

Current status of object. Allowed values:

- SUPPORTED. Default value of status parameter. All objects with this status are available and can be used.
- DEPRECATED. Object with this status is available, but using it in code or templates is undesirable. As usual, can be reference in message to new object, which can be used instead of deprecated resource.
- HIDDEN. The last step in the deprecation process. Old stacks containing resources in this status will continue functioning. Certain functionality is disabled for resources in this status (resource-type-list, resource-type-show, and resource-type-template). Resources in HIDDEN status are not included in the documentation. A known limitation is that new stacks can be created with HIDDEN resources. See below for more details about the removal and deprecation process.
- UNSUPPORTED. Resources with UNSUPPORTED status are not supported by Heat team, i.e. user can use it, but it may be broken.

***substitute_class*:**

Assign substitute class for object. If replacing the object with new object which inherited (or extended) from the substitute class will transfer the object to new class type gracefully (without calling update replace).

***version*:**

Release name, since which current status is active. Parameter is optional, but should be defined or changed any time SupportStatus is specified or status changed. It used for better understanding from which release object in current status. .. note:

```
Since Liberty release mark looks like 5.0.0 instead of 2015.2.
```

message:
Any additional information about objects state, e.g. `'Use property new_property instead.'`.

previous_status:
Option, which allows to display objects previous status, if any. This is helpful for displaying full life cycle of object. Type of *previous_status* is SupportStatus.

Life cycle of resource, property, attribute

This section describes life cycle of such objects as resource, property and attribute. All these objects have same life cycle:

```
UNSUPPORTED -> SUPPORTED -> DEPRECATED -> HIDDEN
                                       \
                                        -> UNSUPPORTED
```

where UNSUPPORTED is optional.

Creating process of object

During creating object there is a reason to add support status. So new object should contains *support_status* parameter equals to `SupportStatus` class with defined version of object and, maybe, *substitute_class* or some message. This parameter allows user to understand, from which OpenStack release this object is available and can be used.

Deprecating process of object

When some object becomes obsolete, user should know about that, so there is need to add information about deprecation in *support_status* of object. Status of `SupportStatus` must equals to DEPRECATED. If there is no *version* parameter, need to add one with current release otherwise move current status to *previous_status* and add to *version* current release as value. If some new object replaces old object, it will be good decision to add some information about new object to *support_status* message of old object, e.g. Use property new_property instead.. If old object is directly replaceable by new object, we should add *substitute_class* to *support_status* in old object.

Removing process of object

After at least one full release cycle deprecated object should be hidden and *support_status* status should equals to HIDDEN. HIDDEN status means hiding object from documentation and from result of `resource-type-list` CLI command, if object is resource. Also, `resource-type-show` command with such resource will raise *NotSupported* exception.

The purpose of hiding, rather than removing, obsolete resources or properties is to ensure that users can continue to operate existing stacks - replacing or removing the offending resources, or deleting the entire stack. Steps should be taken to ensure that these operations can succeed, e.g. by replacing a hidden resource types implementation with one that is equivalent to `OS::Heat::None` when the underlying API no longer exists, supplying a *substitute_class* for a resource type, or adding a property translation rule.

Using Support Status during code writing

When adding new objects or adding objects instead of some old (e.g. property subnet instead of subnet_id in OS::Neutron::RouterInterface), there is some information about time of adding objects (since which release it will be available or unavailable). This section described `SupportStatus` during creating/deprecating/removing resources and properties and attributes. Note, that `SupportStatus` locates in support.py, so you need to import *support*. For specifying status, use *support* constant names, e.g. support.SUPPORTED. All constant names described in section above.

Using Support Status during creation

Option *support_status* may be used for whole resource:

```
class ResourceWithType(resource.Resource):

    support_status=support.SupportStatus(
        version='5.0.0',
        message=_('Optional message')
    )
```

To define *support_status* for property or attribute, follow next steps:

```
PROPERTY: properties.Schema(
    ...
    support_status=support.SupportStatus(
        version='5.0.0',
        message=_('Optional message')
    )
)
```

Same support_status definition for attribute schema.

Note, that in this situation status parameter of `SupportStatus` uses default value, equals to SUPPORTED.

Using Support Status during deprecation and hiding

When time of deprecation or hiding resource/property/attribute comes, follow next steps:

1. If there is some support_status in object, add *previous_status* parameter with current `SupportStatus` value and change all other parameters for current *status*, *version* and, maybe, *substitute_class* or *message*.
2. If there is no support_status option, add new one with parameters status equals to current status, *version* equals to current release note and, optionally, some message.

Using Support Status during resource deprecating looks like:

```
class ResourceWithType(resource.Resource):

    support_status=support.SupportStatus(
        status=support.DEPRECATED,
        version='5.0.0',
        substitute_class=SubstituteResourceWithType,
```

(continues on next page)

(continued from previous page)

```
        message=_('Optional message'),
        previous_status=support.SupportStatus(version='2014.2')
    )
```

Using Support Status during attribute (or property) deprecating looks like:

```
ATTRIBUTE: attributes.Schema(
    ...
    support_status=support.SupportStatus(
        status=support.DEPRECATED,
        version='5.0.0',
        message=_('Optional message like: Use attribute new_attr'),
        previous_status=support.SupportStatus(
            version='2014.2',
            message=_('Feature available since 2014.2'))
    )
)
```

Same *support_status* defining for property schema.

Note, that during hiding object status should be equal support.HIDDEN instead of support.DEPRECATED. Besides that, SupportStatus with DEPRECATED status should be moved to *previous_status*, e.g.:

```
support.SupportStatus(
    status=support.HIDDEN,
    version='6.0.0',
    message=_('Some message'),
    previous_status=support.SupportStatus(
        status=support.DEPRECATED,
        version='2015.1',
        substitute_class=SubstituteResourceWithType,
        previous_status=support.SupportStatus(version='2014.2')
    )
)
```

During hiding properties, if some hidden property has alternative, use translation mechanism for translating properties from old to new one. See below, how to use this mechanism.

Translating mechanism for hidden properties

Sometimes properties become deprecated and replaced by another. There is translation mechanism for that. Mechanism used for such cases:

1. If there are two properties in properties_schema, which have STRING, INTEGER, NUMBER or BOOLEAN type.
2. If there are two properties: one in LIST or MAP property sub-schema and another on the top schema.
3. If there are two properties in LIST property.
4. If there are non-LIST property and LIST property, which is designed to replace non-LIST property.

5. If there is STRING property, which contains name or ID of some entity, e.g. *subnet*, and should be resolved to entitys ID.

Mechanism has rules and executes them. To define rule, `TranslationRule` class called and specifies *translation_path* - list with path in properties_schema for property which will be affected; *value* - value, which will be added to property, specified by previous parameter; *value_name* - name of old property, used for case 4; *value_path* - list with path in properties_schema for property which will be used for getting value. `TranslationRule` supports next rules:

- *ADD*. This rule allows to add some value to LIST-type properties. Only LIST-type values can be added to such properties. Using for other cases is prohibited and will be returned with error.
- *REPLACE*. This rule allows to replace some property value to another. Used for all types of properties. Note, that if property has list type, then value will be replaced for all elements of list, where it needed. If element in such property must be replaced by value of another element of this property, *value_name* must be defined.
- *DELETE*. This rule allows to delete some property. If property has list type, then deleting affects value in all list elements.
- *RESOLVE* - This rule allows to resolve some property using client and the *finder* function. Finders may require an additional *entity* key.

Each resource, which has some hidden properties, which can be replaced by new, must overload *translation_rules* method, which should return a list of `TranslationRules`, for example:

```
def translation_rules(self, properties):
    rules = [
      translation.TranslationRule(
        properties,
        translation.TranslationRule.REPLACE,
        translation_path=[self.NETWORKS, self.NETWORK_ID],
        value_name=self.NETWORK_UUID),
      translation.TranslationRule(
        properties,
        translation.TranslationRule.RESOLVE,
        translation_path=[self.FLAVOR],
        client_plugin=self.client_plugin('nova'),
        finder='find_flavor_by_name_or_id')]
    return rules
```

4.1.8 Using Rally on Heat gates

Heat gate allows to use Rally for performance testing for each particular patch. This functionality can be used for checking patch on performance regressions and also for detecting any floating bugs for common scenarios.

How to run Rally for particular patch

As was mentioned above Heat allows to execute Rally scenarios as a gate job for particular patch. It can be done by posting comment with text `check experimental` for patch on review. It will run bunch of jobs, one of which has name `gate-rally-dsvm-fakevirt-heat`.

List of scenarios, which will be executed, is presented in file `heat-fakevirt.yaml`. Default version of this file is available here: https://github.com/openstack/heat/blob/master/rally-scenarios/heat-fakevirt.

yaml

Obviously performance analysis make sense, when it can be compared with some another performance data. So two different approaches can be used for it:

- Comparison of one part of code with some custom changes (see *Check performance or how to detect regression*)
- Comparison of two different code parts (see *Compare output API performance*)

Examples of using Rally

Previously two main approaches of using Rally job for Heat were highlighted. Corresponding examples will be described in this part of documentation.

However need to note, that there are a lot of other ways how to use Rally job for Heat performance. For example, this job can be launched periodically (twice in week) for random patches and these results will be compared between each other. It allows to see, that Heat has not any performance regressions.

Check performance or how to detect regression

The easiest way of using Rally is to execute already existing scenarios. One of the examples is presented in patch https://review.opendev.org/#/c/279450/ . In this patch was executed scenario already existing in Rally `HeatStacks.create_and_delete_stack`. During executing this scenario Rally creates and then, when stack is created, delete Heat stack. All existing scenarios can be found here: https://github.com/openstack/rally-openstack/blob/master/rally_openstack/scenarios/heat/stacks.py

Mentioned scenario uses Heat template as a parameter for task. The template path should be mentioned for argument `template_path`. It can be one of Heat templates presented in Rally repository (https://github.com/openstack/rally-openstack/tree/master/samples/tasks/scenarios/heat/templates) or new one, like it was done for mentioned patch. New added template should be placed in `rally-scenarios/extra/` directory.

Also its possible to specify other fields for each Rally task, like `sla` or `context`. More information about other configuration setting is available by link https://rally.readthedocs.io/en/latest/plugins/#rally-plugins Mentioned patch was proposed for confirmation caching mechanism of Heat template validation process (see https://specs.openstack.org/openstack/heat-specs/specs/liberty/constraint-validation-cache.html). So it contains some changes in OS::Heat::TestResource resource, which allows to demonstrate mentioned caching feature improvements.

Initially test was run against current devstack installation, where caching is disabled (e.g. Patch Set 7). The follow results were gotten:

Action	Min (sec)	Max (sec)	Avg (sec)	Success	Count
heat.create_stack	38.223	48.085	42.971	100.0%	10
heat.delete_stack	11.755	18.155	14.085	100.0%	10
total	50.188	65.361	57.057	100.0%	10

In the next patch set (Patch Set 8) was updated by adding Depends-On reference to commit message. It let to execute the same test with patch for devstack, which turns on caching (https://review.opendev.org/#/c/279400/). The results for this case were:

Action	Min (sec)	Max (sec)	Avg (sec)	Success	Count
heat.create_stack	11.863	16.074	14.174	100.0%	10
heat.delete_stack	9.144	11.663	10.595	100.0%	10
total	21.557	27.18	24.77	100.0%	10

Comparison average values for create_stack action in the first and the second executions shows, that with enabled caching create_stack works faster in 3 times. It is a tangible improvement for create_stack operation. Need to note, that in described test delay for each constraint validation request takes 0.3 sec. as specified in `constraint_prop_secs` property of TestResource. It may be more, than real time delay, but it allows to confirm, that caching works correct.

Also this approach may be used for detecting regressions. In this case workflow may be presented as follow list of steps:

- add to task list (`heat-fakevirt.yaml`) existing or new tasks.
- wait a result of this execution.
- upload patchset with changes (new feature) and launch the same test again.
- compare performance results.

Compare output API performance

Another example of using Rally job is writing custom Rally scenarios in Heat repository. There is an example of this is presented on review: https://review.opendev.org/#/c/270225/

Its similar on the first example, but requires more Rally specific coding. New tasks in `heat-fakevirt.yaml` use undefined in Rally repository scenarios:

- CustomHeatBenchmark.create_stack_and_show_output_new
- CustomHeatBenchmark.create_stack_and_show_output_old
- CustomHeatBenchmark.create_stack_and_list_output_new
- CustomHeatBenchmark.create_stack_and_list_output_old

All these scenarios are defined in the same patch and placed in `rally-scenarios/plugins/` directory.

The aim of these scenarios and tasks is to demonstrate differences between new and old API calls. Heat client has a two commands for operating stack outputs: `heat output-list` and `heat output-show <output-id>`. Previously there are no special API calls for getting this information from server and this data was obtained from whole Heat Stack object. This was changed after implementation new API for outputs: https://specs.openstack.org/openstack/heat-specs/specs/mitaka/api-calls-for-output.html

As described in the mentioned specification outputs can be obtained via special requests to Heat API. According to this changes code in Heat client was updated to use new API, if its available.

The initial problem for this change was performance issue, which can be formulated as: execution command `heat output-show <output-id>` with old approach required resolving all outputs in Heat Stack, before getting only one output specified by user.

The same issue was and with `heat output-list`, which required to resolve all outputs only for providing list of output keys without resolved values.

Two scenarios with suffix `*_new` use new output API. These scenarios are not presented in Rally yet, because its new API. Another two scenarios with suffix `*_old` are based on the old approach of getting

outputs. This code was partially replaced by new API, so its not possible to use it on fresh devstack. As result this custom code was written as two custom scenarios.

All these scenarios were added to task list and executed in the same time. Results of execution are shown below:

create_stack_and_show_output_old

Action	Min (sec)	Max (sec)	Avg (sec)	Success	Count
heat.create_stack	13.559	14.298	13.899	100.0%	5
heat.show_output_old	5.214	5.297	5.252	100.0%	5
heat.delete_stack	5.445	6.962	6.008	100.0%	5
total	24.243	26.146	25.159	100.0%	5

create_stack_and_show_output_new

Action	Min (sec)	Max (sec)	Avg (sec)	Success	Count
heat.create_stack	13.719	14.286	13.935	100.0%	5
heat.show_output_new	0.699	0.835	0.762	100.0%	5
heat.delete_stack	5.398	6.457	5.636	100.0%	5
total	19.873	21.21	20.334	100.0%	5

Average value for execution `output-show` for old approach obviously more, then for new API. It happens, because new API resolve only one specified output.

Same results are for `output-list`:

create_stack_and_list_output_old

Action	Min (sec)	Max (sec)	Avg (sec)	Success	Count
heat.create_stack	13.861	14.573	14.141	100.0%	5
heat.list_output_old	5.247	5.339	5.281	100.0%	5
heat.delete_stack	6.727	6.845	6.776	100.0%	5
total	25.886	26.696	26.199	100.0%	5

create_stack_and_list_output_new

Action	Min (sec)	Max (sec)	Avg (sec)	Success	Count
heat.create_stack	13.902	21.117	16.729	100.0%	5
heat.list_output_new	0.147	0.363	0.213	100.0%	5
heat.delete_stack	6.616	8.202	7.022	100.0%	5
total	20.838	27.908	23.964	100.0%	5

Its also expected, because for getting list of output names is not necessary resolved values, how it is done in new API.

All mentioned results clearly show performance changes and allow to confirm, that new approach works correctly.

4.2 Source Code Index

4.2.1 heat

heat package

Subpackages

heat.api package

Subpackages

heat.api.aws package

Submodules

heat.api.aws.ec2token module

class heat.api.aws.ec2token.**EC2Token**(*app*, *conf*)

Bases: *Middleware*

Authenticate an EC2 request with keystone and convert to token.

heat.api.aws.ec2token.**EC2Token_filter_factory**(*global_conf*, ***local_conf*)

Factory method for paste.deploy.

heat.api.aws.ec2token.**list_opts**()

heat.api.aws.exception module

Heat API exception subclasses - maps API response errors to AWS Errors.

exception heat.api.aws.exception.**AlreadyExistsError**(*detail=None*)

Bases: *HeatAPIException*

Resource with the name requested already exists.

code = 400

explanation = 'Resource with the name requested already exists'

title = 'AlreadyExists'

exception heat.api.aws.exception.**HeatAPIException**(*detail=None*)

Bases: HTTPError

webob HTTPError subclass that creates a serialized body.

Subclass webob HTTPError so we can correctly serialize the wsgi response into the http response body, using the format specified by the request. Note this should not be used directly, instead use the subclasses defined below which map to AWS API errors.

code = 400

err_type = 'Sender'

explanation = 'Generic HeatAPIException, please use specific subclasses!'

get_unserialized_body()

Return a dict suitable for serialization in the wsgi controller.

This wraps the exception details in a format which maps to the expected format for the AWS API.

title = 'HeatAPIException'

exception heat.api.aws.exception.**HeatAPINotImplementedError**(*detail=None*)

Bases: *HeatAPIException*

API action is not yet implemented.

code = 500

err_type = 'Server'

explanation = 'The requested action is not yet implemented'

title = 'APINotImplemented'

exception heat.api.aws.exception.**HeatAccessDeniedError**(*detail=None*)

Bases: *HeatAPIException*

Authentication fails due to user IAM group memberships.

This is the response given when authentication fails due to user IAM group memberships meaning we deny access.

code = 403

explanation = 'User is not authorized to perform action'

title = 'AccessDenied'

exception heat.api.aws.exception.**HeatActionInProgressError**(*detail=None*)

Bases: *HeatAPIException*

Cannot perform action on stack in its current state.

code = 400

explanation = 'Cannot perform action on stack while other actions are in progress'

title = 'InvalidAction'

exception heat.api.aws.exception.**HeatIncompleteSignatureError**(*detail=None*)

Bases: *HeatAPIException*

The request signature does not conform to AWS standards.

code = 400

explanation = 'The request signature does not conform to AWS standards'

title = 'IncompleteSignature'

exception heat.api.aws.exception.**HeatInternalFailureError**(*detail=None*)

Bases: *HeatAPIException*

The request processing has failed due to some unknown error.

code = 500

err_type = 'Server'

explanation = 'The request processing has failed due to an internal error'

title = 'InternalFailure'

exception heat.api.aws.exception.**HeatInvalidActionError**(*detail=None*)

Bases: *HeatAPIException*

The action or operation requested is invalid.

code = 400

explanation = 'The action or operation requested is invalid'

title = 'InvalidAction'

exception heat.api.aws.exception.**HeatInvalidClientTokenIdError**(*detail=None*)

Bases: *HeatAPIException*

The X.509 certificate or AWS Access Key ID provided does not exist.

code = 403

explanation = 'The certificate or AWS Key ID provided does not exist'

title = 'InvalidClientTokenId'

exception heat.api.aws.exception.**HeatInvalidParameterCombinationError**(*detail=None*)

Bases: *HeatAPIException*

Parameters that must not be used together were used together.

code = 400

explanation = 'Incompatible parameters were used together'

title = 'InvalidParameterCombination'

exception heat.api.aws.exception.**HeatInvalidParameterValueError**(*detail=None*)

Bases: *HeatAPIException*

A bad or out-of-range value was supplied for the input parameter.

code = 400

explanation = 'A bad or out-of-range value was supplied'

title = 'InvalidParameterValue'

exception heat.api.aws.exception.**HeatInvalidQueryParameterError**(*detail=None*)

Bases: *HeatAPIException*

AWS query string is malformed, does not adhere to AWS standards.

code = 400

explanation = 'AWS query string is malformed, does not adhere to AWS spec'

title = 'InvalidQueryParameter'

exception heat.api.aws.exception.**HeatMalformedQueryStringError**(*detail=None*)

Bases: *HeatAPIException*

The query string is malformed.

code = 404

explanation = 'The query string is malformed'

title = 'MalformedQueryString'

exception heat.api.aws.exception.**HeatMissingActionError**(*detail=None*)

Bases: *HeatAPIException*

The request is missing an action or operation parameter.

code = 400

explanation = 'The request is missing an action or operation parameter'

title = 'MissingAction'

exception heat.api.aws.exception.**HeatMissingAuthenticationTokenError**(*detail=None*)

Bases: *HeatAPIException*

Does not contain a valid AWS Access Key or certificate.

Request must contain either a valid (registered) AWS Access Key ID or X.509 certificate.

code = 403

explanation = 'Does not contain a valid AWS Access Key or certificate'

title = 'MissingAuthenticationToken'

exception heat.api.aws.exception.**HeatMissingParameterError**(*detail=None*)

Bases: *HeatAPIException*

A mandatory input parameter is missing.

An input parameter that is mandatory for processing the request is missing.

code = 400

explanation = 'A mandatory input parameter is missing'

title = 'MissingParameter'

exception heat.api.aws.exception.**HeatOptInRequiredError**(*detail=None*)

Bases: *HeatAPIException*

The AWS Access Key ID needs a subscription for the service.

code = 403

explanation = 'The AWS Access Key ID needs a subscription for the service'

title = 'OptInRequired'

exception heat.api.aws.exception.**HeatRequestExpiredError**(*detail=None*)

Bases: *HeatAPIException*

Request expired or more than 15 minutes in the future.

Request is past expires date or the request date (either with 15 minute padding), or the request date occurs more than 15 minutes in the future.

code = 400

explanation = 'Request expired or more than 15mins in the future'

title = 'RequestExpired'

exception heat.api.aws.exception.**HeatRequestLimitExceeded**(*detail=None*)

Bases: *HeatAPIException*

Payload size of the request exceeds maximum allowed size.

code = 400

explanation = 'Payload exceeds maximum allowed size'

title = 'RequestLimitExceeded'

exception heat.api.aws.exception.**HeatServiceUnavailableError**(*detail=None*)

Bases: *HeatAPIException*

The request has failed due to a temporary failure of the server.

code = 503

err_type = 'Server'

explanation = 'Service temporarily unavailable'

title = 'ServiceUnavailable'

exception heat.api.aws.exception.**HeatSignatureError**(*detail=None*)

Bases: *HeatAPIException*

Authentication fails due to a bad signature.

code = 403

explanation = 'The request signature we calculated does not match the signature you provided'

title = 'SignatureDoesNotMatch'

exception heat.api.aws.exception.**HeatThrottlingError**(*detail=None*)

Bases: *HeatAPIException*

Request was denied due to request throttling.

code = 400

explanation = 'Request was denied due to request throttling'

title = 'Throttling'

heat.api.aws.exception.**map_remote_error**(*ex*)

Map rpc_common.RemoteError exceptions to HeatAPIException subclasses.

Map rpc_common.RemoteError exceptions returned by the engine to HeatAPIException subclasses which can be used to return properly formatted AWS error responses.

heat.api.aws.utils module

Helper utilities related to the AWS API implementations.

heat.api.aws.utils.**extract_param_list**(*params*, *prefix=''*)

Extract a list-of-dicts based on parameters containing AWS style list.

MetricData.member.1.MetricName=buffers MetricData.member.1.Unit=Bytes MetricData.member.1.Value=231434333 MetricData.member.2.MetricName=buffers2 MetricData.member.2.Unit=Bytes MetricData.member.2.Value=12345

This can be extracted by passing prefix=MetricData, resulting in a list containing two dicts.

heat.api.aws.utils.**extract_param_pairs**(*params*, *prefix=''*, *keyname=''*, *valuename=''*)

Extract user input params from AWS style parameter-pair encoded list.

In the AWS API list items appear as two key-value pairs (passed as query parameters) with keys of the form below:

Prefix.member.1.keyname=somekey Prefix.member.1.keyvalue=somevalue Prefix.member.2.keyname=anotherkey Prefix.member.2.keyvalue=somevalue

We reformat this into a dict here to match the heat engine API expected format.

heat.api.aws.utils.**format_response**(*action*, *response*)

Format response from engine into API format.

heat.api.aws.utils.**get_param_value**(*params*, *key*)

Looks up an expected parameter in a parsed params dict.

Helper function, looks up an expected parameter in a parsed params dict and returns the result. If params does not contain the requested key we raise an exception of the appropriate type.

heat.api.aws.utils.**reformat_dict_keys**(*keymap=None*, *inputdict=None*)

Utility function for mapping one dict format to another.

Module contents

heat.api.cfn package

Subpackages

heat.api.cfn.v1 package

Submodules

heat.api.cfn.v1.signal module

class heat.api.cfn.v1.signal.**SignalController**(*options*)

Bases: object

signal(*req*, *arn*, *body=None*)

update_waitcondition(*req*, *body*, *arn*)

heat.api.cfn.v1.signal.**create_resource**(*options*)

Signal resource factory method.

heat.api.cfn.v1.stacks module

Stack endpoint for Heat CloudFormation v1 API.

class heat.api.cfn.v1.stacks.**StackController**(*options*)

Bases: object

WSGI controller for stacks resource in Heat CloudFormation v1 API.

Implements the API actions.

CREATE_OR_UPDATE_ACTION = ('CreateStack', 'UpdateStack')

CREATE_STACK = 'CreateStack'

UPDATE_STACK = 'UpdateStack'

cancel_update(*req*)

create(*req*)

create_or_update(*req*, *action=None*)

Implements CreateStack and UpdateStack API actions.

Create or update stack as defined in template file.

default(*req*, ***args*)

delete(*req*)

Implements the DeleteStack API action.

Deletes the specified stack.

describe(*req*)

Implements DescribeStacks API action.

Gets detailed information for a stack (or all stacks).

describe_stack_resource(*req*)

Implements the DescribeStackResource API action.

Return the details of the given resource belonging to the given stack.

describe_stack_resources(*req*)

Implements the DescribeStackResources API action.

Return details of resources specified by the parameters.

StackName: returns all resources belonging to the stack.

PhysicalResourceId: returns all resources belonging to the stack this resource is associated with.

Only one of the parameters may be specified.

Optional parameter:

LogicalResourceId: filter the resources list by the logical resource id.

estimate_template_cost(*req*)

Implements the EstimateTemplateCost API action.

Get the estimated monthly cost of a template.

events_list(*req*)

Implements the DescribeStackEvents API action.

Returns events related to a specified stack (or all stacks).

get_template(*req*)

Implements the GetTemplate API action.

Get the template body for an existing stack.

list(*req*)

Implements ListStacks API action.

Lists summary information for all stacks.

list_stack_resources(*req*)

Implements the ListStackResources API action.

Return summary of the resources belonging to the specified stack.

update(*req*)

validate_template(*req*)

Implements the ValidateTemplate API action.

Validates the specified template.

heat.api.cfn.v1.stacks.**create_resource**(*options*)

Stacks resource factory method.

Module contents

class heat.api.cfn.v1.**API**(*conf*, ***local_conf*)

Bases: *Router*

WSGI router for Heat CloudFormation v1 API requests.

Submodules

heat.api.cfn.versions module

Controller that returns information on the heat API versions.

Now its a subclass of module versions, because of identity with OpenStack module versions.

heat.api.cfn.wsgi module

WSGI script for heat-api-cfn.

Script for running heat-api-cfn under Apache2.

heat.api.cfn.wsgi.**init_application**()

Module contents

heat.api.cfn.**version_negotiation_filter**(*app*, *conf*, ***local_conf*)

heat.api.middleware package

Submodules

heat.api.middleware.fault module

A middleware that turns exceptions into parsable string.

Inspired by Cinders faultwrapper.

class heat.api.middleware.fault.**Fault**(*error*)

Bases: object

class heat.api.middleware.fault.**FaultWrapper**(*application*)

Bases: *Middleware*

Replace error body with something the client can parse.

```
error_map = {'ActionInProgress': <class 'webob.exc.HTTPConflict'>,
'AttributeError': <class 'webob.exc.HTTPBadRequest'>,
'CircularDependencyException': <class 'webob.exc.HTTPBadRequest'>,
'DownloadLimitExceeded': <class 'webob.exc.HTTPBadRequest'>,
'EntityNotFound': <class 'webob.exc.HTTPNotFound'>, 'EventSendFailed':
<class 'webob.exc.HTTPInternalServerError'>, 'Forbidden': <class
'webob.exc.HTTPForbidden'>, 'ImmutableParameterModified': <class
'webob.exc.HTTPBadRequest'>, 'IncompatibleObjectVersion': <class
'webob.exc.HTTPBadRequest'>, 'Invalid': <class
'webob.exc.HTTPBadRequest'>, 'InvalidBreakPointHook': <class
'webob.exc.HTTPBadRequest'>, 'InvalidEncryptionKey': <class
'webob.exc.HTTPInternalServerError'>, 'InvalidGlobalResource': <class
'webob.exc.HTTPInternalServerError'>, 'InvalidSchemaError': <class
'webob.exc.HTTPBadRequest'>, 'InvalidTemplateReference': <class
'webob.exc.HTTPBadRequest'>, 'InvalidTemplateSection': <class
'webob.exc.HTTPBadRequest'>, 'InvalidTemplateVersion': <class
'webob.exc.HTTPBadRequest'>, 'InvalidTenant': <class
'webob.exc.HTTPForbidden'>, 'MissingCredentialError': <class
'webob.exc.HTTPBadRequest'>, 'NotFound': <class
'webob.exc.HTTPNotFound'>, 'NotSupported': <class
'webob.exc.HTTPBadRequest'>, 'ObjectActionError': <class
'webob.exc.HTTPBadRequest'>, 'ObjectFieldInvalid': <class
'webob.exc.HTTPBadRequest'>, 'OrphanedObjectError': <class
'webob.exc.HTTPBadRequest'>, 'PhysicalResourceIDAmbiguity': <class
'webob.exc.HTTPBadRequest'>, 'PhysicalResourceNameAmbiguity': <class
'webob.exc.HTTPBadRequest'>, 'PropertyUnspecifiedError': <class
'webob.exc.HTTPBadRequest'>, 'ReadOnlyFieldError': <class
'webob.exc.HTTPBadRequest'>, 'RequestLimitExceeded': <class
'webob.exc.HTTPBadRequest'>, 'ResourceActionNotSupported': <class
'webob.exc.HTTPBadRequest'>, 'ResourceNotAvailable': <class
'webob.exc.HTTPNotFound'>, 'ResourcePropertyConflict': <class
'webob.exc.HTTPBadRequest'>, 'ResourceTypeUnavailable': <class
'webob.exc.HTTPBadRequest'>, 'RevertFailed': <class
'webob.exc.HTTPInternalServerError'>, 'ServerBuildFailed': <class
'webob.exc.HTTPInternalServerError'>, 'StackExists': <class
'webob.exc.HTTPConflict'>, 'StackValidationFailed': <class
'webob.exc.HTTPBadRequest'>, 'StopActionFailed': <class
'webob.exc.HTTPInternalServerError'>, 'UnknownUserParameter': <class
'webob.exc.HTTPBadRequest'>, 'UnsupportedObjectError': <class
'webob.exc.HTTPBadRequest'>, 'UserParameterMissing': <class
'webob.exc.HTTPBadRequest'>, 'ValueError': <class
'webob.exc.HTTPBadRequest'>}
```

process_request(*req*)

Called on each request.

If this returns None, the next application down the stack will be executed. If it returns a response then that response will be returned and execution will stop here.

heat.api.middleware.version_negotiation module

Inspects the requested URI for a version string and/or Accept headers.

Also attempts to negotiate an API controller to return.

class heat.api.middleware.version_negotiation.**VersionNegotiationFilter**(*version_controller*, *app*, *conf*, ***local_conf*)

Bases: *Middleware*

process_request(*req*)

Process Accept header or simply return correct API controller.

If there is a version identifier in the URI, return the correct API controller, otherwise, if we find an Accept: header, process it

Module contents

heat.api.openstack package

Subpackages

heat.api.openstack.v1 package

Subpackages

heat.api.openstack.v1.views package

Submodules

heat.api.openstack.v1.views.stacks_view module

heat.api.openstack.v1.views.stacks_view.**collection**(*req*, *stacks*, *count=None*, *include_project=False*)

heat.api.openstack.v1.views.stacks_view.**format_stack**(*req*, *stack*, *keys=None*, *include_project=False*)

heat.api.openstack.v1.views.views_common module

heat.api.openstack.v1.views.views_common.**get_collection_links**(*request*, *items*)

Retrieve next link, if applicable.

Module contents

Submodules

heat.api.openstack.v1.actions module

class heat.api.openstack.v1.actions.**ActionController**(*options*)

Bases: object

WSGI controller for Actions in Heat v1 API.

Implements the API for stack actions

ACTIONS = ('suspend', 'resume', 'check', 'cancel_update', 'cancel_without_rollback')

CANCEL_UPDATE = 'cancel_update'

CANCEL_WITHOUT_ROLLBACK = 'cancel_without_rollback'

CHECK = 'check'

REQUEST_SCOPE = 'actions'

RESUME = 'resume'

SUSPEND = 'suspend'

action(*req*, *tenant_id*, *stack_name*, *stack_id*, *body=None*)

Performs a specified action on a stack.

The body is expecting to contain exactly one item whose key specifies the action.

cancel_update(*req*, *identity*, *body=None*)

cancel_without_rollback(*req*, *identity*, *body=None*)

check(*req*, *identity*, *body=None*)

resume(*req*, *identity*, *body=None*)

suspend(*req*, *identity*, *body=None*)

heat.api.openstack.v1.actions.**create_resource**(*options*)

Actions action factory method.

heat.api.openstack.v1.build_info module

class heat.api.openstack.v1.build_info.**BuildInfoController**(*options*)

Bases: object

WSGI controller for BuildInfo in Heat v1 API.

Returns build information for current app.

REQUEST_SCOPE = 'build_info'

build_info(*req*)

heat.api.openstack.v1.build_info.**create_resource**(*options*)

BuildInfo factory method.

heat.api.openstack.v1.events module

class heat.api.openstack.v1.events.**EventController**(*options*)

Bases: object

WSGI controller for Events in Heat v1 API.

Implements the API actions.

REQUEST_SCOPE = 'events'

index(*req*, *identity*, *resource_name=None*)

Lists summary information for all events.

show(*req*, *identity*, *resource_name*, *event_id*)

Gets detailed information for an event.

heat.api.openstack.v1.events.**create_resource**(*options*)

Events resource factory method.

heat.api.openstack.v1.events.**format_event**(*req*, *event*, *keys=None*)

heat.api.openstack.v1.resources module

class heat.api.openstack.v1.resources.**ResourceController**(*options*)

Bases: object

WSGI controller for Resources in Heat v1 API.

Implements the API actions.

REQUEST_SCOPE = 'resource'

index(*req*, *identity*)

Lists information for all resources.

mark_unhealthy(*req*, *identity*, *resource_name*, *body*)

Mark a resource as healthy or unhealthy.

metadata(*req*, *identity*, *resource_name*)

Gets metadata information for a resource.

show(*req*, *identity*, *resource_name*)

Gets detailed information for a resource.

signal(*req*, *identity*, *resource_name*, *body=None*)

heat.api.openstack.v1.resources.**create_resource**(*options*)

Resources resource factory method.

heat.api.openstack.v1.resources.**format_resource**(*req*, *res*, *keys=None*)

heat.api.openstack.v1.services module

class heat.api.openstack.v1.services.**ServiceController**(*options*)

Bases: object

WSGI controller for reporting the heat engine status in Heat v1 API.

REQUEST_SCOPE = 'service'

index(*req*)

heat.api.openstack.v1.services.**create_resource**(*options*)

heat.api.openstack.v1.software_configs module

class heat.api.openstack.v1.software_configs.**SoftwareConfigController**(*options*)

Bases: object

WSGI controller for Software config in Heat v1 API.

Implements the API actions.

REQUEST_SCOPE = 'software_configs'

create(*req*, *body*)

Create a new software config.

default(*req*, ***args*)

delete(*req*, *config_id*)

Delete an existing software config.

global_index(*req*)

index(*req*)

Lists summary information for all software configs.

show(*req*, *config_id*)

Gets detailed information for a software config.

heat.api.openstack.v1.software_configs.**create_resource**(*options*)

Software configs resource factory method.

heat.api.openstack.v1.software_deployments module

class heat.api.openstack.v1.software_deployments.**SoftwareDeploymentController**(*options*)

Bases: object

WSGI controller for Software deployments in Heat v1 API.

Implements the API actions.

REQUEST_SCOPE = 'software_deployments'

create(*req*, *body*)

Create a new software deployment.

default(*req*, ***args*)

delete(*req*, *deployment_id*)

Delete an existing software deployment.

index(*req*)

List software deployments.

metadata(*req*, *server_id*)

List software deployments grouped by the group name.

This is done for the requested server.

show(*req*, *deployment_id*)

Gets detailed information for a software deployment.

update(*req*, *deployment_id*, *body*)

Update an existing software deployment.

heat.api.openstack.v1.software_deployments.**create_resource**(*options*)

Software deployments resource factory method.

heat.api.openstack.v1.stacks module

Stack endpoint for Heat v1 REST API.

class heat.api.openstack.v1.stacks.**InstantiationData**(*data*, *patch=False*)

Bases: object

The data to create or update a stack.

The data accompanying a PUT or POST request.

PARAMS = ('stack_name', 'template', 'template_url', 'parameters', 'environment', 'files', 'environment_files', 'files_container')

PARAM_ENVIRONMENT = 'environment'

PARAM_ENVIRONMENT_FILES = 'environment_files'

PARAM_FILES = 'files'

PARAM_FILES_CONTAINER = 'files_container'

PARAM_STACK_NAME = 'stack_name'

PARAM_TEMPLATE = 'template'

PARAM_TEMPLATE_URL = 'template_url'

PARAM_USER_PARAMS = 'parameters'

args()

Get any additional arguments supplied by the user.

environment()

Get the user-supplied environment for the stack in YAML format.

If the user supplied Parameters then merge these into the environment global options.

environment_files()

files()

files_container()

no_change()

static parse_error_check(*data_type*)

stack_name()

Return the stack name.

template()

Get template file contents.

Get template file contents, either inline, from stack adopt data or from a URL, in JSON or YAML format.

class heat.api.openstack.v1.stacks.**StackController**(*options*)

Bases: object

WSGI controller for stacks resource in Heat v1 API.

Implements the API actions.

REQUEST_SCOPE = 'stacks'

abandon(*req*, *identity*)

Abandons specified stack.

Abandons specified stack by deleting the stack and its resources from the database, but underlying resources will not be deleted.

create(*req*, *body*)

Create a new stack.

default(*req*, ***args*)

delete(*req*, *identity*)

Delete the specified stack.

delete_snapshot(*req*, *identity*, *snapshot_id*)

detail(*req*)

Lists detailed information for all stacks.

environment(*req*, *identity*)

Get the environment for an existing stack.

export(*req*, *identity*)

Export specified stack.

Return stack data in JSON format.

files(*req*, *identity*)

Get the files for an existing stack.

generate_template(*req*, *type_name*)

Generates a template based on the specified type.

global_index(*req*)

index(*req*)

Lists summary information for all stacks.

list_outputs(*req*, *identity*)

list_resource_types(*req*)

Returns a resource types list which may be used in template.

list_snapshots(*req*, *identity*)

list_template_functions(*req*, *template_version*)

Returns a list of available functions in a given template.

list_template_versions(*req*)

Returns a list of available template versions.

lookup(*req*, *stack_name*, *path=''*, *body=None*)

Redirect to the canonical URL for a stack.

prepare_args(*data*, *is_update=False*)

preview(*req*, *body*)

Preview the outcome of a template and its params.

preview_update(*req*, *identity*, *body*)

Preview update for existing stack with a new template/parameters.

preview_update_patch(*req*, *identity*, *body*)

Preview PATCH update for existing stack.

resource_schema(*req*, *type_name*, *with_description=False*)

Returns the schema of the given resource type.

restore_snapshot(*req*, *identity*, *snapshot_id*)

show(*req*, *identity*)

Gets detailed information for a stack.

show_output(*req*, *identity*, *output_key*)

show_snapshot(*req*, *identity*, *snapshot_id*)

snapshot(*req*, *identity*, *body*)

template(*req*, *identity*)

Get the template body for an existing stack.

update(*req*, *identity*, *body*)

Update an existing stack with a new template and/or parameters.

update_patch(*req*, *identity*, *body*)

Update an existing stack with a new template.

Update an existing stack with a new template by patching the parameters Add the flag patch to the args so the engine code can distinguish

validate_template(*req*, *body*)

Implements the ValidateTemplate API action.

Validates the specified template.

class heat.api.openstack.v1.stacks.**StackSerializer**

Bases: *JSONResponseSerializer*

Handles serialization of specific controller method responses.

create(*response*, *result*)

heat.api.openstack.v1.stacks.**create_resource**(*options*)

Stacks resource factory method.

heat.api.openstack.v1.util module

heat.api.openstack.v1.util.**get_allowed_params**(*params*, *param_types*)

Extract from `params` all entries listed in `param_types`.

The returning dict will contain an entry for a key if, and only if, theres an entry in `param_types` for that key and at least one entry in `params`. If `params` contains multiple entries for the same key, it will yield an array of values: `{key: [v1, v2,...]}`

Parameters

- **params** a NestedMultiDict from webob.Request.params
- **param_types** an dict of allowed parameters and their types

Returns

a dict with {key: value} pairs

heat.api.openstack.v1.util.**make_link**(*req*, *identity*, *relationship='self'*)

Return a link structure for the supplied identity dictionary.

heat.api.openstack.v1.util.**make_url**(*req*, *identity*)

Return the URL for the supplied identity dictionary.

heat.api.openstack.v1.util.**no_policy_enforce**(*handler*)

Decorator that does *not* enforce policies.

Checks the path matches the request context.

This is a handler method decorator.

heat.api.openstack.v1.util.**registered_identified_stack**(*handler*)

Decorator that passes a stack identifier instead of path components.

This is a handler method decorator. Policy is enforced using a registered policy name.

`heat.api.openstack.v1.util.registered_policy_enforce(handler)`

Decorator that enforces policies.

Checks the path matches the request context and enforce policy defined in policies.

This is a handler method decorator.

Module contents

`class heat.api.openstack.v1.API(conf, **local_conf)`

Bases: *Router*

WSGI router for Heat v1 REST API requests.

Submodules

heat.api.openstack.versions module

Controller that returns information on the heat API versions.

Now its a subclass of module versions, because of identity with cfn module versions. It can be changed, if there will be another API version.

heat.api.openstack.wsgi module

WSGI script for heat-api.

Script for running heat-api under Apache2.

`heat.api.openstack.wsgi.init_application()`

Module contents

`heat.api.openstack.faultwrap_filter(app, conf, **local_conf)`

`heat.api.openstack.version_negotiation_filter(app, conf, **local_conf)`

Submodules

heat.api.versions module

Controller that returns information on the heat API versions.

`class heat.api.versions.Controller(conf)`

Bases: `object`

A controller that produces information on the heat API versions.

`get_href(req)`

Module contents

`heat.api.pipeline_factory(loader, global_conf, **local_conf)`

A paste pipeline replica that keys off of deployment flavor.

`heat.api.root_app_factory(loader, global_conf, **local_conf)`

heat.common package

Submodules

heat.common.auth_password module

class heat.common.auth_password.**KeystonePasswordAuthProtocol**(*app*, *conf*)

Bases: object

Middleware uses username and password to authenticate against Keystone.

Alternative authentication middleware that uses username and password to authenticate against Keystone instead of validating existing auth token. The benefit being that you no longer require admin/service token to authenticate users.

heat.common.auth_password.**filter_factory**(*global_conf*, ***local_conf*)

Returns a WSGI filter app for use with paste.deploy.

heat.common.auth_plugin module

heat.common.auth_plugin.**get_keystone_plugin_loader**(*auth*, *keystone_session*)

heat.common.auth_plugin.**parse_auth_credential_to_dict**(*cred*)

Parse credential to dict

heat.common.auth_plugin.**validate_auth_plugin**(*auth_plugin*, *keystone_session*)

Validate if this auth_plugin is valid to use.

heat.common.auth_url module

class heat.common.auth_url.**AuthUrlFilter**(*app*, *conf*)

Bases: *Middleware*

property auth_url

process_request(*req*)

Called on each request.

If this returns None, the next application down the stack will be executed. If it returns a response then that response will be returned and execution will stop here.

heat.common.auth_url.**filter_factory**(*global_conf*, ***local_conf*)

heat.common.cache module

The code related to integration between oslo.cache module and heat.

heat.common.cache.**get_cache_region**()

heat.common.cache.**list_opts**()

heat.common.cache.**register_cache_configurations**(*conf*)

Register all configurations required for oslo.cache.

The procedure registers all configurations required for oslo.cache. It should be called before configuring of cache region

Parameters
conf instance of heat configuration

Returns
updated heat configuration

heat.common.config module

Routines for configuring Heat.

`heat.common.config.get_client_option(client, option, fallback=True)`

`heat.common.config.get_ssl_options(client)`

`heat.common.config.list_opts()`

`heat.common.config.load_paste_app(app_name=None)`

Builds and returns a WSGI app from a paste config file.

We assume the last config file specified in the supplied ConfigOpts object is the paste config file.

Parameters
app_name name of the application to load

Raises
RuntimeError when config file cannot be located or application cannot be loaded from config file

`heat.common.config.set_config_defaults()`

This method updates all configuration default values.

`heat.common.config.startup_sanity_check()`

heat.common.context module

`class heat.common.context.ContextMiddleware(app, conf, **local_conf)`

Bases: *Middleware*

`process_request(req)`

Constructs an appropriate context from extracted auth information.

Extract any authentication information in the request and construct an appropriate context from it.

`heat.common.context.ContextMiddleware_filter_factory(global_conf, **local_conf)`

Factory method for paste.deploy.

`class heat.common.context.RequestContext(username=None, password=None, aws_creds=None, auth_url=None, is_admin=None, trust_id=None, trustor_user_id=None, auth_token_info=None, region_name=None, auth_plugin=None, trusts_auth_plugin=None, **kwargs)`

Bases: `RequestContext`

Stores information about the security context.

Under the security context the user accesses the system, as well as additional request information.

FROM_DICT_EXTRA_KEYS: list[str] = ['username', 'password', 'aws_creds', 'trust_id', 'trustor_user_id', 'auth_token_info', 'auth_url', 'region_name']

property auth_plugin

cache(*cache_cls*)

property clients

property connection

classmethod from_dict(*values*, ***kwargs*)

Construct a context object from a provided dictionary.

property keystone_session

property keystone_v3_endpoint

reload_auth_plugin()

property session

to_dict()

Return a dictionary of context attributes.

to_policy_values()

A dictionary of context attributes to enforce policy with.

oslo.policy enforcement requires a dictionary of attributes representing the current logged in user on which it applies policy enforcement. This dictionary defines a standard list of attributes that should be available for enforcement across services.

It is expected that services will often have to override this method with either deprecated values or additional attributes used by that service specific policy.

property transaction

property transaction_ctx

property trusts_auth_plugin

class heat.common.context.**StoredContext**(*username=None*, *password=None*, *aws_creds=None*, *auth_url=None*, *is_admin=None*, *trust_id=None*, *trustor_user_id=None*, *auth_token_info=None*, *region_name=None*, *auth_plugin=None*, *trusts_auth_plugin=None*, ***kwargs*)

Bases: *RequestContext*

property project_domain_id

property roles

property user_domain_id

heat.common.context.**get_admin_context**(*show_deleted=False*)

heat.common.context.**list_opts**()

heat.common.context.**request_context**(*func*)

heat.common.crypt module

class heat.common.crypt.**SymmetricCrypto**(*enctype='AES'*)

Bases: object

Symmetric Key Crypto object.

This class creates a Symmetric Key Crypto object that can be used to decrypt arbitrary data.

Note: This is a reimplementation of the decryption algorithm from oslo-incubator, and is provided for backward compatibility. Once we have a DB migration script available to re-encrypt using new encryption method as part of upgrade, this can be removed.

Parameters

enctype Encryption Cipher name (default: AES)

decrypt(*key*, *msg*, *b64decode=True*)

Decrypts the provided ciphertext.

The ciphertext can be optionally base64 encoded.

Uses AES-128-CBC with an IV by default.

Parameters

- **key** The Encryption key.
- **msg** the ciphetext, the first block is the IV

Returns

the plaintext message, after padding is removed.

heat.common.crypt.**cryptography_decrypt_v1**(*value*, *encryption_key=None*)

heat.common.crypt.**decrypt**(*method*, *data*, *encryption_key=None*)

heat.common.crypt.**decrypted_dict**(*data*, *encryption_key=None*)

Return a decrypted dict. Assume input values are encrypted json fields.

heat.common.crypt.**encrypt**(*value*, *encryption_key=None*)

heat.common.crypt.**encrypted_dict**(*data*, *encryption_key=None*)

Return an encrypted dict. Values converted to json before encrypted

heat.common.crypt.**get_valid_encryption_key**(*encryption_key*, *fix_length=False*)

heat.common.crypt.**heat_decrypt**(*value*, *encryption_key=None*)

Decrypt data that has been encrypted using an older version of Heat.

Note: the encrypt function returns the function that is needed to decrypt the data. The database then stores this. When the data is then retrieved (potentially by a later version of Heat) the decrypt function must still exist. So whilst it may seem that this function is not referenced, it will be referenced from the database.

heat.common.crypt.**list_opts**()

`heat.common.crypt.oslo_decrypt_v1(value, encryption_key=None)`

heat.common.custom_backend_auth module

Middleware for authenticating against custom backends.

`class heat.common.custom_backend_auth.AuthProtocol(app, conf)`

Bases: `object`

`heat.common.custom_backend_auth.filter_factory(global_conf, **local_conf)`

heat.common.endpoint_utils module

`heat.common.endpoint_utils.get_auth_uri(v3=True)`

heat.common.environment_format module

`heat.common.environment_format.default_for_missing(env)`

Checks a parsed environment for missing sections.

`heat.common.environment_format.parse(env_str)`

Takes a string and returns a dict containing the parsed structure.

`heat.common.environment_format.validate(env)`

heat.common.environment_util module

`heat.common.environment_util.get_param_merge_strategy(merge_strategies, param_key, available_strategies=None)`

`heat.common.environment_util.merge_environments(environment_files, files, params, param_schemata)`

Merges environment files into the stack input parameters.

If a list of environment files have been specified, this call will pull the contents of each from the files dict, parse them as environments, and merge them into the stack input params. This behavior is the same as earlier versions of the Heat client that performed this params population client-side.

Parameters

- **environment_files** (*list or None*) ordered names of the environment files found in the files dict
- **files** (*dict*) mapping of stack filenames to contents
- **params** (*dict*) parameters describing the stack
- **param_schemata** (*dict*) parameter schema dict

`heat.common.environment_util.merge_list(old, new)`

merges lists and comma delimited lists.

`heat.common.environment_util.merge_map(old, new, deep_merge=False)`

Merge nested dictionaries.

heat.common.environment_util.**merge_parameters**(*old*, *new*, *param_schemata*, *strategies_in_file*, *available_strategies*, *env_file*)

heat.common.environment_util.**parse_param**(*p_val*, *p_schema*)

heat.common.exception module

Heat exception subclasses

exception heat.common.exception.**ActionInProgress**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Stack %(stack_name)s already has an action (%(action)s) in progress.'

exception heat.common.exception.**ActionNotComplete**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Stack %(stack_name)s has an action (%(action)s) in progress or failed state.'

exception heat.common.exception.**AuthorizationFailure**(*failure_reason=''*)

Bases: *HeatException*

msg_fmt = 'Authorization failed.%(failure_reason)s'

exception heat.common.exception.**CircularDependencyException**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Circular Dependency Found: %(cycle)s'

exception heat.common.exception.**ClientNotAvailable**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The client (%(client_name)s) is not available.'

exception heat.common.exception.**ConcurrentTransaction**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Concurrent transaction for %(action)s'

exception heat.common.exception.**ConflictingMergeStrategyForParam**(***kwargs*)

Bases: *HeatException*

msg_fmt = "Conflicting merge strategy '%(strategy)s' for parameter '%(param)s' in file '%(env_file)s'."

exception heat.common.exception.**DownloadLimitExceeded**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Permissible download limit exceeded: %(message)s'

exception heat.common.exception.**EgressRuleNotAllowed**(***kwargs*)

Bases: *HeatException*

msg_fmt = "Egress rules are only allowed when Neutron is used and the 'VpcId' property is set."

exception heat.common.exception.**EntityNotFound**(*entity=None*, *name=None*, ***kwargs*)

Bases: *HeatException*

msg_fmt = 'The %(entity)s (%(name)s) could not be found.'

exception heat.common.exception.**Error**(*msg*)

Bases: *HeatException*

msg_fmt = '%(message)s'

exception heat.common.exception.**EventSendFailed**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Failed to send message to stack (%(stack_name)s) on other engine (%(engine_id)s)'

exception heat.common.exception.**Forbidden**(*action='this action'*)

Bases: *HeatException*

msg_fmt = 'You are not authorized to use %(action)s.'

exception heat.common.exception.**HTTPExceptionDisguise**(*exception*)

Bases: Exception

Disguises HTTP exceptions.

They can be handled by the webob fault application in the wsgi pipeline.

safe = True

exception heat.common.exception.**HeatException**(***kwargs*)

Bases: Exception

Base Heat Exception.

To correctly use this class, inherit from it and define a msg_fmt property. That msg_fmt will get formatted with the keyword arguments provided to the constructor.

error_code = None

message = 'An unknown exception occurred.'

safe = True

exception heat.common.exception.**HeatExceptionWithPath**(*error=None*, *path=None*, *message=None*)

Bases: *HeatException*

msg_fmt = '%(error)s%(path)s%(message)s'

exception heat.common.exception.**ImmutableParameterModified**(**args*, ***kwargs*)

Bases: *HeatException*

msg_fmt = 'The following parameters are immutable and may not be updated: %(keys)s'

exception heat.common.exception.**IncompatibleObjectVersion**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Version %(objver)s of %(objname)s is not supported'

exception heat.common.exception.**InterfaceAttachFailed**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Failed to attach interface (%(port)s) to server (%(server)s)'

exception heat.common.exception.**InterfaceDetachFailed**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Failed to detach interface (%(port)s) from server (%(server)s)'

exception heat.common.exception.**Invalid**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Data supplied was not valid: %(reason)s'

exception heat.common.exception.**InvalidBreakPointHook**(***kwargs*)

Bases: *HeatException*

msg_fmt = '%(message)s'

exception heat.common.exception.**InvalidContentType**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Invalid content type %(content_type)s'

exception heat.common.exception.**InvalidEncryptionKey**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Can not decrypt data with the auth_encryption_key in heat config.'

exception heat.common.exception.**InvalidExternalResourceDependency**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Invalid dependency with external %(resource_type)s resource: %(external_id)s'

exception heat.common.exception.**InvalidGlobalResource**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'There was an error loading the definition of the global resource type %(type_name)s.'

exception heat.common.exception.**InvalidMergeStrategyForParam**(***kwargs*)

Bases: *HeatException*

msg_fmt = "Invalid merge strategy '%(strategy)s' for parameter '%(param)s'."

exception heat.common.exception.**InvalidRestrictedAction**(***kwargs*)

Bases: *HeatException*

msg_fmt = '%(message)s'

exception heat.common.exception.**InvalidSchemaError**(***kwargs*)

Bases: *HeatException*

msg_fmt = '%(message)s'

exception heat.common.exception.**InvalidServiceVersion**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Invalid service %(service)s version %(version)s'

exception heat.common.exception.**InvalidTemplateAttribute**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The Referenced Attribute (%(resource)s %(key)s) is incorrect.'

exception heat.common.exception.**InvalidTemplateReference**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The specified reference "%(resource)s" (in %(key)s) is incorrect.'

exception heat.common.exception.**InvalidTemplateSection**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The template section is invalid: %(section)s'

exception heat.common.exception.**InvalidTemplateVersion**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The template version is invalid: %(explanation)s'

exception heat.common.exception.**InvalidTemplateVersions**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'A template version alias %(version)s was added for a template class that has no official YYYY-MM-DD version.'

exception heat.common.exception.**InvalidTenant**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Searching Tenant %(target)s from Tenant %(actual)s forbidden.'

exception heat.common.exception.**KeystoneServiceNameConflict**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Keystone has more than one service with same name %(service)s. Please use service id instead of name'

exception heat.common.exception.**MissingCredentialError**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Missing required credential: %(required)s'

exception heat.common.exception.**NotAuthenticated**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'You are not authenticated.'

exception heat.common.exception.**NotAuthorized**(*action='this action'*)

Bases: *Forbidden*

msg_fmt = 'You are not authorized to complete this action.'

exception heat.common.exception.**NotFound**(*msg_fmt='Not found'*)

Bases: *HeatException*

exception heat.common.exception.**NotSupported**(***kwargs*)

Bases: *HeatException*

msg_fmt = '%(feature)s is not supported.'

exception heat.common.exception.**ObjectActionError**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Object action %(action)s failed because: %(reason)s'

exception heat.common.exception.**ObjectFieldInvalid**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Field %(field)s of %(objname)s is not an instance of Field'

exception heat.common.exception.**OrphanedObjectError**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Cannot call %(method)s on orphaned %(objtype)s object'

exception heat.common.exception.**PhysicalResourceExists**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The physical resource for (%(name)s) exists.'

exception heat.common.exception.**PhysicalResourceIDAmbiguity**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Multiple resources were found with the physical ID (%(phys_id)s).'

exception heat.common.exception.**PhysicalResourceNameAmbiguity**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Multiple physical resources were found with name (%(name)s).'

exception heat.common.exception.**PropertyUnspecifiedError**(**args*, ***kwargs*)

Bases: *HeatException*

msg_fmt = 'At least one of the following properties must be specified: %(props)s.'

exception heat.common.exception.**ReadOnlyFieldError**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Cannot modify readonly field %(field)s'

exception heat.common.exception.**RequestLimitExceeded**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Request limit exceeded: %(message)s'

exception heat.common.exception.**ResourceActionNotSupported**(***kwargs*)

Bases: *HeatException*

msg_fmt = '%(action)s is not supported for resource.'

exception heat.common.exception.**ResourceActionRestricted**(***kwargs*)

Bases: *HeatException*

msg_fmt = '%(action)s is restricted for resource.'

exception heat.common.exception.**ResourceFailure**(*exception_or_error*, *resource*, *action=None*)

Bases: *HeatExceptionWithPath*

exception heat.common.exception.**ResourceInError**(*status_reason='Unknown'*, ***kwargs*)

Bases: *HeatException*

msg_fmt = 'Went to status %(resource_status)s due to "%(status_reason)s"'

exception heat.common.exception.**ResourceNotAvailable**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The Resource (%(resource_name)s) is not available.'

exception heat.common.exception.**ResourceNotFound**(*entity=None*, *name=None*, ***kwargs*)

Bases: *EntityNotFound*

msg_fmt = 'The Resource (%(resource_name)s) could not be found in Stack %(stack_name)s.'

exception heat.common.exception.**ResourcePropertyConflict**(**args*, ***kwargs*)

Bases: *HeatException*

msg_fmt = 'Cannot define the following properties at the same time: %(props)s.'

exception heat.common.exception.**ResourcePropertyDependency**(***kwargs*)

Bases: *HeatException*

msg_fmt = '%(prop1)s cannot be specified without %(prop2)s.'

exception heat.common.exception.**ResourcePropertyValueDependency**(***kwargs*)

Bases: *HeatException*

msg_fmt = '%(prop1)s property should only be specified for %(prop2)s with value %(value)s.'

exception heat.common.exception.**ResourceTypeUnavailable**(***kwargs*)

Bases: *HeatException*

error_code = '99001'

exception heat.common.exception.**ResourceUnknownStatus**(*result='Resource failed', status_reason='Unknown', **kwargs*)

Bases: *HeatException*

msg_fmt = '%(result)s - Unknown status %(resource_status)s due to "%(status_reason)s"'

exception heat.common.exception.**SIGHUPInterrupt**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'System SIGHUP signal received.'

exception heat.common.exception.**SnapshotNotFound**(*entity=None, name=None, **kwargs*)

Bases: *EntityNotFound*

msg_fmt = 'The Snapshot (%(snapshot)s) for Stack (%(stack)s) could not be found.'

exception heat.common.exception.**StackExists**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The Stack (%(stack_name)s) already exists.'

exception heat.common.exception.**StackResourceLimitExceeded**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Maximum resources per stack exceeded.'

exception heat.common.exception.**StackValidationFailed**(*error=None, path=None, message=None, resource=None*)

Bases: *HeatExceptionWithPath*

exception heat.common.exception.**StopActionFailed**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Failed to stop stack (%(stack_name)s) on other engine (%(engine_id)s)'

exception heat.common.exception.**TemplateOutputError**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Error in %(resource)s output %(attribute)s: %(message)s'

exception heat.common.exception.**UnableToAutoAllocateNetwork**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Unable to automatically allocate a network: %(message)s'

exception heat.common.exception.**UnknownUserParameter**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The Parameter (%(key)s) was not defined in template.'

exception heat.common.exception.**UnsupportedObjectError**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'Unsupported object type %(objtype)s'

exception heat.common.exception.**UpdateInProgress**(*resource_name='Unknown'*)

Bases: Exception

exception heat.common.exception.**UpdateReplace**(*resource_name='Unknown'*)

Bases: Exception

Raised when resource update requires replacement.

exception heat.common.exception.**UserParameterMissing**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'The Parameter (%(key)s) was not provided.'

heat.common.grouputils module

class heat.common.grouputils.**GroupInspector**(*context*, *rpc_client*, *group_identity*)

Bases: object

A class for returning data about a scaling group.

All data is fetched over RPC, and the groups stack is never loaded into memory locally. Data is cached so it will be fetched only once. To refresh the data, create a new GroupInspector.

classmethod from_parent_resource(*parent_resource*)

Create a GroupInspector from a parent resource.

This is a convenience method to instantiate a GroupInspector from a Heat StackResource object.

member_names(*include_failed*)

Return an iterator over the names of the group members

If include_failed is False, only members not in a FAILED state will be included.

size(*include_failed*)

Return the size of the group.

If include_failed is False, only members not in a FAILED state will be counted.

template()

Return a Template object representing the groups current template.

Note that this does *not* include any environment data.

heat.common.grouputils.**get_child_template_files**(*context*, *stack*, *is_rolling_update*, *old_template_id*)

Return a merged map of old and new template files.

For rolling update files for old and new defintions are required as the nested stack is updated in batches of scaled units.

heat.common.grouputils.**get_member_definitions**(*group*, *include_failed=False*)

Get member definitions in (name, ResourceDefinition) pair for group.

The List is sorted first by created_time then by name. If include_failed is set, failed members will be put first in the List sorted by created_time then by name.

heat.common.grouputils.**get_member_names**(*group*)

Get a list of resource names of the resources in the specified group.

Failed resources will be ignored.

heat.common.grouputils.**get_member_refids**(*group*)

Get a list of member resources managed by the specified group.

The list of resources is sorted first by created_time then by name.

heat.common.grouputils.**get_members**(*group*, *include_failed=False*)

Get a list of member resources managed by the specified group.

Sort the list of instances first by created_time then by name. If include_failed is set, failed members will be put first in the list sorted by created_time then by name.

heat.common.grouputils.**get_nested_attrs**(*stack*, *key*, *use_indices*, **path*)

heat.common.grouputils.**get_resource**(*stack*, *resource_name*, *use_indices*, *key=None*)

heat.common.grouputils.**get_rsrc_attr**(*stack*, *key*, *use_indices*, *resource_name*, **attr_path*)

heat.common.grouputils.**get_rsrc_id**(*stack*, *key*, *use_indices*, *resource_name*)

heat.common.grouputils.**get_size**(*group*, *include_failed=False*)

Get number of member resources managed by the specified group.

The size excludes failed members by default; set include_failed=True to get the total size.

heat.common.i18n module

heat.common.identifier module

class heat.common.identifier.**EventIdentifier**(*tenant*, *stack_name*, *stack_id*, *path*, *event_id=None*)

Bases: *HeatIdentifier*

An identifier for an event.

EVENT_ID = 'event_id'

RESOURCE_NAME = 'resource_name'

resource()

Return a HeatIdentifier for the owning resource.

stack()

Return a HeatIdentifier for the owning stack.

class heat.common.identifier.**HeatIdentifier**(*tenant*, *stack_name*, *stack_id*, *path=''*)

Bases: Mapping

FIELDS = ('tenant', 'stack_name', 'stack_id', 'path')

PATH = 'path'

STACK_ID = 'stack_id'

STACK_NAME = 'stack_name'

TENANT = 'tenant'

arn()

Return as an ARN.

Returned in the form:
arn:openstack:heat::<tenant>:stacks/<stack_name>/<stack_id><path>

arn_url_path()

Return an ARN quoted correctly for use in a URL.

classmethod from_arn(*arn*)

Generate a new HeatIdentifier by parsing the supplied ARN.

classmethod from_arn_url(*url*)

Generate a new HeatIdentifier by parsing the supplied URL.

The URL is expected to contain a valid arn as part of the path.

path_re = re.compile('stacks/([^/]+)/([^/]+)(.*)')

stack_path()

Return a URL-encoded path segment of a URL without a tenant.

Returned in the form:
<stack_name>/<stack_id>

url_path()

Return a URL-encoded path segment of a URL.

Returned in the form:
<tenant>/stacks/<stack_name>/<stack_id><path>

class heat.common.identifier.ResourceIdentifier(*tenant*, *stack_name*, *stack_id*, *path*, *resource_name=None*)

Bases: *HeatIdentifier*

An identifier for a resource.

RESOURCE_NAME = 'resource_name'

stack()

Return a HeatIdentifier for the owning stack.

heat.common.lifecycle_plugin_utils module

Utility for fetching and running plug point implementation classes.

`heat.common.lifecycle_plugin_utils.do_post_ops(cnxt, stack, current_stack=None, action=None, is_stack_failure=False)`

Call available post-op methods sequentially.

In order determined with get_ordinal(), with parameters context, stack, current_stack, action, is_stack_failure.

`heat.common.lifecycle_plugin_utils.do_pre_ops(cnxt, stack, current_stack=None, action=None)`

Call available pre-op methods sequentially.

In order determined with get_ordinal(), with parameters context, stack, current_stack, action.

On failure of any pre_op method, will call post-op methods corresponding to successful calls of pre-op methods.

`heat.common.lifecycle_plugin_utils.get_plug_point_class_instances()`

Instances of classes that implements pre/post stack operation methods.

Get list of instances of classes that (may) implement pre and post stack operation methods.

The list of class instances is sorted using get_ordinal methods on the plug point classes. If class1.ordinal() < class2.ordinal(), then class1 will be before class2 in the list.

heat.common.messaging module

`class heat.common.messaging.RequestContextSerializer(base)`

Bases: `Serializer`

`static deserialize_context(ctxt)`

Deserialize a dictionary into a request context.

Parameters
ctxt Request context dictionary

Returns
Deserialized form of entity

`deserialize_entity(ctxt, entity)`

Deserialize something from primitive form.

Parameters

- **ctxt** Request context, in deserialized form
- **entity** Primitive to be deserialized

Returns
Deserialized form of entity

`static serialize_context(ctxt)`

Serialize a request context into a dictionary.

Parameters
ctxt Request context

Returns
Serialized form of context

serialize_entity(*ctxt*, *entity*)

Serialize something to primitive form.

Parameters

- **ctxt** Request context, in deserialized form
- **entity** Entity to be serialized

Returns

Serialized form of entity

heat.common.messaging.**cleanup**()

Cleanup the oslo_messaging layer.

heat.common.messaging.**get_notifier**(*publisher_id*)

Return a configured oslo_messaging notifier.

heat.common.messaging.**get_rpc_client**(***kwargs*)

Return a configured oslo_messaging RPCClient.

heat.common.messaging.**get_rpc_server**(*target*, *endpoint*)

Return a configured oslo_messaging rpc server.

heat.common.messaging.**get_specific_transport**(*url*, *optional*, *exmods*, *is_for_notifications=False*)

heat.common.messaging.**setup**(*url=None*, *optional=False*)

Initialise the oslo_messaging layer.

heat.common.messaging.**setup_transports**(*url*, *optional*)

heat.common.netutils module

heat.common.netutils.**is_prefix_subset**(*orig_prefixes*, *new_prefixes*)

Check whether orig_prefixes is subset of new_prefixes.

This takes valid prefix lists for orig_prefixes and new_prefixes, returns True, if orig_prefixes is subset of new_prefixes.

heat.common.netutils.**validate_dns_format**(*data*)

heat.common.noauth module

Middleware that accepts any authentication.

class heat.common.noauth.**NoAuthProtocol**(*app*, *conf*)

Bases: object

heat.common.noauth.**filter_factory**(*global_conf*, ***local_conf*)

heat.common.param_utils module

heat.common.param_utils.**delim_string_to_list**(*value*)

heat.common.param_utils.**extract_bool**(*name*, *value*)

Convert any true/false string to its corresponding boolean value.

Value is case insensitive.

heat.common.param_utils.**extract_int**(*name*, *value*, *allow_zero=True*, *allow_negative=False*)

heat.common.param_utils.**extract_tags**(*subject*)

heat.common.param_utils.**extract_template_type**(*subject*)

heat.common.password_gen module

class heat.common.password_gen.**CharClass**(*allowed_chars*, *min_count*)

Bases: tuple

allowed_chars

Alias for field number 0

min_count

Alias for field number 1

heat.common.password_gen.**generate_openstack_password**()

Generate a random password suitable for a Keystone User.

heat.common.password_gen.**generate_password**(*length*, *char_classes*)

Generate a random password.

The password will be of the specified length, and comprised of characters from the specified character classes, which can be generated using the *named_char_class()* and *special_char_class()* functions. Where a minimum count is specified in the character class, at least that number of characters in the resulting password are guaranteed to be from that character class.

Parameters

- **length** The length of the password to generate, in characters
- **char_classes** Iterable over classes of characters from which to generate a password

heat.common.password_gen.**named_char_class**(*char_class*, *min_count=0*)

Return a predefined character class.

The result of this function can be passed to *generate_password()* as one of the character classes to use in generating a password.

Parameters

- **char_class** Any of the character classes named in CHARACTER_CLASSES
- **min_count** The minimum number of members of this class to appear in a generated password

heat.common.password_gen.**special_char_class**(*allowed_chars*, *min_count=0*)

Return a character class containing custom characters.

The result of this function can be passed to *generate_password()* as one of the character classes to use in generating a password.

Parameters

- **allowed_chars** Iterable of the characters in the character class
- **min_count** The minimum number of members of this class to appear in a generated password

heat.common.plugin_loader module

Utilities to dynamically load plugin modules.

Modules imported this way remain accessible to static imports, regardless of the order in which they are imported. For modules that are not part of an existing package tree, use create_subpackage() to dynamically create a package for them before loading them.

`heat.common.plugin_loader.`**`create_subpackage`**(*path*, *parent_package_name*, *subpackage_name='plugins'*)

Dynamically create a package into which to load plugins.

This allows us to not include an __init__.py in the plugins directory. We must still create a package for plugins to go in, otherwise we get warning messages during import. This also provides a convenient place to store the path(s) to the plugins directory.

`heat.common.plugin_loader.`**`load_modules`**(*package*, *ignore_error=False*)

Dynamically load all modules from a given package.

heat.common.pluginutils module

`heat.common.pluginutils.`**`log_fail_msg`**(*manager*, *entrypoint*, *exception*)

heat.common.policy module

Policy Engine For Heat.

`class` `heat.common.policy.`**`Enforcer`**(*scope='heat'*, *exc=<class 'heat.common.exception.Forbidden'>*, *default_rule=<oslo_policy._checks.FalseCheck object>*, *policy_file=None*)

Bases: `object`

Responsible for loading and enforcing rules.

`check_is_admin`(*context*)

Whether or not is admin according to policy.

By default the rule will check whether or not roles contains admin role and is admin project.

param context
Heat request context

returns
A non-False value if the user is admin according to policy

`enforce`(*context*, *action*, *scope=None*, *target=None*, *is_registered_policy=False*)

Verifies that the action is valid on the target in this context.

Parameters

- **context** Heat request context
- **action** String representing the action to be checked
- **target** Dictionary representing the object of the action.

Raises
heat.common.exception.Forbidden When permission is denied (or self.exc if supplied).

Returns
A non-False value if access is allowed.

load_rules(*force_reload=False*)

Set the rules found in the json file on disk.

set_rules(*rules*, *overwrite=True*)

Create a new Rules object based on the provided dict of rules.

class heat.common.policy.**ResourceEnforcer**(*default_rule=<oslo_policy._checks.TrueCheck object>*, ***kwargs*)

Bases: *Enforcer*

enforce(*context*, *res_type*, *scope=None*, *target=None*, *is_registered_policy=False*)

Verifies that the action is valid on the target in this context.

Parameters

- **context** Heat request context
- **action** String representing the action to be checked
- **target** Dictionary representing the object of the action.

Raises
heat.common.exception.Forbidden When permission is denied (or self.exc if supplied).

Returns
A non-False value if access is allowed.

enforce_stack(*stack*, *scope=None*, *target=None*, *is_registered_policy=False*)

heat.common.policy.**get_enforcer**()

heat.common.policy.**get_policy_enforcer**()

heat.common.profiler module

heat.common.profiler.**setup**(*binary*, *host*)

heat.common.serializers module

Utility methods for serializing responses.

class heat.common.serializers.**JSONResponseSerializer**

Bases: object

default(*response*, *result*)

to_json(*data*)

class heat.common.serializers.**XMLResponseSerializer**

Bases: object

default(*response*, *result*)

object_to_element(*obj*, *element*)

to_xml(*data*)

heat.common.service_utils module

heat.common.service_utils.**engine_alive**(*context*, *engine_id*)

heat.common.service_utils.**format_service**(*service*)

heat.common.service_utils.**generate_engine_id**()

heat.common.short_id module

Utilities for creating short ID strings based on a random UUID.

The IDs each comprise 12 (lower-case) alphanumeric characters.

heat.common.short_id.**generate_id**()

Generate a short (12 character), random id.

heat.common.short_id.**get_id**(*source_uuid*)

Derive a short (12 character) id from a random UUID.

The supplied UUID must be a version 4 UUID object.

heat.common.template_format module

heat.common.template_format.**convert_json_to_yaml**(*json_str*)

Convert AWS JSON template format to Heat YAML format.

Parameters

json_str a string containing the AWS JSON template format.

Returns

the equivalent string containing the Heat YAML format.

heat.common.template_format.**parse**(*tmpl_str*, *tmpl_url=None*)

Takes a string and returns a dict containing the parsed structure.

This includes determination of whether the string is using the JSON or YAML format.

heat.common.template_format.**simple_parse**(*tmpl_str*, *tmpl_url=None*)

heat.common.template_format.**validate_template_limit**(*contain_str*)

Validate limit for the template.

Check if the contain exceeds allowed size range.

class heat.common.template_format.**yaml_dumper**(*stream*, *default_style=None*, *default_flow_style=False*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*, *encoding=None*, *explicit_start=None*, *explicit_end=None*, *version=None*, *tags=None*, *sort_keys=True*)

Bases: CSafeDumper

represent_ordered_dict(*data*)

yaml_representers = {<class 'NoneType'>: <function SafeRepresenter.represent_none>, <class 'bool'>: <function SafeRepresenter.represent_bool>, <class 'bytes'>: <function SafeRepresenter.represent_binary>, <class 'collections.OrderedDict'>: <function yaml_dumper.represent_ordered_dict>, <class 'datetime.date'>: <function SafeRepresenter.represent_date>, <class 'datetime.datetime'>: <function SafeRepresenter.represent_datetime>, <class 'dict'>: <function SafeRepresenter.represent_dict>, <class 'float'>: <function SafeRepresenter.represent_float>, <class 'int'>: <function SafeRepresenter.represent_int>, <class 'list'>: <function SafeRepresenter.represent_list>, <class 'set'>: <function SafeRepresenter.represent_set>, <class 'str'>: <function SafeRepresenter.represent_str>, <class 'tuple'>: <function SafeRepresenter.represent_list>, None: <function SafeRepresenter.represent_undefined>}

class heat.common.template_format.**yaml_loader**(*stream*)

Bases: CSafeLoader

yaml_constructors = {'tag:yaml.org,2002:binary': <function SafeConstructor.construct_yaml_binary>, 'tag:yaml.org,2002:bool': <function SafeConstructor.construct_yaml_bool>, 'tag:yaml.org,2002:float': <function SafeConstructor.construct_yaml_float>, 'tag:yaml.org,2002:int': <function SafeConstructor.construct_yaml_int>, 'tag:yaml.org,2002:map': <function SafeConstructor.construct_yaml_map>, 'tag:yaml.org,2002:null': <function SafeConstructor.construct_yaml_null>, 'tag:yaml.org,2002:omap': <function SafeConstructor.construct_yaml_omap>, 'tag:yaml.org,2002:pairs': <function SafeConstructor.construct_yaml_pairs>, 'tag:yaml.org,2002:seq': <function SafeConstructor.construct_yaml_seq>, 'tag:yaml.org,2002:set': <function SafeConstructor.construct_yaml_set>, 'tag:yaml.org,2002:str': <function yaml_loader._construct_yaml_str>, 'tag:yaml.org,2002:timestamp': <function yaml_loader._construct_yaml_str>, None: <function SafeConstructor.construct_undefined>}

heat.common.timeutils module

Utilities for handling ISO 8601 duration format.

class heat.common.timeutils.**Duration**(*timeout=0*)

Bases: object

endtime()

expired()

heat.common.timeutils.**isotime**(*at*)

Stringify UTC time in ISO 8601 format.

Parameters

at Timestamp in UTC to format.

heat.common.timeutils.**parse_isoduration**(*duration*)

Convert duration in ISO 8601 format to second(s).

Year, Month, Week, and Day designators are not supported. Example: PT12H30M5S

heat.common.timeutils.**retry_backoff_delay**(*attempt*, *scale_factor=1.0*, *jitter_max=0.0*)

Calculate an exponential backoff delay with jitter.

Delay is calculated as 2^attempt + (uniform random from [0,1) * jitter_max)

Parameters

- **attempt** The count of the current retry attempt
- **scale_factor** Multiplier to scale the exponential delay by
- **jitter_max** Maximum of random seconds to add to the delay

Returns

Seconds since epoch to wait until

heat.common.urlfetch module

Utility for fetching a resource (e.g. a template) from a URL.

exception heat.common.urlfetch.**URLFetchError**(*msg*)

Bases: *Error*, OSError

heat.common.urlfetch.**get**(*url*, *allowed_schemes=('http', 'https')*)

Get the data at the specified URL.

The URL must use the http: or https: schemes. The file: scheme is also supported if you override the allowed_schemes argument. Raise an IOError if getting the data fails.

heat.common.wsgi module

Utility methods for working with WSGI servers.

class heat.common.wsgi.**AppFactory**(*conf*)

Bases: *BasePasteFactory*

A Generic paste.deploy app factory.

This requires heat.app_factory to be set to a callable which returns a WSGI app when invoked. The format of the name is <module>:<callable> e.g.

[app:apiv1app] paste.app_factory = heat.common.wsgi:app_factory heat.app_factory = heat.api.cfn.v1:API

The WSGI app constructor must accept a ConfigOpts object and a local config dict as its two arguments.

KEY = 'heat.app_factory'

class heat.common.wsgi.**BasePasteFactory**(*conf*)

Bases: object

A base class for paste app and filter factories.

Sub-classes must override the KEY class attribute and provide a __call__ method.

KEY = None

class heat.common.wsgi.**Debug**(*application*)

Bases: *Middleware*

Helper class to get information about the request and response.

Helper class that can be inserted into any WSGI application chain to get information about the request and response.

static print_generator(*app_iter*)

Prints the contents of a wrapper string iterator when iterated.

class heat.common.wsgi.**DefaultMethodController**

Bases: object

Controller that handles the OPTIONS request method.

This controller handles the OPTIONS request method and any of the HTTP methods that are not explicitly implemented by the application.

options(*req*, *allowed_methods*, **args*, ***kwargs*)

Return a response that includes the Allow header.

Return a response that includes the Allow header listing the methods that are implemented. A 204 status code is used for this response.

reject(*req*, *allowed_methods*, **args*, ***kwargs*)

Return a 405 method not allowed error.

As a convenience, the Allow header with the list of implemented methods is included in the response as well.

class heat.common.wsgi.**FilterFactory**(*conf*)

Bases: *AppFactory*

A Generic paste.deploy filter factory.

This requires heat.filter_factory to be set to a callable which returns a WSGI filter when invoked. The format is <module>:<callable> e.g.

> [filter:cache] paste.filter_factory = heat.common.wsgi:filter_factory heat.filter_factory = heat.api.middleware.cache:CacheFilter

The WSGI filter constructor must accept a WSGI app, a ConfigOpts object and a local config dict as its three arguments.

KEY = 'heat.filter_factory'

class heat.common.wsgi.**JSONRequestDeserializer**

Bases: object

default(*request*)

from_json(*datastring*)

has_body(*request*)

Returns whether a Webob.Request object will possess an entity body.

Parameters

request Webob.Request object

class heat.common.wsgi.**Middleware**(*application*)

Bases: object

Base WSGI middleware wrapper.

These classes require an application to be initialized that will be called next. By default the middleware will simply call its wrapped app, or you can override __call__ to customize its behavior.

process_request(*req*)

Called on each request.

If this returns None, the next application down the stack will be executed. If it returns a response then that response will be returned and execution will stop here.

process_response(*response*)

Do whatever youd like to the response.

class heat.common.wsgi.**Request**(*environ*, *charset=None*, *unicode_errors=None*, *decode_param_names=None*, ***kw*)

Bases: Request

Add some OpenStack API-specific logic to the base webob.Request.

best_match_content_type()

Determine the requested response content-type.

best_match_language()

Determines best available locale from the Accept-Language header.

Returns

the best language match or None if the Accept-Language header was not available in the request.

get_content_type(*allowed_content_types*)

Determine content type of the request body.

class heat.common.wsgi.**Resource**(*controller*, *deserializer*, *serializer=None*)

Bases: object

WSGI app that handles (de)serialization and controller dispatch.

Reads routing information supplied by RoutesMiddleware and calls the requested action method upon its deserializer, controller, and serializer. Those three objects may implement any of the

basic controller action methods (create, update, show, index, delete) along with any that may be specified in the api router. A default method may also be implemented to be used in place of any non-implemented actions. Deserializer methods must accept a request argument and return a dictionary. Controller methods must accept a request argument. Additionally, they must also accept keyword arguments that represent the keys returned by the Deserializer. They may raise a webob.exc exception or return a dict, which will be serialized by requested content type.

dispatch(*obj*, *action*, **args*, ***kwargs*)

Find action-specific method on self and call it.

get_action_args(*request_environment*)

Parse dictionary created by routes library.

class heat.common.wsgi.**Router**(*mapper*)

Bases: object

WSGI middleware that maps incoming requests to WSGI apps.

heat.common.wsgi.**debug_filter**(*app*, *conf*, ***local_conf*)

heat.common.wsgi.**is_json_content_type**(*request*)

heat.common.wsgi.**list_opts**()

heat.common.wsgi.**log_exception**(*err*, *exc_info*)

heat.common.wsgi.**paste_deploy_app**(*paste_config_file*, *app_name*, *conf*)

Load a WSGI app from a PasteDeploy configuration.

Use deploy.loadapp() to load the app from the PasteDeploy configuration, ensuring that the supplied ConfigOpts object is passed to the app and filter constructors.

Parameters

- **paste_config_file** a PasteDeploy config file
- **app_name** the name of the app/pipeline to load from the file
- **conf** a ConfigOpts object to supply to the app and its filters

Returns

the WSGI app

heat.common.wsgi.**setup_paste_factories**(*conf*)

Set up the generic paste app and filter factories.

Set things up so that:

paste.app_factory = heat.common.wsgi:app_factory

and

paste.filter_factory = heat.common.wsgi:filter_factory

work correctly while loading PasteDeploy configuration.

The app factories are constructed at runtime to allow us to pass a ConfigOpts object to the WSGI classes.

Parameters

conf a ConfigOpts object

heat.common.wsgi.**teardown_paste_factories**()

Reverse the effect of setup_paste_factories().

heat.common.wsgi.**translate_exception**(*exc*, *locale*)

Translates all translatable elements of the given exception.

Module contents

heat.db package

Submodules

heat.db.api module

Implementation of SQLAlchemy backend.

heat.db.api.**engine_get_all_locked_by_stack**(*context*, *stack_id*)

heat.db.api.**event_count_all_by_stack**(*context*, *stack_id*)

heat.db.api.**event_create**(*context*, *values*)

heat.db.api.**event_get_all_by_stack**(*context*, *stack_id*, *limit=None*, *marker=None*, *sort_keys=None*, *sort_dir=None*, *filters=None*)

heat.db.api.**event_get_all_by_tenant**(*context*, *limit=None*, *marker=None*, *sort_keys=None*, *sort_dir=None*, *filters=None*)

heat.db.api.**get_engine**()

heat.db.api.**persist_state_and_release_lock**(*context*, *stack_id*, *engine_id*, *values*)

heat.db.api.**purge_deleted**(*age*, *granularity='days'*, *project_id=None*, *batch_size=20*)

heat.db.api.**raw_template_create**(*context*, *values*)

heat.db.api.**raw_template_delete**(*context*, *template_id*)

heat.db.api.**raw_template_files_create**(*context*, *values*)

heat.db.api.**raw_template_files_get**(*context*, *files_id*)

heat.db.api.**raw_template_get**(*context*, *template_id*)

heat.db.api.**raw_template_update**(*context*, *template_id*, *values*)

heat.db.api.**reset_stack_status**(*context*, *stack_id*)

heat.db.api.**resource_attr_data_delete**(*context*, *resource_id*, *attr_id*)

heat.db.api.**resource_attr_id_set**(*context*, *resource_id*, *atomic_key*, *attr_id*)

heat.db.api.**resource_create**(*context*, *values*)

heat.db.api.**resource_create_replacement**(*context*, *existing_res_id*, *new_res_values*, *atomic_key*, *expected_engine_id=None*)

heat.db.api.**resource_data_delete**(*context*, *resource_id*, *key*)

heat.db.api.**resource_data_get**(*context*, *resource_id*, *key*)

Lookup value of resources data by key.

Decrypts resource data if necessary.

heat.db.api.**resource_data_get_all**(*context*, *resource_id*, *data=None*)

Looks up resource_data by resource.id.

If data is encrypted, this method will decrypt the results.

heat.db.api.**resource_data_get_by_key**(*context*, *resource_id*, *key*)

heat.db.api.**resource_data_set**(*context*, *resource_id*, *key*, *value*, *redact=False*)

Save resources key/value pair to database.

heat.db.api.**resource_delete**(*context*, *resource_id*)

heat.db.api.**resource_exchange_stacks**(*context*, *resource_id1*, *resource_id2*)

heat.db.api.**resource_get**(*context*, *resource_id*, *refresh=False*, *refresh_data=False*)

heat.db.api.**resource_get_all**(*context*)

heat.db.api.**resource_get_all_active_by_stack**(*context*, *stack_id*)

heat.db.api.**resource_get_all_by_physical_resource_id**(*context*, *physical_resource_id*)

heat.db.api.**resource_get_all_by_root_stack**(*context*, *stack_id*, *filters=None*, *stack_id_only=False*)

heat.db.api.**resource_get_all_by_stack**(*context*, *stack_id*, *filters=None*)

heat.db.api.**resource_get_by_name_and_stack**(*context*, *resource_name*, *stack_id*)

heat.db.api.**resource_get_by_physical_resource_id**(*context*, *physical_resource_id*)

heat.db.api.**resource_prop_data_create**(*context*, *values*)

heat.db.api.**resource_prop_data_create_or_update**(*context*, *values*, *rpd_id=None*)

heat.db.api.**resource_prop_data_get**(*context*, *resource_prop_data_id*)

heat.db.api.**resource_purge_deleted**(*context*, *stack_id*)

heat.db.api.**resource_snapshot_create**(*context*, *values*)

heat.db.api.**resource_snapshot_delete**(*context*, *resource_snapshot_id*)

heat.db.api.**resource_snapshot_get**(*context*, *resource_snapshot_id*)

heat.db.api.**resource_snapshot_get_all**(*context*, *snapshot_id*)

heat.db.api.**resource_snapshot_get_by_snapshot_and_rsrc**(*context*, *snapshot_id*, *resource_name*)

heat.db.api.**resource_snapshot_update**(*context*, *resource_snapshot_id*, *values*)

heat.db.api.**resource_update**(*context*, *resource_id*, *values*, *atomic_key*, *expected_engine_id=None*)

heat.db.api.**resource_update_and_save**(*context*, *resource_id*, *values*)

heat.db.api.**retry_on_db_error**(*func*)

heat.db.api.**service_create**(*context*, *values*)

heat.db.api.**service_delete**(*context*, *service_id*, *soft_delete=True*)

heat.db.api.**service_get**(*context*, *service_id*)

heat.db.api.**service_get_all**(*context*)

heat.db.api.**service_get_all_by_args**(*context*, *host*, *binary*, *hostname*)

heat.db.api.**service_update**(*context*, *service_id*, *values*)

heat.db.api.**snapshot_count_all_by_stack**(*context*, *stack_id*)

heat.db.api.**snapshot_create**(*context*, *values*)

heat.db.api.**snapshot_delete**(*context*, *snapshot_id*)

heat.db.api.**snapshot_get**(*context*, *snapshot_id*)

heat.db.api.**snapshot_get_all_by_stack**(*context*, *stack_id*)

heat.db.api.**snapshot_get_by_stack**(*context*, *snapshot_id*, *stack*)

heat.db.api.**snapshot_update**(*context*, *snapshot_id*, *values*)

heat.db.api.**software_config_count_all**(*context*)

heat.db.api.**software_config_create**(*context*, *values*)

heat.db.api.**software_config_delete**(*context*, *config_id*)

heat.db.api.**software_config_get**(*context*, *config_id*)

heat.db.api.**software_config_get_all**(*context*, *limit=None*, *marker=None*)

heat.db.api.**software_deployment_count_all**(*context*)

heat.db.api.**software_deployment_create**(*context*, *values*)

heat.db.api.**software_deployment_delete**(*context*, *deployment_id*)

heat.db.api.**software_deployment_get**(*context*, *deployment_id*)

heat.db.api.**software_deployment_get_all**(*context*, *server_id=None*)

heat.db.api.**software_deployment_update**(*context*, *deployment_id*, *values*)

heat.db.api.**stack_count_all**(*context*, *filters=None*, *show_deleted=False*, *show_nested=False*, *show_hidden=False*, *tags=None*, *tags_any=None*, *not_tags=None*, *not_tags_any=None*)

heat.db.api.**stack_count_total_resources**(*context*, *stack_id*)

heat.db.api.**stack_create**(*context*, *values*)

heat.db.api.**stack_delete**(*context*, *stack_id*)

heat.db.api.**stack_get**(*context*, *stack_id*, *show_deleted=False*, *eager_load=True*)

heat.db.api.**stack_get_all**(*context*, *limit=None*, *sort_keys=None*, *marker=None*, *sort_dir=None*, *filters=None*, *show_deleted=False*, *show_nested=False*, *show_hidden=False*, *tags=None*, *tags_any=None*, *not_tags=None*, *not_tags_any=None*, *eager_load=False*)

heat.db.api.**stack_get_all_by_owner_id**(*context*, *owner_id*)

heat.db.api.**stack_get_all_by_root_owner_id**(*context*, *owner_id*)

heat.db.api.**stack_get_by_name**(*context*, *stack_name*)

heat.db.api.**stack_get_by_name_and_owner_id**(*context*, *stack_name*, *owner_id*)

heat.db.api.**stack_get_root_id**(*context*, *stack_id*)

heat.db.api.**stack_get_status**(*context*, *stack_id*)

heat.db.api.**stack_lock_create**(*context*, *stack_id*, *engine_id*)

heat.db.api.**stack_lock_get_engine_id**(*context*, *stack_id*)

heat.db.api.**stack_lock_release**(*context*, *stack_id*, *engine_id*)

heat.db.api.**stack_lock_steal**(*context*, *stack_id*, *old_engine_id*, *new_engine_id*)

heat.db.api.**stack_tags_delete**(*context*, *stack_id*)

heat.db.api.**stack_tags_get**(*context*, *stack_id*)

heat.db.api.**stack_tags_set**(*context*, *stack_id*, *tags*)

heat.db.api.**stack_update**(*context*, *stack_id*, *values*, *exp_trvsl=None*)

heat.db.api.**sync_point_create**(*context*, *values*)

heat.db.api.**sync_point_delete_all_by_stack_and_traversal**(*context*, *stack_id*, *traversal_id*)

heat.db.api.**sync_point_get**(*context*, *entity_id*, *traversal_id*, *is_update*)

heat.db.api.**sync_point_update_input_data**(*context*, *entity_id*, *traversal_id*, *is_update*, *atomic_key*, *input_data*, *extra_data=None*)

heat.db.api.**user_creds_create**(*context*)

heat.db.api.**user_creds_delete**(*context*, *user_creds_id*)

heat.db.api.**user_creds_get**(*context*, *user_creds_id*)

heat.db.filters module

`heat.db.filters.exact_filter(query, model, filters)`

Applies exact match filtering to a query.

Returns the updated query. Modifies filters argument to remove filters consumed.

Parameters

- **query** query to apply filters to
- **model** model object the query applies to, for IN-style filtering
- **filters** dictionary of filters; values that are lists, tuples, sets, or frozensets cause an IN test to be performed, while exact matching (== operator) is used for other values

heat.db.migration module

`heat.db.migration.db_sync(version=None, engine=None)`

Migrate the database to *version* or the most recent version.

`heat.db.migration.db_version()`

Get database version.

heat.db.models module

SQLAlchemy models for heat data.

`class heat.db.models.Event(**kwargs)`

Bases: `Base`, *`HeatBase`*

Represents an event generated by the heat engine.

created_at

id

physical_resource_id

resource_action

resource_name

resource_properties

resource_status

property resource_status_reason

resource_type

rsrc_prop_data

rsrc_prop_data_id

stack

stack_id

updated_at

uuid

class heat.db.models.**HeatBase**

Bases: ModelBase, TimestampMixin

Base class for Heat Models.

class heat.db.models.**RawTemplate**(***kwargs*)

Bases: Base, *HeatBase*

Represents an unparsed template which should be in JSON format.

created_at

environment

files

files_id

id

template

updated_at

class heat.db.models.**RawTemplateFiles**(***kwargs*)

Bases: Base, *HeatBase*

Where template files json dicts are stored.

created_at

files

id

updated_at

class heat.db.models.**Resource**(***kwargs*)

Bases: Base, *HeatBase*, *StateAware*

Represents a resource created by the heat engine.

action

atomic_key

attr_data

attr_data_id

created_at

current_template_id

data

engine_id

id

name

needed_by

physical_resource_id

properties_data

properties_data_encrypted

replaced_by

replaces

requires

root_stack_id

rsrc_metadata

rsrc_prop_data

rsrc_prop_data_id

stack

stack_id

status

status_reason

updated_at

uuid

class heat.db.models.**ResourceData**(***kwargs*)

Bases: Base, *HeatBase*

Key/value store of arbitrary, resource-specific data.

created_at

decrypt_method

id

key

redact

resource_id

updated_at

value

class heat.db.models.**ResourcePropertiesData**(***kwargs*)

Bases: Base, *HeatBase*

Represents resource properties data, current or older

created_at

data

encrypted

id

updated_at

class heat.db.models.**ResourceSnapshot**(***kwargs*)

Bases: Base, *HeatBase*

created_at

data

id

resource_name

snapshot

snapshot_id

updated_at

class heat.db.models.**Service**(***kwargs*)

Bases: Base, *HeatBase*, *SoftDelete*

binary

created_at

deleted_at

engine_id

host

hostname

id

report_interval

topic

updated_at

class heat.db.models.**Snapshot**(***kwargs*)

Bases: Base, *HeatBase*

created_at

data

id

name

stack

stack_id

status

status_reason

tenant

updated_at

class heat.db.models.**SoftDelete**

Bases: object

deleted_at = Column(None, DateTime(), table=None)

class heat.db.models.**SoftwareConfig**(***kwargs*)

Bases: Base, *HeatBase*

Represents a software configuration resource.

Represents a software configuration resource to be applied to one or more servers.

config

created_at

group

id

name

tenant

updated_at

class heat.db.models.**SoftwareDeployment**(***kwargs*)

Bases: Base, *HeatBase*, *StateAware*

Represents a software deployment resource.

Represents applying a software configuration resource to a single server resource.

action

config

config_id

created_at

id

input_values

output_values

server_id

stack_user_project_id

status

status_reason

tenant

updated_at

class heat.db.models.**Stack**(***kwargs*)

Bases: Base, *HeatBase*, *SoftDelete*, *StateAware*

Represents a stack created by the heat engine.

action

backup

convergence

created_at

current_deps

current_traversal

deleted_at

disable_rollback

id

name

nested_depth

owner_id

parent_resource_name

prev_raw_template

prev_raw_template_id

raw_template

raw_template_id

stack_user_project_id

status

status_reason

tags

tenant

timeout

updated_at

user_creds_id

username

class heat.db.models.**StackLock**(***kwargs*)

Bases: Base, *HeatBase*

Store stack locks for deployments with multiple-engines.

created_at

engine_id

stack_id

updated_at

class heat.db.models.**StackTag**(***kwargs*)

Bases: Base, *HeatBase*

Key/value store of arbitrary stack tags.

created_at

id

stack_id

tag

updated_at

class heat.db.models.**StateAware**

Bases: object

action = Column('action', String(length=255), table=None)

status = Column('status', String(length=255), table=None)

status_reason = Column('status_reason', Text(), table=None)

class heat.db.models.**SyncPoint**(***kwargs*)

Bases: Base, *HeatBase*

Represents a syncpoint for a stack that is being worked on.

atomic_key

created_at

entity_id

extra_data

input_data

is_update

stack_id

traversal_id

updated_at

class heat.db.models.**UserCreds**(***kwargs*)

Bases: Base, *HeatBase*

Represents user credentials.

Also, mirrors the context handed in by wsgi.

auth_url

created_at

decrypt_method

id

password

region_name

stack

tenant

tenant_id

trust_id

trustor_user_id

updated_at

username

heat.db.types module

class heat.db.types.**Json**(**args: Any, **kwargs: Any*)

Bases: *LongText*

cache_ok: bool | None = True

Indicate if statements using this `ExternalType` are safe to cache.

The default value `None` will emit a warning and then not allow caching of a statement which includes this type. Set to `False` to disable statements using this type from being cached at all without a warning. When set to `True`, the objects class and selected elements from its state will be used as part of the cache key. For example, using a `TypeDecorator`:

```
class MyType(TypeDecorator):
    impl = String

    cache_ok = True

    def __init__(self, choices):
        self.choices = tuple(choices)
        self.internal_only = True
```

The cache key for the above type would be equivalent to:

```
>>> MyType(["a", "b", "c"])._static_cache_key
(<class '__main__.MyType'>, ('choices', ('a', 'b', 'c')))
```

The caching scheme will extract attributes from the type that correspond to the names of parameters in the `__init__()` method. Above, the choices attribute becomes part of the cache key but internal_only does not, because there is no parameter named internal_only.

The requirements for cacheable elements is that they are hashable and also that they indicate the same SQL rendered for expressions using this type every time for a given cache value.

To accommodate for datatypes that refer to unhashable structures such as dictionaries, sets and lists, these objects can be made cacheable by assigning hashable structures to the attributes whose names correspond with the names of the arguments. For example, a datatype which accepts a dictionary of lookup values may publish this as a sorted series of tuples. Given a previously un-cacheable type as:

```
class LookupType(UserDefinedType):
    """a custom type that accepts a dictionary as a parameter.

    this is the non-cacheable version, as "self.lookup" is not
    hashable.

    """

    def __init__(self, lookup):
        self.lookup = lookup

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"
```

(continues on next page)

(continued from previous page)

```
    def bind_processor(self, dialect): ...  # works with "self.lookup
↪" ...
```

Where lookup is a dictionary. The type will not be able to generate a cache key:

```
>>> type_ = LookupType({"a": 10, "b": 20})
>>> type_._static_cache_key
<stdin>:1: SAWarning: UserDefinedType LookupType({'a': 10, 'b': 20})␣
↪will not
produce a cache key because the ``cache_ok`` flag is not set to True.
Set this flag to True if this type object's state is safe to use
in a cache key, or False to disable this warning.
symbol('no_cache')
```

If we **did** set up such a cache key, it wouldnt be usable. We would get a tuple structure that contains a dictionary inside of it, which cannot itself be used as a key in a cache dictionary such as SQLAlchemys statement cache, since Python dictionaries arent hashable:

```
>>> # set cache_ok = True
>>> type_.cache_ok = True

>>> # this is the cache key it would generate
>>> key = type_._static_cache_key
>>> key
(<class '__main__.LookupType'>, ('lookup', {'a': 10, 'b': 20}))

>>> # however this key is not hashable, will fail when used with
>>> # SQLAlchemy statement cache
>>> some_cache = {key: "some sql value"}
Traceback (most recent call last): File "<stdin>", line 1,
in <module> TypeError: unhashable type: 'dict'
```

The type may be made cacheable by assigning a sorted tuple of tuples to the .lookup attribute:

```
class LookupType(UserDefinedType):
    """a custom type that accepts a dictionary as a parameter.

    The dictionary is stored both as itself in a private variable,
    and published in a public variable as a sorted tuple of tuples,
    which is hashable and will also return the same value for any
    two equivalent dictionaries.  Note it assumes the keys and
    values of the dictionary are themselves hashable.

    """

    cache_ok = True

    def __init__(self, lookup):
        self._lookup = lookup
```

(continues on next page)

(continued from previous page)

```
        # assume keys/values of "lookup" are hashable; otherwise
        # they would also need to be converted in some way here
        self.lookup = tuple((key, lookup[key]) for key in␣
↪sorted(lookup))

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect): ...  # works with "self._
↪lookup" ...
```

Where above, the cache key for `LookupType({"a": 10, "b": 20})` will be:

```
>>> LookupType({"a": 10, "b": 20})._static_cache_key
(<class '__main__.LookupType'>, ('lookup', (('a', 10), ('b', 20))))
```

Added in version 1.4.14: - added the `cache_ok` flag to allow some configurability of caching for `TypeDecorator` classes.

Added in version 1.4.28: - added the `ExternalType` mixin which generalizes the `cache_ok` flag to both the `TypeDecorator` and `UserDefinedType` classes.

See also

SQL Compilation Caching

process_bind_param(*value*, *dialect*)

Receive a bound parameter value to be converted.

Custom subclasses of `_types.TypeDecorator` should override this method to provide custom behaviors for incoming data values. This method is called at **statement execution time** and is passed the literal Python data value which is to be associated with a bound parameter in the statement.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

Parameters

- **value** Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **dialect** the `Dialect` in use.

See also

Augmenting Existing Types

`_types.TypeDecorator.process_result_value()`

process_result_value(*value*, *dialect*)

Receive a result-row column value to be converted.

Custom subclasses of `_types.TypeDecorator` should override this method to provide custom behaviors for data values being received in result rows coming from the database. This method is called at **result fetching time** and is passed the literal Python data value thats extracted from a database result row.

The operation could be anything desired to perform custom behavior, such as transforming or deserializing data.

Parameters

- **value** Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **dialect** the `Dialect` in use.

See also

Augmenting Existing Types

`_types.TypeDecorator.process_bind_param()`

class heat.db.types.**List**(**args: Any*, ***kwargs: Any*)

Bases: `TypeDecorator`

cache_ok: bool | None = True

Indicate if statements using this `ExternalType` are safe to cache.

The default value `None` will emit a warning and then not allow caching of a statement which includes this type. Set to `False` to disable statements using this type from being cached at all without a warning. When set to `True`, the objects class and selected elements from its state will be used as part of the cache key. For example, using a `TypeDecorator`:

```
class MyType(TypeDecorator):
    impl = String

    cache_ok = True

    def __init__(self, choices):
        self.choices = tuple(choices)
        self.internal_only = True
```

The cache key for the above type would be equivalent to:

```
>>> MyType(["a", "b", "c"])._static_cache_key
(<class '__main__.MyType'>, ('choices', ('a', 'b', 'c')))
```

The caching scheme will extract attributes from the type that correspond to the names of parameters in the `__init__()` method. Above, the choices attribute becomes part of the cache key but internal_only does not, because there is no parameter named internal_only.

The requirements for cacheable elements is that they are hashable and also that they indicate the same SQL rendered for expressions using this type every time for a given cache value.

To accommodate for datatypes that refer to unhashable structures such as dictionaries, sets and lists, these objects can be made cacheable by assigning hashable structures to the attributes whose names correspond with the names of the arguments. For example, a datatype which accepts a dictionary of lookup values may publish this as a sorted series of tuples. Given a previously un-cacheable type as:

```
class LookupType(UserDefinedType):
    """a custom type that accepts a dictionary as a parameter.

    this is the non-cacheable version, as "self.lookup" is not
    hashable.

    """

    def __init__(self, lookup):
        self.lookup = lookup

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect): ...  # works with "self.lookup
↪" ...
```

Where lookup is a dictionary. The type will not be able to generate a cache key:

```
>>> type_ = LookupType({"a": 10, "b": 20})
>>> type_._static_cache_key
<stdin>:1: SAWarning: UserDefinedType LookupType({'a': 10, 'b': 20})␣
↪will not
produce a cache key because the ``cache_ok`` flag is not set to True.
Set this flag to True if this type object's state is safe to use
in a cache key, or False to disable this warning.
symbol('no_cache')
```

If we **did** set up such a cache key, it wouldnt be usable. We would get a tuple structure that contains a dictionary inside of it, which cannot itself be used as a key in a cache dictionary such as SQLAlchemys statement cache, since Python dictionaries arent hashable:

```
>>> # set cache_ok = True
>>> type_.cache_ok = True

>>> # this is the cache key it would generate
>>> key = type_._static_cache_key
>>> key
(<class '__main__.LookupType'>, ('lookup', {'a': 10, 'b': 20}))

>>> # however this key is not hashable, will fail when used with
>>> # SQLAlchemy statement cache
>>> some_cache = {key: "some sql value"}
Traceback (most recent call last): File "<stdin>", line 1,
in <module> TypeError: unhashable type: 'dict'
```

The type may be made cacheable by assigning a sorted tuple of tuples to the .lookup attribute:

```
class LookupType(UserDefinedType):
    """a custom type that accepts a dictionary as a parameter.

    The dictionary is stored both as itself in a private variable,
    and published in a public variable as a sorted tuple of tuples,
    which is hashable and will also return the same value for any
    two equivalent dictionaries.  Note it assumes the keys and
    values of the dictionary are themselves hashable.

    """

    cache_ok = True

    def __init__(self, lookup):
        self._lookup = lookup

        # assume keys/values of "lookup" are hashable; otherwise
        # they would also need to be converted in some way here
        self.lookup = tuple((key, lookup[key]) for key in␣
↪sorted(lookup))

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect): ...  # works with "self._
↪lookup" ...
```

Where above, the cache key for `LookupType({"a": 10, "b": 20})` will be:

```
>>> LookupType({"a": 10, "b": 20})._static_cache_key
(<class '__main__.LookupType'>, ('lookup', (('a', 10), ('b', 20))))
```

Added in version 1.4.14: - added the `cache_ok` flag to allow some configurability of caching for `TypeDecorator` classes.

Added in version 1.4.28: - added the `ExternalType` mixin which generalizes the `cache_ok` flag to both the `TypeDecorator` and `UserDefinedType` classes.

See also

SQL Compilation Caching

impl

alias of `Text`

load_dialect_impl(*dialect*)

Return a `TypeEngine` object corresponding to a dialect.

This is an end-user override hook that can be used to provide differing types depending on the given dialect. It is used by the `TypeDecorator` implementation of `type_engine()` to

help determine what type should ultimately be returned for a given `TypeDecorator`.

By default returns `self.impl`.

process_bind_param(*value*, *dialect*)

Receive a bound parameter value to be converted.

Custom subclasses of `_types.TypeDecorator` should override this method to provide custom behaviors for incoming data values. This method is called at **statement execution time** and is passed the literal Python data value which is to be associated with a bound parameter in the statement.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

Parameters

- **value** Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **dialect** the `Dialect` in use.

See also

Augmenting Existing Types

`_types.TypeDecorator.process_result_value()`

process_result_value(*value*, *dialect*)

Receive a result-row column value to be converted.

Custom subclasses of `_types.TypeDecorator` should override this method to provide custom behaviors for data values being received in result rows coming from the database. This method is called at **result fetching time** and is passed the literal Python data value thats extracted from a database result row.

The operation could be anything desired to perform custom behavior, such as transforming or deserializing data.

Parameters

- **value** Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **dialect** the `Dialect` in use.

See also

Augmenting Existing Types

`_types.TypeDecorator.process_bind_param()`

class heat.db.types.**LongText**(**args: Any*, ***kwargs: Any*)

Bases: `TypeDecorator`

cache_ok: bool | None = True

Indicate if statements using this `ExternalType` are safe to cache.

The default value `None` will emit a warning and then not allow caching of a statement which includes this type. Set to `False` to disable statements using this type from being cached at all without a warning. When set to `True`, the objects class and selected elements from its state will be used as part of the cache key. For example, using a `TypeDecorator`:

```
class MyType(TypeDecorator):
    impl = String

    cache_ok = True

    def __init__(self, choices):
        self.choices = tuple(choices)
        self.internal_only = True
```

The cache key for the above type would be equivalent to:

```
>>> MyType(["a", "b", "c"])._static_cache_key
(<class '__main__.MyType'>, ('choices', ('a', 'b', 'c')))
```

The caching scheme will extract attributes from the type that correspond to the names of parameters in the `__init__()` method. Above, the choices attribute becomes part of the cache key but internal_only does not, because there is no parameter named internal_only.

The requirements for cacheable elements is that they are hashable and also that they indicate the same SQL rendered for expressions using this type every time for a given cache value.

To accommodate for datatypes that refer to unhashable structures such as dictionaries, sets and lists, these objects can be made cacheable by assigning hashable structures to the attributes whose names correspond with the names of the arguments. For example, a datatype which accepts a dictionary of lookup values may publish this as a sorted series of tuples. Given a previously un-cacheable type as:

```
class LookupType(UserDefinedType):
    """a custom type that accepts a dictionary as a parameter.

    this is the non-cacheable version, as "self.lookup" is not
    hashable.

    """

    def __init__(self, lookup):
        self.lookup = lookup

    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect): ...  # works with "self.lookup
↪" ...
```

Where lookup is a dictionary. The type will not be able to generate a cache key:

```
>>> type_ = LookupType({"a": 10, "b": 20})
>>> type_._static_cache_key
<stdin>:1: SAWarning: UserDefinedType LookupType({'a': 10, 'b': 20})␣
↪will not
produce a cache key because the ``cache_ok`` flag is not set to True.
Set this flag to True if this type object's state is safe to use
in a cache key, or False to disable this warning.
symbol('no_cache')
```

If we **did** set up such a cache key, it wouldnt be usable. We would get a tuple structure that contains a dictionary inside of it, which cannot itself be used as a key in a cache dictionary such as SQLAlchemys statement cache, since Python dictionaries arent hashable:

```
>>> # set cache_ok = True
>>> type_.cache_ok = True

>>> # this is the cache key it would generate
>>> key = type_._static_cache_key
>>> key
(<class '__main__.LookupType'>, ('lookup', {'a': 10, 'b': 20}))

>>> # however this key is not hashable, will fail when used with
>>> # SQLAlchemy statement cache
>>> some_cache = {key: "some sql value"}
Traceback (most recent call last): File "<stdin>", line 1,
in <module> TypeError: unhashable type: 'dict'
```

The type may be made cacheable by assigning a sorted tuple of tuples to the .lookup attribute:

```
class LookupType(UserDefinedType):
    """a custom type that accepts a dictionary as a parameter.

    The dictionary is stored both as itself in a private variable,
    and published in a public variable as a sorted tuple of tuples,
    which is hashable and will also return the same value for any
    two equivalent dictionaries.  Note it assumes the keys and
    values of the dictionary are themselves hashable.

    """

    cache_ok = True

    def __init__(self, lookup):
        self._lookup = lookup

        # assume keys/values of "lookup" are hashable; otherwise
        # they would also need to be converted in some way here
        self.lookup = tuple((key, lookup[key]) for key in␣
↪sorted(lookup))
```

(continues on next page)

(continued from previous page)

```
    def get_col_spec(self, **kw):
        return "VARCHAR(255)"

    def bind_processor(self, dialect): ...  # works with "self._
↪lookup" ...
```

Where above, the cache key for `LookupType({"a": 10, "b": 20})` will be:

```
>>> LookupType({"a": 10, "b": 20})._static_cache_key
(<class '__main__.LookupType'>, ('lookup', (('a', 10), ('b', 20))))
```

Added in version 1.4.14: - added the `cache_ok` flag to allow some configurability of caching for `TypeDecorator` classes.

Added in version 1.4.28: - added the `ExternalType` mixin which generalizes the `cache_ok` flag to both the `TypeDecorator` and `UserDefinedType` classes.

See also

SQL Compilation Caching

impl

alias of `Text`

load_dialect_impl(*dialect*)

Return a `TypeEngine` object corresponding to a dialect.

This is an end-user override hook that can be used to provide differing types depending on the given dialect. It is used by the `TypeDecorator` implementation of `type_engine()` to help determine what type should ultimately be returned for a given `TypeDecorator`.

By default returns `self.impl`.

heat.db.utils module

`heat.db.utils.`**retry_on_stale_data_error**(*func*)

Module contents

heat.engine package

Subpackages

heat.engine.cfn package

Submodules

heat.engine.cfn.functions module

class `heat.engine.cfn.functions.`**And**(*stack*, *fn_name*, *args*)

Bases: *And*

A function that acts as an AND operator on conditions.

Takes the form:

```
{ "Fn::And" : [ "<condition_1>", "<condition_2>", ... ] }
```

Returns true if all the specified conditions evaluate to true, or returns false if any one of the conditions evaluates to false. The minimum number of conditions that you can include is 2.

class heat.engine.cfn.functions.**Base64**(*stack*, *fn_name*, *args*)

Bases: *Function*

A placeholder function for converting to base64.

Takes the form:

```
{ "Fn::Base64" : "<string>" }
```

This function actually performs no conversion. It is included for the benefit of templates that convert UserData to Base64. Heat accepts UserData in plain text.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.cfn.functions.**Equals**(*stack*, *fn_name*, *args*)

Bases: *Equals*

A function for comparing whether two values are equal.

Takes the form:

```
{ "Fn::Equals" : [ "<value_1>", "<value_2>" ] }
```

The value can be any type that you want to compare. Returns true if the two values are equal or false if they arent.

class heat.engine.cfn.functions.**FindInMap**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for resolving keys in the template mappings.

Takes the form:

```
{ "Fn::FindInMap" : [ "mapping",
                      "key",
                      "value" ] }
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.cfn.functions.**GetAZs**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for retrieving the availability zones.

Takes the form:

```
{ "Fn::GetAZs" : "<region>" }
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.cfn.functions.**GetAtt**(*stack*, *fn_name*, *args*)

Bases: *GetAttThenSelect*

A function for resolving resource attributes.

Takes the form:

```
{ "Fn::GetAtt" : [ "<resource_name>",
                   "<attribute_name>" ] }
```

class heat.engine.cfn.functions.**If**(*stack*, *fn_name*, *raw_args*, *parse_func*, *template*)

Bases: *If*

A function to return corresponding value based on condition evaluation.

Takes the form:

```
{ "Fn::If" : [ "<condition_name>",
               "<value_if_true>",
               "<value_if_false>" ] }
```

The value_if_true to be returned if the specified condition evaluates to true, the value_if_false to be returned if the specified condition evaluates to false.

class heat.engine.cfn.functions.**Join**(*stack*, *fn_name*, *args*)

Bases: *Join*

A function for joining strings.

Takes the form:

```
{ "Fn::Join" : [ "<delim>", [ "<string_1>", "<string_2>", ... ] ] }
```

And resolves to:

```
"<string_1><delim><string_2><delim>..."
```

class heat.engine.cfn.functions.**MemberListToMap**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function to convert lists with enumerated keys and values to mapping.

Takes the form:

```
{ 'Fn::MemberListToMap' : [ 'Name',
                            'Value',
                            [ '.member.0.Name=<key_0>',
                              '.member.0.Value=<value_0>',
                              ... ] ] }
```

And resolves to:

```
{ "<key_0>" : "<value_0>", ... }
```

The first two arguments are the names of the key and value.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.cfn.functions.**Not**(*stack*, *fn_name*, *args*)

Bases: *Not*

A function that acts as a NOT operator on a condition.

Takes the form:

```
{ "Fn::Not" : [ "<condition>" ] }
```

Returns true for a condition that evaluates to false or returns false for a condition that evaluates to true.

class heat.engine.cfn.functions.**Or**(*stack*, *fn_name*, *args*)

Bases: *Or*

A function that acts as an OR operator on conditions.

Takes the form:

```
{ "Fn::Or" : [ "<condition_1>", "<condition_2>", ... ] }
```

Returns true if any one of the specified conditions evaluate to true, or returns false if all of the conditions evaluates to false. The minimum number of conditions that you can include is 2.

class heat.engine.cfn.functions.**ParamRef**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for resolving parameter references.

Takes the form:

```
{ "Ref" : "<param_name>" }
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

heat.engine.cfn.functions.**Ref**(*stack*, *fn_name*, *args*)

A function for resolving parameters or resource references.

Takes the form:

```
{ "Ref" : "<param_name>" }
```

or:

```
{ "Ref" : "<resource_name>" }
```

class heat.engine.cfn.functions.**Replace**(*stack*, *fn_name*, *args*)

Bases: *Replace*

A function for performing string substitutions.

Takes the form:

```
{ "Fn::Replace" : [
    { "<key_1>": "<value_1>", "<key_2>": "<value_2>", ... },
    "<key_1> <key_2>"
  ] }
```

And resolves to:

```
"<value_1> <value_2>"
```

When keys overlap in the template, longer matches are preferred. For keys of equal length, lexicographically smaller keys are preferred.

class heat.engine.cfn.functions.**ResourceFacade**(*stack*, *fn_name*, *args*)

Bases: *ResourceFacade*

A function for retrieving data in a parent provider template.

A function for obtaining data from the facade resource from within the corresponding provider template.

Takes the form:

```
{ "Fn::ResourceFacade": "<attribute_type>" }
```

where the valid attribute types are Metadata, DeletionPolicy and UpdatePolicy.

DELETION_POLICY = 'DeletionPolicy'

METADATA = 'Metadata'

UPDATE_POLICY = 'UpdatePolicy'

class heat.engine.cfn.functions.**Select**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for selecting an item from a list or map.

Takes the form (for a list lookup):

```
{ "Fn::Select" : [ "<index>", [ "<value_1>", "<value_2>", ... ] ] }
```

or (for a map lookup):

```
{ "Fn::Select" : [ "<index>", { "<key_1>": "<value_1>", ... } ] }
```

If the selected index is not found, this function resolves to an empty string.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.cfn.functions.**Split**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for splitting strings.

Takes the form:

```
{ "Fn::Split" : [ "<delim>", "<string_1><delim><string_2>..." ] }
```

And resolves to:

```
[ "<string_1>", "<string_2>", ... ]
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

heat.engine.cfn.parameters module

class heat.engine.cfn.parameters.**CfnParameters**(*stack_identifier*, *tmpl*, *user_params=None*, *param_defaults=None*)

Bases: *Parameters*

PARAM_REGION = 'AWS::Region'

PARAM_STACK_ID = 'AWS::StackId'

PARAM_STACK_NAME = 'AWS::StackName'

PSEUDO_PARAMETERS = ('AWS::StackId', 'AWS::StackName', 'AWS::Region')

heat.engine.cfn.template module

class heat.engine.cfn.template.**CfnTemplate**(*template*, **args*, ***kwargs*)

Bases: *CfnTemplateBase*

CONDITIONS = 'Conditions'

HOT_TO_CFN_RES_ATTRS = {'condition': 'Condition', 'deletion_policy': 'DeletionPolicy', 'depends_on': 'DependsOn', 'metadata': 'Metadata', 'properties': 'Properties', 'type': 'Type', 'update_policy': 'UpdatePolicy'}

OUTPUT_CONDITION = 'Condition'

OUTPUT_KEYS = ('Description', 'Value', 'Condition')

RES_CONDITION = 'Condition'

```
SECTIONS = ('AWSTemplateFormatVersion', 'HeatTemplateFormatVersion',
'Description', 'Mappings', 'Parameters', 'Resources', 'Outputs',
'Conditions')
```

```
SECTIONS_NO_DIRECT_ACCESS = {'AWSTemplateFormatVersion', 'Conditions',
'HeatTemplateFormatVersion', 'Parameters'}
```

```
condition_functions = {'Fn::And': <class
'heat.engine.cfn.functions.And'>, 'Fn::Equals': <class
'heat.engine.cfn.functions.Equals'>, 'Fn::FindInMap': <class
'heat.engine.cfn.functions.FindInMap'>, 'Fn::Not': <class
'heat.engine.cfn.functions.Not'>, 'Fn::Or': <class
'heat.engine.cfn.functions.Or'>, 'Ref': <class
'heat.engine.cfn.functions.ParamRef'>}
```

```
functions = {'Fn::Base64': <class 'heat.engine.cfn.functions.Base64'>,
'Fn::FindInMap': <class 'heat.engine.cfn.functions.FindInMap'>,
'Fn::GetAZs': <class 'heat.engine.cfn.functions.GetAZs'>, 'Fn::GetAtt':
<class 'heat.engine.cfn.functions.GetAtt'>, 'Fn::If': <class
'heat.engine.cfn.functions.If'>, 'Fn::Join': <class
'heat.engine.cfn.functions.Join'>, 'Fn::MemberListToMap': <class
'heat.engine.cfn.functions.MemberListToMap'>, 'Fn::Replace': <class
'heat.engine.cfn.functions.Replace'>, 'Fn::ResourceFacade': <class
'heat.engine.cfn.functions.ResourceFacade'>, 'Fn::Select': <class
'heat.engine.cfn.functions.Select'>, 'Fn::Split': <class
'heat.engine.cfn.functions.Split'>, 'Ref': <function Ref>}
```

class heat.engine.cfn.template.**CfnTemplateBase**(*template*, **args*, ***kwargs*)

Bases: *CommonTemplate*

The base implementation of cfn template.

```
ALTERNATE_VERSION = 'HeatTemplateFormatVersion'
```

```
DESCRIPTION = 'Description'
```

```
HOT_TO_CFN_OUTPUT_ATTRS = {'description': 'Description', 'value':
'Value'}
```

```
HOT_TO_CFN_RES_ATTRS = {'condition': 'Condition', 'deletion_policy':
'DeletionPolicy', 'depends_on': 'DependsOn', 'metadata': 'Metadata',
'properties': 'Properties', 'type': 'Type', 'update_policy':
'UpdatePolicy'}
```

```
MAPPINGS = 'Mappings'
```

```
OUTPUTS = 'Outputs'
```

```
OUTPUT_DESCRIPTION = 'Description'
```

```
OUTPUT_KEYS = ('Description', 'Value')
```

```
OUTPUT_VALUE = 'Value'
```

```
PARAMETERS = 'Parameters'
```

RESOURCES = 'Resources'

RES_DELETION_POLICY = 'DeletionPolicy'

RES_DEPENDS_ON = 'DependsOn'

RES_DESCRIPTION = 'Description'

RES_METADATA = 'Metadata'

RES_PROPERTIES = 'Properties'

RES_TYPE = 'Type'

RES_UPDATE_POLICY = 'UpdatePolicy'

SECTIONS = ('AWSTemplateFormatVersion', 'HeatTemplateFormatVersion', 'Description', 'Mappings', 'Parameters', 'Resources', 'Outputs')

SECTIONS_NO_DIRECT_ACCESS = {'AWSTemplateFormatVersion', 'HeatTemplateFormatVersion', 'Parameters'}

VERSION = 'AWSTemplateFormatVersion'

add_output(*definition*)

Add an output to the template.

The output is passed as a OutputDefinition object.

add_resource(*definition*, *name=None*)

Add a resource to the template.

The resource is passed as a ResourceDefinition object. If no name is specified, the name from the ResourceDefinition should be used.

deletion_policies = {'Delete': 'Delete', 'Retain': 'Retain', 'Snapshot': 'Snapshot'}

functions = {'Fn::Base64': <class 'heat.engine.cfn.functions.Base64'>, 'Fn::FindInMap': <class 'heat.engine.cfn.functions.FindInMap'>, 'Fn::GetAZs': <class 'heat.engine.cfn.functions.GetAZs'>, 'Fn::GetAtt': <class 'heat.engine.cfn.functions.GetAtt'>, 'Fn::Join': <class 'heat.engine.cfn.functions.Join'>, 'Fn::Select': <class 'heat.engine.cfn.functions.Select'>, 'Ref': <function Ref>}

get_section_name(*section*)

Get the name of a field within a resource or output definition.

Return the name of the given field (specified by the constants given in heat.engine.rsrc_defn and heat.engine.output) in the template format. This is used in error reporting to help users find the location of errors in the template.

Note that section here does not refer to a top-level section of the template (like parameters, resources, &c.) as it does everywhere else.

param_schemata(*param_defaults=None*)

Return a dict of parameters.Schema objects for the parameters.

parameters(*stack_identifier*, *user_params*, *param_defaults=None*)

Return a parameters.Parameters object for the stack.

resource_definitions(*stack*)

Return a dictionary of ResourceDefinition objects.

class heat.engine.cfn.template.**HeatTemplate**(*template*, **args*, ***kwargs*)

Bases: *CfnTemplateBase*

functions = {'Fn::Base64': <class 'heat.engine.cfn.functions.Base64'>, 'Fn::FindInMap': <class 'heat.engine.cfn.functions.FindInMap'>, 'Fn::GetAZs': <class 'heat.engine.cfn.functions.GetAZs'>, 'Fn::GetAtt': <class 'heat.engine.cfn.functions.GetAtt'>, 'Fn::Join': <class 'heat.engine.cfn.functions.Join'>, 'Fn::MemberListToMap': <class 'heat.engine.cfn.functions.MemberListToMap'>, 'Fn::Replace': <class 'heat.engine.cfn.functions.Replace'>, 'Fn::ResourceFacade': <class 'heat.engine.cfn.functions.ResourceFacade'>, 'Fn::Select': <class 'heat.engine.cfn.functions.Select'>, 'Fn::Split': <class 'heat.engine.cfn.functions.Split'>, 'Ref': <function Ref>}

Module contents

heat.engine.clients package

Subpackages

heat.engine.clients.os package

Subpackages

heat.engine.clients.os.keystone package

Submodules

heat.engine.clients.os.keystone.fake_keystoneclient module

A fake FakeKeystoneClient. This can be used during some runtime scenarios where you want to disable Heats internal Keystone dependencies entirely. One example is the TripleO Undercloud installer.

To use this class at runtime set to following heat.conf config setting:

keystone_backend = heat.engine.clients.os.keystone.fake_keystoneclient .FakeKeystoneClient

class heat.engine.clients.os.keystone.fake_keystoneclient.**FakeKeystoneClient**(*username='test_us pass-word='password', user_id='1234', ac-cess='4567', se-cret='8901', cre-den-tial_id='abcdxyz', auth_token='abcd con-text=None, stack_domain_id= client=None*)

Bases: object

create_ec2_keypair(*user_id*)

create_stack_domain_project(*stack_id*)

create_stack_domain_user(*username*, *project_id*, *password=None*)

create_stack_domain_user_keypair(*user_id*, *project_id*)

create_stack_user(*username*, *password*)

create_trust_context()

delete_ec2_keypair(*credential_id=None*, *user_id=None*, *access=None*)

delete_stack_domain_project(*project_id*)

delete_stack_domain_user(*user_id*, *project_id*)

delete_stack_domain_user_keypair(*user_id*, *project_id*, *credential_id*)

delete_stack_user(*user_id*)

delete_trust(*trust_id*)

disable_stack_domain_user(*user_id*, *project_id*)

disable_stack_user(*user_id*)

enable_stack_domain_user(*user_id*, *project_id*)

enable_stack_user(*user_id*)

get_ec2_keypair(*access*, *user_id*)

regenerate_trust_context()

server_keystone_endpoint_url(*fallback_endpoint*)

stack_domain_user_token(*user_id*, *project_id*, *password*)

heat.engine.clients.os.keystone.heat_keystoneclient module

Keystone Client functionality for use by resources.

class heat.engine.clients.os.keystone.heat_keystoneclient.**AccessKey**(*id*, *access*, *secret*)

Bases: tuple

access

Alias for field number 1

id

Alias for field number 0

secret

Alias for field number 2

class heat.engine.clients.os.keystone.heat_keystoneclient.**KeystoneClient**(*context*, *region_name=None*)

Bases: object

Keystone Auth Client.

Delay choosing the backend client module until the clients class needs to be initialized.

class heat.engine.clients.os.keystone.heat_keystoneclient.**KsClientWrapper**(*context*, *region_name*)

Bases: object

Wrap keystone client so we can encapsulate logic used in resources.

Note this is intended to be initialized from a resource on a per-session basis, so the session context is passed in on initialization Also note that an instance of this is created in each request context as part of a lazy-loaded cloud backend and it can be easily referenced in each resource as `self.keystone()`, so there should not be any need to directly instantiate instances of this class inside resources themselves.

property auth_region_name

property client

property context

create_ec2_keypair(*user_id=None*)

create_stack_domain_project(*stack_id*)

Create a project in the heat stack-user domain.

create_stack_domain_user(*username*, *project_id*, *password=None*)

Create a domain user defined as part of a stack.

The user is defined either via template or created internally by a resource. This user will be added to the heat_stack_user_role as defined in the config, and created in the specified project (which is expected to be in the stack_domain).

Returns the keystone ID of the resulting user.

create_stack_domain_user_keypair(*user_id*, *project_id*)

create_stack_user(*username*, *password*)

Create a user defined as part of a stack.

The user is defined either via template or created internally by a resource. This user will be added to the heat_stack_user_role as defined in the config.

Returns the keystone ID of the resulting user.

create_trust_context()

Create a trust using the trustor identity in the current context.

The trust is created with the trustee as the heat service user.

If the current context already contains a trust_id, we do nothing and return the current context.

Returns a context containing the new trust_id.

delete_ec2_keypair(*credential_id=None*, *access=None*, *user_id=None*)

Delete credential containing ec2 keypair.

delete_stack_domain_project(*project_id*)

delete_stack_domain_user(*user_id*, *project_id*)

delete_stack_domain_user_keypair(*user_id*, *project_id*, *credential_id*)

delete_stack_user(*user_id*)

delete_trust(*trust_id*)

Delete the specified trust.

disable_stack_domain_user(*user_id*, *project_id*)

disable_stack_user(*user_id*)

property domain_admin_auth

property domain_admin_client

enable_stack_domain_user(*user_id*, *project_id*)

enable_stack_user(*user_id*)

get_ec2_keypair(*credential_id=None*, *access=None*, *user_id=None*)

Get an ec2 keypair via v3/credentials, by id or access.

regenerate_trust_context()

Regenerate a trust using the trustor identity of current user_id.

The trust is created with the trustee as the heat service user.

Returns a context containing the new trust_id.

server_keystone_endpoint_url(*fallback_endpoint*)

property stack_domain

Domain scope data.

This is only used for checking for scoping data, not using the value.

property stack_domain_id

stack_domain_user_token(*user_id*, *project_id*, *password*)

Get a token for a stack domain user.

heat.engine.clients.os.keystone.heat_keystoneclient.**list_opts**()

heat.engine.clients.os.keystone.keystone_constraints module

class heat.engine.clients.os.keystone.keystone_constraints.**KeystoneBaseConstraint**

Bases: *BaseCustomConstraint*

entity = None

resource_client_name = 'keystone'

validate_with_client(*client*, *resource_id*)

class heat.engine.clients.os.keystone.keystone_constraints.**KeystoneDomainConstraint**

Bases: *KeystoneBaseConstraint*

entity = 'KeystoneDomain'

resource_getter_name = 'get_domain_id'

class heat.engine.clients.os.keystone.keystone_constraints.**KeystoneGroupConstraint**

Bases: *KeystoneBaseConstraint*

entity = 'KeystoneGroup'

resource_getter_name = 'get_group_id'

class heat.engine.clients.os.keystone.keystone_constraints.**KeystoneProjectConstraint**

Bases: *KeystoneBaseConstraint*

entity = 'KeystoneProject'

resource_getter_name = 'get_project_id'

class heat.engine.clients.os.keystone.keystone_constraints.**KeystoneRegionConstraint**

Bases: *KeystoneBaseConstraint*

entity = 'KeystoneRegion'

resource_getter_name = 'get_region_id'

class heat.engine.clients.os.keystone.keystone_constraints.**KeystoneRoleConstraint**

Bases: *KeystoneBaseConstraint*

entity = 'KeystoneRole'

resource_getter_name = 'get_role_id'

class heat.engine.clients.os.keystone.keystone_constraints.**KeystoneServiceConstraint**

Bases: *KeystoneBaseConstraint*

entity = 'KeystoneService'

expected_exceptions = (<class 'heat.common.exception.EntityNotFound'>, <class 'heat.common.exception.KeystoneServiceNameConflict'>)

resource_getter_name = 'get_service_id'

class heat.engine.clients.os.keystone.keystone_constraints.**KeystoneUserConstraint**

Bases: *KeystoneBaseConstraint*

entity = 'KeystoneUser'

resource_getter_name = 'get_user_id'

Module contents

class heat.engine.clients.os.keystone.**KeystoneClientPlugin**(*context*)

Bases: *ClientPlugin*

IDENTITY = 'identity'

exceptions_module = [<module 'keystoneauth1.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/keystoneauth1/exceptions/__init__.py'>, <module 'heat.common.exception' from '/home/zuul/src/opendev.org/openstack/heat/heat/common/exception.py'>]

get_domain_id(*domain*)

get_group_id(*group*, *domain=None*)

get_project_id(*project*, *domain=None*)

get_region_id(*region*)

get_role_id(*role*, *domain=None*)

get_service_id(*service*)

get_user_id(*user*, *domain=None*)

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

parse_entity_with_domain(*entity_with_domain*, *entity_type*)

Parse keystone entity user/role/project with domain.

entity_with_domain should be in entity{domain} format.

Returns a tuple of (entity, domain).

service_types = ['identity']

heat.engine.clients.os.neutron package

Submodules

heat.engine.clients.os.neutron.lbaas_constraints module

class heat.engine.clients.os.neutron.lbaas_constraints.**LBaasV2ProviderConstraint**

Bases: *ProviderConstraint*

service_type = 'LOADBALANCERV2'

class heat.engine.clients.os.neutron.lbaas_constraints.**ListenerConstraint**

Bases: *NeutronConstraint*

extension = 'lbaasv2'

resource_name = 'listener'

class heat.engine.clients.os.neutron.lbaas_constraints.**LoadbalancerConstraint**

Bases: *NeutronConstraint*

extension = 'lbaasv2'

resource_name = 'loadbalancer'

class heat.engine.clients.os.neutron.lbaas_constraints.**PoolConstraint**

Bases: *NeutronConstraint*

extension = 'lbaasv2'

resource_name = 'pool'

heat.engine.clients.os.neutron.neutron_constraints module

class heat.engine.clients.os.neutron.neutron_constraints.**AddressScopeConstraint**

Bases: *NeutronConstraint*

extension = 'address-scope'

resource_name = 'address_scope'

class heat.engine.clients.os.neutron.neutron_constraints.**FlowClassifierConstraint**

Bases: *NeutronExtConstraint*

extension = 'sfc'

resource_name = 'flow_classifier'

class heat.engine.clients.os.neutron.neutron_constraints.**NetworkConstraint**

Bases: *NeutronConstraint*

resource_name = 'network'

class heat.engine.clients.os.neutron.neutron_constraints.**NeutronConstraint**

Bases: *BaseCustomConstraint*

expected_exceptions = (<class 'neutronclient.common.exceptions.NeutronClientException'>, <class 'heat.common.exception.EntityNotFound'>)

extension = None

resource_name = None

validate_with_client(*client*, *value*)

class heat.engine.clients.os.neutron.neutron_constraints.**NeutronExtConstraint**

Bases: *NeutronConstraint*

validate_with_client(*client*, *value*)

class heat.engine.clients.os.neutron.neutron_constraints.**PortConstraint**

Bases: *NeutronConstraint*

resource_name = 'port'

class heat.engine.clients.os.neutron.neutron_constraints.**PortPairConstraint**

Bases: *NeutronExtConstraint*

extension = 'sfc'

resource_name = 'port_pair'

class heat.engine.clients.os.neutron.neutron_constraints.**PortPairGroupConstraint**

Bases: *NeutronExtConstraint*

extension = 'sfc'

resource_name = 'port_pair_group'

class heat.engine.clients.os.neutron.neutron_constraints.**ProviderConstraint**

Bases: *BaseCustomConstraint*

expected_exceptions = (<class 'heat.common.exception.StackValidationFailed'>,)

service_type = None

validate_with_client(*client*, *value*)

class heat.engine.clients.os.neutron.neutron_constraints.**QoSPolicyConstraint**

Bases: *NeutronConstraint*

extension = 'qos'

resource_name = 'policy'

class heat.engine.clients.os.neutron.neutron_constraints.**RouterConstraint**

Bases: *NeutronConstraint*

resource_name = 'router'

class heat.engine.clients.os.neutron.neutron_constraints.**SecurityGroupConstraint**

Bases: *NeutronConstraint*

resource_name = 'security_group'

class heat.engine.clients.os.neutron.neutron_constraints.**SubnetConstraint**

Bases: *NeutronConstraint*

resource_name = 'subnet'

class heat.engine.clients.os.neutron.neutron_constraints.**SubnetPoolConstraint**

Bases: *NeutronConstraint*

resource_name = 'subnetpool'

heat.engine.clients.os.neutron.taas_constraints module

class heat.engine.clients.os.neutron.taas_constraints.**TaaSProviderConstraint**

Bases: *ProviderConstraint*

service_type = 'TAPASASERVICE'

class heat.engine.clients.os.neutron.taas_constraints.**TapFlowConstraint**

Bases: *NeutronExtConstraint*

extension = 'taas'

resource_name = 'tap_flow'

class heat.engine.clients.os.neutron.taas_constraints.**TapServiceConstraint**

Bases: *NeutronExtConstraint*

extension = 'taas'

resource_name = 'tap_service'

Module contents

class heat.engine.clients.os.neutron.**NeutronClientPlugin**(**args*, ***kwargs*)

Bases: *ExtensionMixin*, *ClientPlugin*

NETWORK = 'network'

RES_TYPES = ('network', 'subnet', 'router', 'port', 'subnetpool', 'address_scope', 'security_group', 'policy', 'loadbalancer', 'listener', 'pool', 'l7policy')

RES_TYPE_ADDRESS_SCOPE = 'address_scope'

RES_TYPE_LB_L7POLICY = 'l7policy'

RES_TYPE_LB_LISTENER = 'listener'

RES_TYPE_LB_POOL = 'pool'

RES_TYPE_LOADBALANCER = 'loadbalancer'

RES_TYPE_NETWORK = 'network'

RES_TYPE_PORT = 'port'

RES_TYPE_QOS_POLICY = 'policy'

RES_TYPE_ROUTER = 'router'

RES_TYPE_SECURITY_GROUP = 'security_group'

RES_TYPE_SUBNET = 'subnet'

RES_TYPE_SUBNET_POOL = 'subnetpool'

check_ext_resource_status(*resource*, *resource_id*)

check_lb_status(*lb_id*)

create_ext_resource(*resource*, *props*)

Returns created ext resource record.

delete_ext_resource(*resource*, *resource_id*)

Deletes ext resource record and returns status.

exceptions_module = <module 'neutronclient.common.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/neutronclient/common/exceptions.py'>

find_resourceid_by_name_or_id(*resource*, *name_or_id*, *cmd_resource=None*)

Find a resource ID given either a name or an ID.

The *resource* argument should be one of the constants defined in RES_TYPES.

get_qos_policy_id(*policy*)

Returns the id of QoS policy.

Args: policy: ID or name of the policy.

get_secgroup_uuids(*security_groups*)

Returns a list of security group UUIDs.

Args: security_groups: List of security group names or UUIDs

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_invalid(*ex*)

is_no_unique(*ex*)

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

network_id_from_subnet_id(*subnet_id*)

resolve_ext_resource(*resource*, *name_or_id*)

Returns the id and validate neutron ext resource.

resolve_pool(*props*, *pool_key*, *pool_id_key*)

resolve_router(*props*, *router_key*, *router_id_key*)

service_types = ['network']

show_ext_resource(*resource*, *resource_id*)

Returns specific ext resource record.

update_ext_resource(*resource*, *prop_diff*, *resource_id*)

Returns updated ext resource record.

Submodules

heat.engine.clients.os.aodh module

class heat.engine.clients.os.aodh.**AodhClientPlugin**(*context*)

Bases: *ClientPlugin*

ALARMING = 'alarming'

V2 = '2'

default_version = '2'

exceptions_module = <module 'aodhclient.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/aodhclient/exceptions.py'>

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

service_types = ['alarming']

supported_versions = ['2']

heat.engine.clients.os.barbican module

class heat.engine.clients.os.barbican.**BarbicanClientPlugin**(*context*)

Bases: *ClientPlugin*

KEY_MANAGER = 'key-manager'

create_certificate(***props*)

create_generic_container(***props*)

create_rsa(***props*)

get_container_by_ref(*container_ref*)

get_secret_by_ref(*secret_ref*)

get_secret_payload_by_ref(*secret_ref*)

is_not_found(*ex*)

Returns True if the exception is a not-found.

service_types = ['key-manager']

class heat.engine.clients.os.barbican.**ContainerConstraint**

Bases: *BaseCustomConstraint*

expected_exceptions = (<class 'heat.common.exception.EntityNotFound'>,)

resource_client_name = 'barbican'

resource_getter_name = 'get_container_by_ref'

class heat.engine.clients.os.barbican.**SecretConstraint**

Bases: *BaseCustomConstraint*

expected_exceptions = (<class 'heat.common.exception.EntityNotFound'>,)

resource_client_name = 'barbican'

resource_getter_name = 'get_secret_by_ref'

heat.engine.clients.os.blazar module

class heat.engine.clients.os.blazar.**BlazarBaseConstraint**

Bases: *BaseCustomConstraint*

resource_client_name = 'blazar'

class heat.engine.clients.os.blazar.**BlazarClientPlugin**(*context*)

Bases: *ClientPlugin*

RESERVATION = 'reservation'

create_host(***args*)

create_lease(***args*)

get_host(*id*)

get_lease(*id*)

has_host()

is_not_found(*exc*)

Returns True if the exception is a not-found.

service_types = ['reservation']

class heat.engine.clients.os.blazar.**ReservationConstraint**

Bases: *BlazarBaseConstraint*

expected_exceptions = (<class 'heat.common.exception.EntityNotFound'>, <class 'blazarclient.exception.BlazarClientException'>)

resource_getter_name = 'get_lease'

heat.engine.clients.os.cinder module

class heat.engine.clients.os.cinder.**BaseCinderConstraint**

Bases: *BaseCustomConstraint*

resource_client_name = 'cinder'

class heat.engine.clients.os.cinder.**CinderClientPlugin**(**args*, ***kwargs*)

Bases: *MicroversionMixin*, *ExtensionMixin*, *ClientPlugin*

CINDER_API_VERSION = '3'

VOLUME_V3 = 'volumev3'

check_attach_volume_complete(*vol_id*)

check_detach_volume_complete(*vol_id*, *server_id=None*)

exceptions_module = <module 'cinderclient.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/cinderclient/exceptions.py'>

get_max_microversion()

get_qos_specs(*qos_specs*)

get_volume(*volume*)

get_volume_api_version()

Returns the most recent API version.

get_volume_backup(*backup*)

get_volume_snapshot(*snapshot*)

get_volume_type(*volume_type*)

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

is_version_supported(*version*)

max_microversion = None

service_types = ['volumev3']

class heat.engine.clients.os.cinder.**QoSSpecsConstraint**

Bases: *BaseCinderConstraint*

expected_exceptions = (<class 'cinderclient.exceptions.NotFound'>,)

resource_getter_name = 'get_qos_specs'

class heat.engine.clients.os.cinder.**VolumeBackupConstraint**

Bases: *BaseCinderConstraint*

resource_getter_name = 'get_volume_backup'

class heat.engine.clients.os.cinder.**VolumeConstraint**

Bases: *BaseCinderConstraint*

resource_getter_name = 'get_volume'

class heat.engine.clients.os.cinder.**VolumeSnapshotConstraint**

Bases: *BaseCinderConstraint*

resource_getter_name = 'get_volume_snapshot'

class heat.engine.clients.os.cinder.**VolumeTypeConstraint**

Bases: *BaseCinderConstraint*

resource_getter_name = 'get_volume_type'

heat.engine.clients.os.designate module

class heat.engine.clients.os.designate.**DesignateClientPlugin**(*context*)

Bases: *ClientPlugin*

DNS = 'dns'

exceptions_module = [<module 'designateclient.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/designateclient/exceptions.py'>]

get_zone_id(*zone_id_or_name*)

is_not_found(*ex*)

Returns True if the exception is a not-found.

service_types = ['dns']

class heat.engine.clients.os.designate.**DesignateZoneConstraint**

Bases: *BaseCustomConstraint*

resource_client_name = 'designate'

resource_getter_name = 'get_zone_id'

heat.engine.clients.os.glance module

class heat.engine.clients.os.glance.**GlanceClientPlugin**(*context*)

Bases: *ClientPlugin*

IMAGE = 'image'

V2 = '2'

default_version = '2'

exceptions_module = [<module 'heat.engine.clients.client_exception' from '/home/zuul/src/opendev.org/openstack/heat/heat/engine/clients/client_exception.py'>, <module 'glanceclient.exc' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/glanceclient/exc.py'>]

find_image_by_name_or_id(*image_identifier*)

Return the ID for the specified image name or identifier.

Parameters

image_identifier image name or a UUID-like identifier

Returns

the id of the requested :image_identifier:

get_image(*image_identifier*)

Return the image object for the specified image name/id.

Parameters

image_identifier image name

Returns

an image object with name/id :image_identifier:

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

service_types = ['image']

supported_versions = ['2']

class heat.engine.clients.os.glance.**ImageConstraint**

Bases: *BaseCustomConstraint*

expected_exceptions = (<class 'heat.engine.clients.client_exception.EntityMatchNotFound'>, <class 'heat.engine.clients.client_exception.EntityUniqueMatchNotFound'>)

resource_client_name = 'glance'

resource_getter_name = 'find_image_by_name_or_id'

heat.engine.clients.os.heat_plugin module

class heat.engine.clients.os.heat_plugin.**HeatClientPlugin**(*context*)

Bases: *ClientPlugin*

CLOUDFORMATION = 'cloudformation'

ORCHESTRATION = 'orchestration'

exceptions_module = <module 'heatclient.exc' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/heatclient/exc.py'>

get_cfn_metadata_server_url()

get_heat_cfn_url()

get_heat_url()

get_insecure_option()

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

service_types = ['orchestration', 'cloudformation']

heat.engine.clients.os.ironic module

class heat.engine.clients.os.ironic.**IronicClientPlugin**(*context*)

Bases: *MicroversionMixin*, *ClientPlugin*

BAREMETAL = 'baremetal'

IRONIC_API_VERSION = '1.104'

get_max_microversion()

get_node(*value*)

get_portgroup(*value*)

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

is_version_supported(*version*)

max_microversion = None

service_types = ['baremetal']

class heat.engine.clients.os.ironic.**NodeConstraint**

Bases: *BaseCustomConstraint*

resource_client_name = 'ironic'

resource_getter_name = 'get_node'

class heat.engine.clients.os.ironic.**PortGroupConstraint**

Bases: *BaseCustomConstraint*

resource_client_name = 'ironic'

resource_getter_name = 'get_portgroup'

heat.engine.clients.os.magnum module

class heat.engine.clients.os.magnum.**ClusterTemplateConstraint**

Bases: *BaseCustomConstraint*

resource_client_name = 'magnum'

resource_getter_name = 'get_cluster_template'

class heat.engine.clients.os.magnum.**MagnumClientPlugin**(*context*)

Bases: *ClientPlugin*

CONTAINER = 'container-infra'

get_cluster_template(*value*)

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

service_types = ['container-infra']

heat.engine.clients.os.manila module

class heat.engine.clients.os.manila.**ManilaClientPlugin**(*context*)

Bases: *ClientPlugin*

SHARE = 'sharev2'

exceptions_module = <module 'manilaclient.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/manilaclient/exceptions.py'>

get_security_service(*service_identity*)

get_share_network(*share_network_identity*)

get_share_snapshot(*snapshot_identity*)

get_share_type(*share_type_identity*)

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

service_types = ['sharev2']

class heat.engine.clients.os.manila.**ManilaShareBaseConstraint**

Bases: *BaseCustomConstraint*

expected_exceptions = (<class 'heat.common.exception.EntityNotFound'>, <class 'manilaclient.common.apiclient.exceptions.NoUniqueMatch'>)

resource_client_name = 'manila'

class heat.engine.clients.os.manila.**ManilaShareNetworkConstraint**

Bases: *ManilaShareBaseConstraint*

resource_getter_name = 'get_share_network'

class heat.engine.clients.os.manila.**ManilaShareSnapshotConstraint**

Bases: *ManilaShareBaseConstraint*

resource_getter_name = 'get_share_snapshot'

class heat.engine.clients.os.manila.**ManilaShareTypeConstraint**

Bases: *ManilaShareBaseConstraint*

resource_getter_name = 'get_share_type'

heat.engine.clients.os.mistral module

class heat.engine.clients.os.mistral.**MistralClientPlugin**(*context*)

Bases: *ClientPlugin*

WORKFLOW_V2 = 'workflowv2'

get_workflow_by_identifier(*workflow_identifier*)

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

service_types = ['workflowv2']

class heat.engine.clients.os.mistral.**WorkflowConstraint**

Bases: *BaseCustomConstraint*

expected_exceptions = (<class 'heat.common.exception.EntityNotFound'>,)

resource_client_name = 'mistral'

resource_getter_name = 'get_workflow_by_identifier'

heat.engine.clients.os.nova module

class heat.engine.clients.os.nova.**FlavorConstraint**

Bases: *NovaBaseConstraint*

expected_exceptions = (<class 'novaclient.exceptions.NotFound'>,)

resource_getter_name = 'find_flavor_by_name_or_id'

class heat.engine.clients.os.nova.**HostConstraint**

Bases: *NovaBaseConstraint*

expected_exceptions = (<class 'novaclient.exceptions.NotFound'>,)

resource_getter_name = 'get_host'

class heat.engine.clients.os.nova.**KeypairConstraint**

Bases: *NovaBaseConstraint*

resource_getter_name = 'get_keypair'

validate_with_client(*client*, *key_name*)

class heat.engine.clients.os.nova.**NovaBaseConstraint**

Bases: *BaseCustomConstraint*

resource_client_name = 'nova'

class heat.engine.clients.os.nova.**NovaClientPlugin**(*context*)

Bases: *MicroversionMixin*, *ClientPlugin*

COMPUTE = 'compute'

NOVA_API_VERSION = '2.1'

absolute_limits()

Return the absolute limits as a dictionary.

associate_floatingip(*server_id*, *floatingip_id*)

associate_floatingip_address(*server_id*, *fip_address*)

attach_volume(*server_id*, *volume_id*, *device*)

static build_ignition_data(*metadata*, *userdata*)

build_userdata(*metadata*, *userdata=None*, *instance_user=None*, *user_data_format='HEAT_CFNTOOLS'*)

Build multipart data blob for CloudInit and Ignition.

Data blob includes user-supplied Metadata, user data, and the required Heat in-instance configuration.

Parameters

- **resource** (*heat.engine.Resource*) the resource implementation
- **userdata** (*str or None*) user data string
- **instance_user** (*string*) the user to create on the server
- **user_data_format** (*string*) Format of user data to return

Returns

multipart mime as a string

check_delete_server_complete(*server_id*)

Wait for server to disappear from Nova.

check_detach_volume_complete(*server_id*, *attach_id*)

Check that nova server lost attachment.

This check is needed for immediate reattachment when updating: there might be some time between cinder marking volume as available and nova removing attachment from its own objects, so we check that nova already knows that the volume is detached.

check_interface_attach(*server_id*, *port_id*)

check_interface_detach(*server_id*, *port_id*)

check_rebuild(*server_id*)

Verify that a rebuilding server is rebuilt.

Raise error if it ends up in an ERROR state.

check_resize(*server_id*, *flavor*)

Verify that a resizing server is properly resized.

If thats the case, confirm the resize, if not raise an error.

check_verify_resize(*server_id*)

deferred_server_statuses = {'BUILD', 'HARD_REBOOT', 'PASSWORD', 'REBOOT', 'RESCUE', 'RESIZE', 'REVERT_RESIZE', 'SHUTOFF', 'SUSPENDED', 'VERIFY_RESIZE'}

detach_volume(*server_id*, *attach_id*)

dissociate_floatingip(*floatingip_id*)

dissociate_floatingip_address(*fip_address*)

exceptions_module = <module 'novaclient.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/novaclient/exceptions.py'>

fetch_server(*server_id*)

Fetch fresh server object from Nova.

Log warnings and return None for non-critical API errors. Use this method in various `check_*_complete` resource methods, where intermittent errors can be tolerated.

fetch_server_attr(*server_id*, *attr*)

find_flavor_by_name_or_id(*flavor*)

Find the specified flavor by name or id.

Parameters

flavor the name of the flavor to find

Returns

the id of :flavor:

get_console_urls(*server*)

Return dict-like structure of servers console urls.

The actual console url is lazily resolved on access.

get_flavor(*flavor_identifier*)

Get the flavor object for the specified flavor name or id.

Parameters

flavor_identifier the name or id of the flavor to find

Returns
a flavor object with name or id :flavor:

get_host(*hypervisor_hostname*)

Gets list of matching hypervisors by specified name.

Parameters
hypervisor_hostname the name of host to find

Returns
list of matching hypervisor hosts

Raises
nova client exceptions.NotFound

get_ip(*server*, *net_type*, *ip_version*)

Return the servers IP of the given type and version.

get_keypair(*key_name*)

Get the public key specified by :key_name:

Parameters
key_name the name of the key to look for

Returns
the keypair (name, public_key) for :key_name:

Raises
exception.EntityNotFound

get_max_microversion()

get_server(*server*)

Return fresh server object.

Substitutes Novas NotFound for Heats EntityNotFound, to be returned to user as HTTP error.

get_status(*server*)

Return the servers status.

Parameters
server server object

Returns
status as a string

interface_attach(*server_id*, *port_id=None*, *net_id=None*, *fip=None*, *security_groups=None*)

interface_detach(*server_id*, *port_id*)

is_bad_request(*ex*)

is_conflict(*ex*)

Returns True if the exception is a conflict.

static is_ignition_format(*userdata*)

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

is_unprocessable_entity(*ex*)

is_version_supported(*version*)

max_microversion = None

meta_serialize(*metadata*)

Serialize non-string metadata values before sending them to Nova.

meta_update(*server*, *metadata*)

Delete/Add the metadata in nova as needed.

rebuild(*server_id*, *image_id*, *password=None*, *preserve_ephemeral=False*, *meta=None*, *files=None*)

Rebuild the server and call check_rebuild to verify.

refresh_server(*server*)

Refresh servers attributes.

Also log warnings for non-critical API errors.

rename(*server*, *name*)

Update the name for a server.

resize(*server_id*, *flavor_id*)

Resize the server.

server_to_ipaddress(*server*)

Return the servers IP address, fetching it from Nova.

service_types = ['compute']

verify_resize(*server_id*)

class heat.engine.clients.os.nova.**ServerConstraint**

Bases: *NovaBaseConstraint*

resource_getter_name = 'get_server'

heat.engine.clients.os.octavia module

class heat.engine.clients.os.octavia.**AvailabilityZoneConstraint**

Bases: *OctaviaConstraint*

base_url = '/lbaas/availabilityzones'

class heat.engine.clients.os.octavia.**AvailabilityZoneProfileConstraint**

Bases: *OctaviaConstraint*

base_url = '/lbaas/availabilityzoneprofiles'

class heat.engine.clients.os.octavia.**FlavorConstraint**

Bases: *OctaviaConstraint*

base_url = '/lbaas/flavors'

class heat.engine.clients.os.octavia.**FlavorProfileConstraint**

Bases: *OctaviaConstraint*

base_url = '/lbaas/flavorprofiles'

class heat.engine.clients.os.octavia.**L7PolicyConstraint**

Bases: *OctaviaConstraint*

base_url = '/lbaas/l7policies'

class heat.engine.clients.os.octavia.**ListenerConstraint**

Bases: *OctaviaConstraint*

base_url = '/lbaas/listeners'

class heat.engine.clients.os.octavia.**LoadbalancerConstraint**

Bases: *OctaviaConstraint*

base_url = '/lbaas/loadbalancers'

class heat.engine.clients.os.octavia.**OctaviaClientPlugin**(*context*)

Bases: *ClientPlugin*

LOADBALANCER = 'load-balancer'

V2 = '2'

default_version = '2'

exceptions_module = <module 'octaviaclient.api.v2.octavia' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/octaviaclient/api/v2/octavia.py'>

get_availabilityzoneprofile(*value*)

get_flavor(*value*)

get_flavorprofile(*value*)

get_l7policy(*value*)

get_listener(*value*)

get_loadbalancer(*value*)

get_pool(*value*)

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

service_types = ['load-balancer']

supported_versions = ['2']

class heat.engine.clients.os.octavia.**OctaviaConstraint**

Bases: *BaseCustomConstraint*

base_url = None

expected_exceptions = (<class 'osc_lib.exceptions.NotFound'>, <class 'octaviaclient.api.exceptions.OctaviaClientException'>)

validate_with_client(*client*, *value*)

class heat.engine.clients.os.octavia.**PoolConstraint**

Bases: *OctaviaConstraint*

base_url = '/lbaas/pools'

heat.engine.clients.os.openstacksdk module

class heat.engine.clients.os.openstacksdk.**OpenStackSDKPlugin**(**args*, ***kwargs*)

Bases: *ExtensionMixin*, *ClientPlugin*

NETWORK = 'network'

exceptions_module = <module 'openstack.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/openstack/exceptions.py'>

find_network_ip(*value*)

find_network_port(*value*)

find_network_segment(*value*)

is_not_found(*ex*)

Returns True if the exception is a not-found.

service_types = ['network']

class heat.engine.clients.os.openstacksdk.**SegmentConstraint**

Bases: *BaseCustomConstraint*

expected_exceptions = (<class 'openstack.exceptions.NotFoundException'>, <class 'openstack.exceptions.DuplicateResource'>)

validate_with_client(*client*, *value*)

heat.engine.clients.os.swift module

class heat.engine.clients.os.swift.**SwiftClientPlugin**(*context*)

Bases: *ClientPlugin*

OBJECT_STORE = 'object-store'

exceptions_module = <module 'swiftclient.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/swiftclient/exceptions.py'>

get_files_from_container(*files_container*, *files_to_skip=None*)

Gets the file contents from a container.

Get the file contents from the container in a files map. A list of files to skip can also be specified and those would not be downloaded from swift.

get_signal_url(*container_name*, *obj_name*, *timeout=None*)

Turn on object versioning.

We can use a single TempURL for multiple signals and return a Swift TempURL.

get_temp_url(*container_name*, *obj_name*, *timeout=None*, *method='PUT'*)

Return a Swift TempURL.

is_client_exception(*ex*)

Returns True if the current exception comes from the client.

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

is_valid_temp_url_path(*path*)

Return True if path is a valid Swift TempURL path, False otherwise.

A Swift TempURL path must: - Be five parts, [, v1, account, container, object] - Be a v1 request - Have account, container, and object values - Have an object value with more than just /s

Parameters

path (*string*) The TempURL path

parse_last_modified(*lm*)

Parses the last-modified value.

For example, last-modified values from a swift object header. Returns the datetime.datetime of that value.

Parameters

lm (*string*) The last-modified value (or None)

Returns

An offset-naive UTC datetime of the value (or None)

service_types = ['object-store']

heat.engine.clients.os.trove module

class heat.engine.clients.os.trove.**FlavorConstraint**

Bases: *BaseCustomConstraint*

expected_exceptions = (<class 'troveclient.apiclient.exceptions.NotFound'>,)

resource_client_name = 'trove'

resource_getter_name = 'find_flavor_by_name_or_id'

class heat.engine.clients.os.trove.**TroveClientPlugin**(*context*)

Bases: *ClientPlugin*

DATABASE = 'database'

exceptions_module = <module 'troveclient.exceptions' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/troveclient/exceptions.py'>

find_flavor_by_name_or_id(*flavor*)

Find the specified flavor by name or id.

Parameters

flavor the name of the flavor to find

Returns

the id of :flavor:

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

service_types = ['database']

validate_datastore(*datastore_type*, *datastore_version*, *ds_type_key*, *ds_version_key*)

heat.engine.clients.os.vitrage module

class heat.engine.clients.os.vitrage.**VitrageClientPlugin**(*context*)

Bases: *ClientPlugin*

RCA = 'rca'

exceptions_module = None

service_types = ['rca']

heat.engine.clients.os.zaqar module

class heat.engine.clients.os.zaqar.**QueueConstraint**

Bases: *BaseCustomConstraint*

resource_client_name = 'zaqar'

resource_getter_name = 'get_queue'

class heat.engine.clients.os.zaqar.**ZaqarClientPlugin**(*context*)

Bases: *ClientPlugin*

DEFAULT_TTL = 3600

MESSAGING = 'messaging'

create_for_tenant(*tenant_id*, *token*)

create_from_signed_url(*project_id*, *paths*, *expires*, *methods*, *signature*)

exceptions_module = <module 'zaqarclient.transport.errors' from '/home/zuul/src/opendev.org/openstack/heat/.tox/pdf-docs/lib/python3.12/site-packages/zaqarclient/transport/errors.py'>

get_queue(*queue_name*)

is_not_found(*ex*)

Returns True if the exception is a not-found.

service_types = ['messaging']

class heat.engine.clients.os.zaqar.**ZaqarEventSink**(*target*, *ttl=None*)

Bases: object

consume(*context*, *event*)

heat.engine.clients.os.zun module

class heat.engine.clients.os.zun.**ZunClientPlugin**(*context*)

Bases: *ClientPlugin*

CONTAINER = 'container'

V1_12 = '1.12'

V1_18 = '1.18'

V1_36 = '1.36'

check_network_attach(*container_id*, *port_id*)

check_network_detach(*container_id*, *port_id*)

default_version = '1.12'

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

network_attach(*container_id*, *port_id=None*, *net_id=None*, *fip=None*, *security_groups=None*)

network_detach(*container_id*, *port_id*)

service_types = ['container']

supported_versions = ['1.12', '1.18', '1.36']

update_container(*container_id*, ***prop_diff*)

Module contents

class heat.engine.clients.os.**ExtensionMixin**(**args*, ***kwargs*)

Bases: object

has_extension(*alias*)

Check if specific extension is present.

Submodules

heat.engine.clients.client_exception module

exception heat.engine.clients.client_exception.**EntityMatchNotFound**(*entity=None*, *args=None*, ***kwargs*)

Bases: *HeatException*

msg_fmt = 'No %(entity)s matching %(args)s.'

exception heat.engine.clients.client_exception.**EntityUniqueMatchNotFound**(*entity=None*, *args=None*, ***kwargs*)

Bases: *EntityMatchNotFound*

msg_fmt = 'No %(entity)s unique match found for %(args)s.'

exception heat.engine.clients.client_exception.**InterfaceNotFound**(***kwargs*)

Bases: *HeatException*

msg_fmt = 'No network interface found for server %(id)s.'

heat.engine.clients.client_plugin module

class heat.engine.clients.client_plugin.**ClientPlugin**(*context*)

Bases: object

client(*version=None*)

property clients

property context

default_version = None

does_endpoint_exist(*service_type*, *service_name*)

exceptions_module = None

ignore_conflict_and_not_found(*ex*)

Raises the exception unless it is a conflict or not-found.

ignore_not_found(*ex*)

Raises the exception unless it is a not-found.

is_client_exception(*ex*)

Returns True if the current exception comes from the client.

is_conflict(*ex*)

Returns True if the exception is a conflict.

is_not_found(*ex*)

Returns True if the exception is a not-found.

is_over_limit(*ex*)

Returns True if the exception is an over-limit.

service_types = []

supported_versions = []

url_for(***kwargs*)

heat.engine.clients.client_plugin.**retry_if_connection_err**(*exception*)

heat.engine.clients.client_plugin.**retry_if_result_is_false**(*result*)

heat.engine.clients.default_client_plugin module

class heat.engine.clients.default_client_plugin.**DefaultClientPlugin**(*context*)

Bases: *ClientPlugin*

A ClientPlugin that has no client.

This is provided so that Resource can make use of the is_not_found() and is_conflict() methods even if the resource plugin has not specified a client plugin.

heat.engine.clients.microversion_mixin module

class heat.engine.clients.microversion_mixin.**MicroversionMixin**

Bases: object

Mixin For microversion support.

client(*version=None*)

abstract get_max_microversion()

abstract is_version_supported(*version*)

heat.engine.clients.progress module

Helper classes that are simple key-value storages meant to be passed between handle_* and check_*_complete, being mutated during subsequent check_*_complete calls.

Some of them impose restrictions on client plugin API, thus they are put in this client-plugin-agnostic module.

class heat.engine.clients.progress.**ContainerUpdateProgress**(*container_id*, *handler*, *complete=False*, *called=False*, *handler_extra=None*, *checker_extra=None*)

Bases: *UpdateProgressBase*

class heat.engine.clients.progress.**PoolDeleteProgress**(*task_complete=False*)

Bases: object

class heat.engine.clients.progress.**ServerCreateProgress**(*server_id*, *complete=False*)

Bases: object

class heat.engine.clients.progress.**ServerDeleteProgress**(*server_id*, *image_id=None*, *image_complete=True*)

Bases: object

class heat.engine.clients.progress.**ServerUpdateProgress**(*server_id*, *handler*, *complete=False*, *called=False*, *handler_extra=None*, *checker_extra=None*)

Bases: *UpdateProgressBase*

class heat.engine.clients.progress.**UpdateProgressBase**(*resource_id*, *handler*, *complete=False*, *called=False*, *handler_extra=None*, *checker_extra=None*)

Bases: object

Keeps track on particular server update task.

handler is a method of client plugin performing required update operation. Its first positional argument must be resource_id and this method must be resilent to intermittent failures, returning True if API was successfully called, False otherwise.

If result of API call is asynchronous, client plugin must have corresponding check_<handler> method. Its first positional argument must be resource_id and it must return True or False indicating completeness of the update operation.

For synchronous API calls, set complete attribute of this object to True.

[handler|checker]_extra arguments, if passed to constructor, should be dictionaries of

{args: tuple(), kwargs: dict()}

structure and contain parameters with which corresponding `handler` and `check_<handler>` methods of client plugin must be called. `args` is automatically prepended with `resource_id`. Missing `args` or `kwargs` are interpreted as empty tuple/dict respectively. Defaults are interpreted as both `args` and `kwargs` being empty.

class heat.engine.clients.progress.**VolumeAttachProgress**(*srv_id*, *vol_id*, *device*, *task_complete=False*)

Bases: object

class heat.engine.clients.progress.**VolumeBackupRestoreProgress**(*vol_id*, *backup_id*)

Bases: object

class heat.engine.clients.progress.**VolumeDeleteProgress**(*task_complete=False*)

Bases: object

class heat.engine.clients.progress.**VolumeDetachProgress**(*srv_id*, *vol_id*, *attach_id*, *task_complete=False*)

Bases: object

class heat.engine.clients.progress.**VolumeResizeProgress**(*task_complete=False*, *size=None*, *pre_check=False*)

Bases: object

class heat.engine.clients.progress.**VolumeUpdateAccessModeProgress**(*task_complete=False*, *read_only=None*)

Bases: object

Module contents

class heat.engine.clients.**ClientBackend**(*context*)

Bases: object

Class for delaying choosing the backend client module.

Delay choosing the backend client module until the clients class needs to be initialized.

heat.engine.clients.**Clients**

alias of *ClientBackend*

class heat.engine.clients.**OpenStackClients**(*context*)

Bases: object

Convenience class to create and cache client instances.

client(*name*, *version=None*)

client_plugin(*name*)

property context

heat.engine.clients.**has_client**(*name*)

heat.engine.clients.**initialise**()

heat.engine.clients.**list_opts**()

heat.engine.constraint package

Submodules

heat.engine.constraint.common_constraints module

class heat.engine.constraint.common_constraints.**CIDRConstraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*, *template=None*)

class heat.engine.constraint.common_constraints.**CRONExpressionConstraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*, *template=None*)

class heat.engine.constraint.common_constraints.**DNSDomainConstraint**

Bases: *DNSNameConstraint*

validate(*value*, *context*)

class heat.engine.constraint.common_constraints.**DNSNameConstraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*)

class heat.engine.constraint.common_constraints.**ExpirationConstraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*)

class heat.engine.constraint.common_constraints.**IPCIDRConstraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*, *template=None*)

class heat.engine.constraint.common_constraints.**IPConstraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*, *template=None*)

class heat.engine.constraint.common_constraints.**ISO8601Constraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*, *template=None*)

class heat.engine.constraint.common_constraints.**JsonStringConstraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*)

class heat.engine.constraint.common_constraints.**MACConstraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*, *template=None*)

class heat.engine.constraint.common_constraints.**RelativeDNSNameConstraint**

Bases: *DNSNameConstraint*

validate(*value*, *context*)

class heat.engine.constraint.common_constraints.**TestConstraintDelay**

Bases: *BaseCustomConstraint*

validate_with_client(*client*, *value*)

class heat.engine.constraint.common_constraints.**TimezoneConstraint**

Bases: *BaseCustomConstraint*

validate(*value*, *context*, *template=None*)

Module contents

heat.engine.hot package

Submodules

heat.engine.hot.functions module

class heat.engine.hot.functions.**And**(*stack*, *fn_name*, *args*)

Bases: *ConditionBoolean*

A function that acts as an AND operator on conditions.

Takes the form:

```
and:
  - <condition_1>
  - <condition_2>
  - ...
```

Returns true if all the specified conditions evaluate to true, or returns false if any one of the conditions evaluates to false. The minimum number of conditions that you can include is 2.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**ConditionBoolean**(*stack*, *fn_name*, *args*)

Bases: *Function*

Abstract parent class of boolean condition functions.

class heat.engine.hot.functions.**Contains**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for checking whether specific value is in sequence.

Takes the form:

```
contains:
  - <value>
  - <sequence>
```

The value can be any type that you want to check. Returns true if the specific value is in the sequence, otherwise returns false.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**Digest**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for performing digest operations.

Takes the form:

```
digest:
  - <algorithm>
  - <value>
```

Valid algorithms are the ones provided by natively by hashlib (md5, sha1, sha224, sha256, sha384, and sha512) or any one provided by OpenSSL.

digest(*algorithm*, *value*)

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

validate_usage(*args*)

class heat.engine.hot.functions.**Equals**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for comparing whether two values are equal.

Takes the form:

```
equals:
  - <value_1>
  - <value_2>
```

The value can be any type that you want to compare. Returns true if the two values are equal or false if they arent.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**Filter**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for filtering out values from lists.

Takes the form:

```
filter:
  - <values>
  - <list>
```

Returns a new list without the values.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**GetAtt**(*stack*, *fn_name*, *args*)

Bases: *GetAttThenSelect*

A function for resolving resource attributes.

Takes the form:

```
get_attr:
  - <resource_name>
  - <attribute_name>
  - <path1>
  - ...
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**GetAttAllAttributes**(*stack*, *fn_name*, *args*)

Bases: *GetAtt*

A function for resolving resource attributes.

Takes the form:

```
get_attr:
  - <resource_name>
  - <attributes_name>
  - <path1>
  - ...
```

where <attributes_name> and <path1>, are optional arguments. If there is no <attributes_name>, result will be dict of all resources attributes. Else function returns resolved resources attribute.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**GetAttThenSelect**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for resolving resource attributes.

Takes the form:

```
get_attr:
  - <resource_name>
  - <attribute_name>
  - <path1>
  - ...
```

all_dep_attrs()

Return resource, attribute name pairs of all attributes referenced.

Return an iterator over the resource name, attribute name tuples of all attributes that this function references.

The special value heat.engine.attributes.ALL_ATTRIBUTES may be used to indicate that all attributes of the resource are required.

By default this calls the dep_attrs() method, but subclasses can override to provide a more efficient implementation.

dep_attrs(*resource_name*)

Return the attributes of the specified resource that are referenced.

Return an iterator over any attributes of the specified resource that this function references.

The special value heat.engine.attributes.ALL_ATTRIBUTES may be used to indicate that all attributes of the resource are required.

dependencies(*path*)

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

validate()

Validate arguments without resolving the function.

Function subclasses must override this method to validate their args.

class heat.engine.hot.functions.**GetFile**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for including a file inline.

Takes the form:

```
get_file: <file_key>
```

And resolves to the content stored in the files dictionary under the given key.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**GetParam**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for resolving parameter references.

Takes the form:

```
get_param: <param_name>
```

or:

```
get_param:
  - <param_name>
  - <path1>
  - ...
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**GetResource**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for resolving resource references.

Takes the form:

```
get_resource: <resource_name>
```

dependencies(*path*)

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**If**(*stack*, *fn_name*, *raw_args*, *parse_func*, *template*)

Bases: *Macro*

A function to return corresponding value based on condition evaluation.

Takes the form:

```
if:
  - <condition_name>
  - <value_if_true>
  - <value_if_false>
```

The value_if_true to be returned if the specified condition evaluates to true, the value_if_false to be returned if the specified condition evaluates to false.

parse_args(*parse_func*)

Parse the macro using the supplied parsing function.

Macro subclasses should override this method to control parsing of the arguments.

class heat.engine.hot.functions.**IfNullable**(*stack*, *fn_name*, *raw_args*, *parse_func*, *template*)

Bases: *If*

A function to return corresponding value based on condition evaluation.

Takes the form:

```
if:
  - <condition_name>
  - <value_if_true>
  - <value_if_false>
```

The value_if_true to be returned if the specified condition evaluates to true, the value_if_false to be returned if the specified condition evaluates to false.

If the value_if_false is omitted and the condition is false, the enclosing item (list item, dictionary key/value pair, property definition) will be treated as if it were not mentioned in the template:

```
if:
  - <condition_name>
  - <value_if_true>
```

class heat.engine.hot.functions.**Join**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for joining strings.

Takes the form:

```
list_join:
  - <delim>
  - - <string_1>
    - <string_2>
    - ...
```

And resolves to:

```
"<string_1><delim><string_2><delim>..."
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**JoinMultiple**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for joining one or more lists of strings.

Takes the form:

```
list_join:
  - <delim>
  - - <string_1>
    - <string_2>
    - ...
  - - ...
```

And resolves to:

```
"<string_1><delim><string_2><delim>..."
```

Optionally multiple lists may be specified, which will also be joined.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**ListConcat**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for extending lists.

Takes the form:

```
list_concat:
  - [<value 1>, <value 2>]
  - [<value 3>, <value 4>]
```

And resolves to:

```
[<value 1>, <value 2>, <value 3>, <value 4>]
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**ListConcatUnique**(*stack*, *fn_name*, *args*)

Bases: *ListConcat*

A function for extending lists with unique items.

list_concat_unique is identical to the list_concat function, only contains unique items in retuning list.

class heat.engine.hot.functions.**MakeURL**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for performing substitutions on maps.

Takes the form:

```
make_url:
  scheme: <protocol>
  username: <username>
  password: <password>
  host: <hostname or IP>
  port: <port>
  path: <path>
  query:
    <key1>: <value1>
  fragment: <fragment>
```

And resolves to a correctly-escaped URL constructed from the various components.

FRAGMENT = 'fragment'

HOST = 'host'

PASSWORD = 'password'

PATH = 'path'

PORT = 'port'

QUERY = 'query'

SCHEME = 'scheme'

USERNAME = 'username'

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

validate()

Validate arguments without resolving the function.

Function subclasses must override this method to validate their args.

class heat.engine.hot.functions.**MapMerge**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for merging maps.

Takes the form:

```
map_merge:
  - <k1>: <v1>
    <k2>: <v2>
  - <k1>: <v3>
```

And resolves to:

```
{"<k1>": "<v3>", "<k2>": "<v2>"}
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**MapReplace**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for performing substitutions on maps.

Takes the form:

```
map_replace:
  - <k1>: <v1>
    <k2>: <v2>
```

(continues on next page)

(continued from previous page)

```
  - keys:
      <k1>: <K1>
    values:
      <v2>: <V2>
```

And resolves to:

```
{"<K1>": "<v1>", "<k2>": "<V2>"}
```

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**Not**(*stack*, *fn_name*, *args*)

Bases: *ConditionBoolean*

A function that acts as a NOT operator on a condition.

Takes the form:

```
not: <condition>
```

Returns true for a condition that evaluates to false or returns false for a condition that evaluates to true.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**Or**(*stack*, *fn_name*, *args*)

Bases: *ConditionBoolean*

A function that acts as an OR operator on conditions.

Takes the form:

```
or:
  - <condition_1>
  - <condition_2>
  - ...
```

Returns true if any one of the specified conditions evaluate to true, or returns false if all of the conditions evaluates to false. The minimum number of conditions that you can include is 2.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**Removed**(*stack*, *fn_name*, *args*)

Bases: *Function*

This function existed in previous versions of HOT, but has been removed.

Check the HOT guide for an equivalent native function.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

validate()

Validate arguments without resolving the function.

Function subclasses must override this method to validate their args.

class heat.engine.hot.functions.**Repeat**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for iterating over a list of items.

Takes the form:

```
repeat:
    template:
        <body>
    for_each:
        <var>: <list>
```

The result is a new list of the same size as <list>, where each element is a copy of <body> with any occurrences of <var> replaced with the corresponding item of <list>.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

validate()

Validate arguments without resolving the function.

Function subclasses must override this method to validate their args.

class heat.engine.hot.functions.**RepeatWithMap**(*stack*, *fn_name*, *args*)

Bases: *Repeat*

A function for iterating over a list of items or a dict of keys.

Takes the form:

```
repeat:
    template:
        <body>
    for_each:
        <var>: <list> or <dict>
```

The result is a new list of the same size as <list> or <dict>, where each element is a copy of <body> with any occurrences of <var> replaced with the corresponding item of <list> or key of <dict>.

class heat.engine.hot.functions.**RepeatWithNestedLoop**(*stack*, *fn_name*, *args*)

Bases: *RepeatWithMap*

A function for iterating over a list of items or a dict of keys.

Takes the form:

```
repeat:
    template:
        <body>
    for_each:
        <var>: <list> or <dict>
```

The result is a new list of the same size as <list> or <dict>, where each element is a copy of <body> with any occurrences of <var> replaced with the corresponding item of <list> or key of <dict>.

This function also allows to specify permutations to decide whether to iterate nested the over all the permutations of the elements in the given lists.

Takes the form:

```
repeat:
  template:
    var: %var%
    bar: %bar%
  for_each:
    %var%: <list1>
    %bar%: <list2>
  permutations: false
```

If permutations is not specified, we set the default value to true to compatible with before behavior. The args have to be lists instead of dicts if permutations is False because keys in a dict are unordered, and the list args all have to be of the same length.

class heat.engine.hot.functions.**Replace**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for performing string substitutions.

Takes the form:

```
str_replace:
  template: <key_1> <key_2>
  params:
    <key_1>: <value_1>
    <key_2>: <value_2>
    ...
```

And resolves to:

```
"<value_1> <value_2>"
```

When keys overlap in the template, longer matches are preferred. For keys of equal length, lexicographically smaller keys are preferred.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**ReplaceJson**(*stack*, *fn_name*, *args*)

Bases: *Replace*

A function for performing string substitutions.

Takes the form:

```
str_replace:
  template: <key_1> <key_2>
  params:
    <key_1>: <value_1>
    <key_2>: <value_2>
    ...
```

And resolves to:

```
"<value_1> <value_2>"
```

When keys overlap in the template, longer matches are preferred. For keys of equal length, lexicographically smaller keys are preferred.

Non-string param values (e.g maps or lists) are serialized as JSON before being substituted in.

class heat.engine.hot.functions.**ReplaceJsonStrict**(*stack*, *fn_name*, *args*)

Bases: *ReplaceJson*

A function for performing string substitutions.

str_replace_strict is identical to the str_replace function, only a ValueError is raised if any of the params are not present in the template.

class heat.engine.hot.functions.**ReplaceJsonVeryStrict**(*stack*, *fn_name*, *args*)

Bases: *ReplaceJsonStrict*

A function for performing string substitutions.

str_replace_vstrict is identical to the str_replace_strict function, only a ValueError is raised if any of the params are None or empty.

class heat.engine.hot.functions.**ResourceFacade**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for retrieving data in a parent provider template.

A function for obtaining data from the facade resource from within the corresponding provider template.

Takes the form:

```
resource_facade: <attribute_type>
```

where the valid attribute types are metadata, deletion_policy and update_policy.

DELETION_POLICY = 'deletion_policy'

METADATA = 'metadata'

UPDATE_POLICY = 'update_policy'

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**StrSplit**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for splitting delimited strings into a list.

Optionally extracting a specific list member by index.

Takes the form:

```
str_split:
  - <delimiter>
  - <string>
  - <index>
```

If <index> is specified, the specified list item will be returned otherwise, the whole list is returned, similar to get_attr with path based attributes accessing lists.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.hot.functions.**Yaql**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for executing a yaql expression.

Takes the form:

```
yaql:
    expression:
        <body>
    data:
        <var>: <list>
```

Evaluates expression <body> on the given data.

classmethod get_yaql_parser()

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

validate()

Validate arguments without resolving the function.

Function subclasses must override this method to validate their args.

heat.engine.hot.functions.**list_opts()**

heat.engine.hot.parameters module

class heat.engine.hot.parameters.**HOTParamSchema**(*data_type*, *description=None*, *default=None*, *schema=None*, *constraints=None*, *hidden=False*, *label=None*, *immutable=False*, *tags=None*)

Bases: *Schema*

HOT parameter schema.

BOOLEAN = 'boolean'

CONSTRAINTS = 'constraints'

DEFAULT = 'default'

DESCRIPTION = 'description'

HIDDEN = 'hidden'

IMMUTABLE = 'immutable'

KEYS = ('type', 'description', 'default', 'schema', 'constraints', 'hidden', 'label', 'immutable')

LABEL = 'label'

LIST = 'comma_delimited_list'

MAP = 'json'

NUMBER = 'number'

PARAMETER_KEYS = ('type', 'description', 'default', 'schema', 'constraints', 'hidden', 'label', 'immutable')

SCHEMA = 'schema'

STRING = 'string'

TYPE = 'type'

TYPES = ('string', 'number', 'comma_delimited_list', 'json', 'boolean')

classmethod from_dict(*param_name*, *schema_dict*)

Return a Parameter Schema object from a legacy schema dictionary.

Parameters

param_name (*str*) name of the parameter owning the schema; used for more verbose logging

class heat.engine.hot.parameters.**HOTParamSchema20170224**(*data_type*, *description=None*, *default=None*, *schema=None*, *constraints=None*, *hidden=False*, *label=None*, *immutable=False*, *tags=None*)

Bases: *HOTParamSchema*

class heat.engine.hot.parameters.**HOTParamSchema20180302**(*data_type*, *description=None*, *default=None*, *schema=None*, *constraints=None*, *hidden=False*, *label=None*, *immutable=False*, *tags=None*)

Bases: *HOTParamSchema20170224*

KEYS = ('type', 'description', 'default', 'schema', 'constraints', 'hidden', 'label', 'immutable', 'tags')

KEYS_20180302 = ('tags',)

PARAMETER_KEYS = ('type', 'description', 'default', 'schema', 'constraints', 'hidden', 'label', 'immutable', 'tags')

TAGS = 'tags'

classmethod from_dict(*param_name*, *schema_dict*)

Return a Parameter Schema object from a legacy schema dictionary.

Parameters

param_name (*str*) name of the parameter owning the schema; used for more verbose logging

class heat.engine.hot.parameters.**HOTParameters**(*stack_identifier*, *tmpl*, *user_params=None*, *param_defaults=None*)

Bases: *Parameters*

PARAM_PROJECT_ID = 'OS::project_id'

PARAM_REGION = 'OS::region'

PARAM_STACK_ID = 'OS::stack_id'

PARAM_STACK_NAME = 'OS::stack_name'

PSEUDO_PARAMETERS = ('OS::stack_id', 'OS::stack_name', 'OS::region', 'OS::project_id')

set_stack_id(*stack_identifier*)

Set the StackId pseudo parameter value.

heat.engine.hot.template module

class heat.engine.hot.template.**HOTemplate20130523**(*template*, **args*, ***kwargs*)

Bases: *CommonTemplate*

A Heat Orchestration Template format stack template.

DESCRIPTION = 'description'

MAPPINGS = '__undefined__'

OUTPUTS = 'outputs'

OUTPUT_DESCRIPTION = 'description'

OUTPUT_KEYS = ('description', 'value')

OUTPUT_VALUE = 'value'

PARAMETERS = 'parameters'

PARAMETER_GROUPS = 'parameter_groups'

RESOURCES = 'resources'

RES_DELETION_POLICY = 'deletion_policy'

RES_DEPENDS_ON = 'depends_on'

RES_DESCRIPTION = 'description'

RES_METADATA = 'metadata'

RES_PROPERTIES = 'properties'

RES_TYPE = 'type'

RES_UPDATE_POLICY = 'update_policy'

SECTIONS = ('heat_template_version', 'description', 'parameter_groups', 'parameters', 'resources', 'outputs', '__undefined__')

SECTIONS_NO_DIRECT_ACCESS = {'heat_template_version', 'parameters'}

VERSION = 'heat_template_version'

add_output(*definition*)

Add an output to the template.

The output is passed as a OutputDefinition object.

add_resource(*definition*, *name=None*)

Add a resource to the template.

The resource is passed as a ResourceDefinition object. If no name is specified, the name from the ResourceDefinition should be used.

deletion_policies = {'Delete': 'Delete', 'Retain': 'Retain', 'Snapshot': 'Snapshot'}

```
functions = {'Fn::Base64': <class 'heat.engine.cfn.functions.Base64'>,
'Fn::GetAZs': <class 'heat.engine.cfn.functions.GetAZs'>, 'Fn::Join':
<class 'heat.engine.cfn.functions.Join'>, 'Fn::MemberListToMap': <class
'heat.engine.cfn.functions.MemberListToMap'>, 'Fn::Replace': <class
'heat.engine.cfn.functions.Replace'>, 'Fn::ResourceFacade': <class
'heat.engine.cfn.functions.ResourceFacade'>, 'Fn::Select': <class
'heat.engine.cfn.functions.Select'>, 'Fn::Split': <class
'heat.engine.cfn.functions.Split'>, 'Ref': <function Ref>, 'get_attr':
<class 'heat.engine.hot.functions.GetAttThenSelect'>, 'get_file': <class
'heat.engine.hot.functions.GetFile'>, 'get_param': <class
'heat.engine.hot.functions.GetParam'>, 'get_resource': <class
'heat.engine.hot.functions.GetResource'>, 'list_join': <class
'heat.engine.hot.functions.Join'>, 'resource_facade': <class
'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class
'heat.engine.hot.functions.Replace'>}
```

get_section_name(*section*)

Get the name of a field within a resource or output definition.

Return the name of the given field (specified by the constants given in heat.engine.rsrc_defn and heat.engine.output) in the template format. This is used in error reporting to help users find the location of errors in the template.

Note that section here does not refer to a top-level section of the template (like parameters, resources, &c.) as it does everywhere else.

param_schema_class

alias of *HOTParamSchema*

param_schemata(*param_defaults=None*)

Return a dict of parameters.Schema objects for the parameters.

parameters(*stack_identifier*, *user_params*, *param_defaults=None*)

Return a parameters.Parameters object for the stack.

resource_definitions(*stack*)

Return a dictionary of ResourceDefinition objects.

validate_section(*section*, *sub_section*, *data*, *allowed_keys*)

class heat.engine.hot.template.**HOTemplate20141016**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20130523*

functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>, 'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join': <class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class 'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class 'heat.engine.cfn.functions.Select'>, 'Fn::Split': <class 'heat.engine.hot.functions.Removed'>, 'Ref': <class 'heat.engine.hot.functions.Removed'>, 'get_attr': <class 'heat.engine.hot.functions.GetAtt'>, 'get_file': <class 'heat.engine.hot.functions.GetFile'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'get_resource': <class 'heat.engine.hot.functions.GetResource'>, 'list_join': <class 'heat.engine.hot.functions.Join'>, 'resource_facade': <class 'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class 'heat.engine.hot.functions.Replace'>}

class heat.engine.hot.template.**HOTemplate20150430**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20141016*

functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>, 'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join': <class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class 'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class 'heat.engine.cfn.functions.Select'>, 'Fn::Split': <class 'heat.engine.hot.functions.Removed'>, 'Ref': <class 'heat.engine.hot.functions.Removed'>, 'digest': <class 'heat.engine.hot.functions.Digest'>, 'get_attr': <class 'heat.engine.hot.functions.GetAtt'>, 'get_file': <class 'heat.engine.hot.functions.GetFile'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'get_resource': <class 'heat.engine.hot.functions.GetResource'>, 'list_join': <class 'heat.engine.hot.functions.Join'>, 'repeat': <class 'heat.engine.hot.functions.Repeat'>, 'resource_facade': <class 'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class 'heat.engine.hot.functions.Replace'>}

class heat.engine.hot.template.**HOTemplate20151015**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20150430*

functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>, 'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join': <class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class 'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Split': <class 'heat.engine.hot.functions.Removed'>, 'Ref': <class 'heat.engine.hot.functions.Removed'>, 'digest': <class 'heat.engine.hot.functions.Digest'>, 'get_attr': <class 'heat.engine.hot.functions.GetAttAllAttributes'>, 'get_file': <class 'heat.engine.hot.functions.GetFile'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'get_resource': <class 'heat.engine.hot.functions.GetResource'>, 'list_join': <class 'heat.engine.hot.functions.JoinMultiple'>, 'repeat': <class 'heat.engine.hot.functions.Repeat'>, 'resource_facade': <class 'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class 'heat.engine.hot.functions.ReplaceJson'>, 'str_split': <class 'heat.engine.hot.functions.StrSplit'>}

class heat.engine.hot.template.**HOTemplate20160408**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20151015*

functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>, 'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join': <class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class 'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Split': <class 'heat.engine.hot.functions.Removed'>, 'Ref': <class 'heat.engine.hot.functions.Removed'>, 'digest': <class 'heat.engine.hot.functions.Digest'>, 'get_attr': <class 'heat.engine.hot.functions.GetAttAllAttributes'>, 'get_file': <class 'heat.engine.hot.functions.GetFile'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'get_resource': <class 'heat.engine.hot.functions.GetResource'>, 'list_join': <class 'heat.engine.hot.functions.JoinMultiple'>, 'map_merge': <class 'heat.engine.hot.functions.MapMerge'>, 'repeat': <class 'heat.engine.hot.functions.Repeat'>, 'resource_facade': <class 'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class 'heat.engine.hot.functions.ReplaceJson'>, 'str_split': <class 'heat.engine.hot.functions.StrSplit'>}

class heat.engine.hot.template.**HOTemplate20161014**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20160408*

CONDITIONS = 'conditions'

OUTPUT_CONDITION = 'condition'

OUTPUT_KEYS = ('description', 'value', 'condition')

```
RES_CONDITION = 'condition'

RES_EXTERNAL_ID = 'external_id'

SECTIONS = ('heat_template_version', 'description', 'parameter_groups',
'parameters', 'resources', 'outputs', '__undefined__', 'conditions')

SECTIONS_NO_DIRECT_ACCESS = {'conditions', 'heat_template_version',
'parameters'}

condition_functions = {'and': <class 'heat.engine.hot.functions.And'>,
'equals': <class 'heat.engine.hot.functions.Equals'>, 'get_param':
<class 'heat.engine.hot.functions.GetParam'>, 'not': <class
'heat.engine.hot.functions.Not'>, 'or': <class
'heat.engine.hot.functions.Or'>}

deletion_policies = {'Delete': 'Delete', 'Retain': 'Retain', 'Snapshot':
'Snapshot', 'delete': 'Delete', 'retain': 'Retain', 'snapshot':
'Snapshot'}

functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>,
'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join':
<class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap':
<class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class
'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class
'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class
'heat.engine.hot.functions.Removed'>, 'Fn::Split': <class
'heat.engine.hot.functions.Removed'>, 'Ref': <class
'heat.engine.hot.functions.Removed'>, 'digest': <class
'heat.engine.hot.functions.Digest'>, 'get_attr': <class
'heat.engine.hot.functions.GetAttAllAttributes'>, 'get_file': <class
'heat.engine.hot.functions.GetFile'>, 'get_param': <class
'heat.engine.hot.functions.GetParam'>, 'get_resource': <class
'heat.engine.hot.functions.GetResource'>, 'if': <class
'heat.engine.hot.functions.If'>, 'list_join': <class
'heat.engine.hot.functions.JoinMultiple'>, 'map_merge': <class
'heat.engine.hot.functions.MapMerge'>, 'map_replace': <class
'heat.engine.hot.functions.MapReplace'>, 'repeat': <class
'heat.engine.hot.functions.RepeatWithMap'>, 'resource_facade': <class
'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class
'heat.engine.hot.functions.ReplaceJson'>, 'str_split': <class
'heat.engine.hot.functions.StrSplit'>, 'yaql': <class
'heat.engine.hot.functions.Yaql'>}
```

class heat.engine.hot.template.**HOTemplate20170224**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20161014*

functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>, 'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join': <class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class 'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Split': <class 'heat.engine.hot.functions.Removed'>, 'Ref': <class 'heat.engine.hot.functions.Removed'>, 'digest': <class 'heat.engine.hot.functions.Digest'>, 'filter': <class 'heat.engine.hot.functions.Filter'>, 'get_attr': <class 'heat.engine.hot.functions.GetAttAllAttributes'>, 'get_file': <class 'heat.engine.hot.functions.GetFile'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'get_resource': <class 'heat.engine.hot.functions.GetResource'>, 'if': <class 'heat.engine.hot.functions.If'>, 'list_join': <class 'heat.engine.hot.functions.JoinMultiple'>, 'map_merge': <class 'heat.engine.hot.functions.MapMerge'>, 'map_replace': <class 'heat.engine.hot.functions.MapReplace'>, 'repeat': <class 'heat.engine.hot.functions.RepeatWithMap'>, 'resource_facade': <class 'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class 'heat.engine.hot.functions.ReplaceJson'>, 'str_replace_strict': <class 'heat.engine.hot.functions.ReplaceJsonStrict'>, 'str_split': <class 'heat.engine.hot.functions.StrSplit'>, 'yaql': <class 'heat.engine.hot.functions.Yaql'>}

param_schema_class

alias of *HOTParamSchema20170224*

class heat.engine.hot.template.**HOTemplate20170901**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20170224*

condition_functions = {'and': <class 'heat.engine.hot.functions.And'>, 'contains': <class 'heat.engine.hot.functions.Contains'>, 'equals': <class 'heat.engine.hot.functions.Equals'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'not': <class 'heat.engine.hot.functions.Not'>, 'or': <class 'heat.engine.hot.functions.Or'>, 'yaql': <class 'heat.engine.hot.functions.Yaql'>}

functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>, 'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join': <class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class 'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Split': <class 'heat.engine.hot.functions.Removed'>, 'Ref': <class 'heat.engine.hot.functions.Removed'>, 'contains': <class 'heat.engine.hot.functions.Contains'>, 'digest': <class 'heat.engine.hot.functions.Digest'>, 'filter': <class 'heat.engine.hot.functions.Filter'>, 'get_attr': <class 'heat.engine.hot.functions.GetAttAllAttributes'>, 'get_file': <class 'heat.engine.hot.functions.GetFile'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'get_resource': <class 'heat.engine.hot.functions.GetResource'>, 'if': <class 'heat.engine.hot.functions.If'>, 'list_concat': <class 'heat.engine.hot.functions.ListConcat'>, 'list_concat_unique': <class 'heat.engine.hot.functions.ListConcatUnique'>, 'list_join': <class 'heat.engine.hot.functions.JoinMultiple'>, 'make_url': <class 'heat.engine.hot.functions.MakeURL'>, 'map_merge': <class 'heat.engine.hot.functions.MapMerge'>, 'map_replace': <class 'heat.engine.hot.functions.MapReplace'>, 'repeat': <class 'heat.engine.hot.functions.RepeatWithNestedLoop'>, 'resource_facade': <class 'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class 'heat.engine.hot.functions.ReplaceJson'>, 'str_replace_strict': <class 'heat.engine.hot.functions.ReplaceJsonStrict'>, 'str_replace_vstrict': <class 'heat.engine.hot.functions.ReplaceJsonVeryStrict'>, 'str_split': <class 'heat.engine.hot.functions.StrSplit'>, 'yaql': <class 'heat.engine.hot.functions.Yaql'>}

class heat.engine.hot.template.**HOTemplate20180302**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20170901*

condition_functions = {'and': <class 'heat.engine.hot.functions.And'>, 'contains': <class 'heat.engine.hot.functions.Contains'>, 'equals': <class 'heat.engine.hot.functions.Equals'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'not': <class 'heat.engine.hot.functions.Not'>, 'or': <class 'heat.engine.hot.functions.Or'>, 'yaql': <class 'heat.engine.hot.functions.Yaql'>}

functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>, 'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join': <class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class 'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Split': <class 'heat.engine.hot.functions.Removed'>, 'Ref': <class 'heat.engine.hot.functions.Removed'>, 'contains': <class 'heat.engine.hot.functions.Contains'>, 'digest': <class 'heat.engine.hot.functions.Digest'>, 'filter': <class 'heat.engine.hot.functions.Filter'>, 'get_attr': <class 'heat.engine.hot.functions.GetAttAllAttributes'>, 'get_file': <class 'heat.engine.hot.functions.GetFile'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'get_resource': <class 'heat.engine.hot.functions.GetResource'>, 'if': <class 'heat.engine.hot.functions.If'>, 'list_concat': <class 'heat.engine.hot.functions.ListConcat'>, 'list_concat_unique': <class 'heat.engine.hot.functions.ListConcatUnique'>, 'list_join': <class 'heat.engine.hot.functions.JoinMultiple'>, 'make_url': <class 'heat.engine.hot.functions.MakeURL'>, 'map_merge': <class 'heat.engine.hot.functions.MapMerge'>, 'map_replace': <class 'heat.engine.hot.functions.MapReplace'>, 'repeat': <class 'heat.engine.hot.functions.RepeatWithNestedLoop'>, 'resource_facade': <class 'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class 'heat.engine.hot.functions.ReplaceJson'>, 'str_replace_strict': <class 'heat.engine.hot.functions.ReplaceJsonStrict'>, 'str_replace_vstrict': <class 'heat.engine.hot.functions.ReplaceJsonVeryStrict'>, 'str_split': <class 'heat.engine.hot.functions.StrSplit'>, 'yaql': <class 'heat.engine.hot.functions.Yaql'>}

param_schema_class

alias of *HOTParamSchema20180302*

class heat.engine.hot.template.**HOTemplate20180831**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20180302*

condition_functions = {'and': <class 'heat.engine.hot.functions.And'>, 'contains': <class 'heat.engine.hot.functions.Contains'>, 'equals': <class 'heat.engine.hot.functions.Equals'>, 'get_param': <class 'heat.engine.hot.functions.GetParam'>, 'not': <class 'heat.engine.hot.functions.Not'>, 'or': <class 'heat.engine.hot.functions.Or'>, 'yaql': <class 'heat.engine.hot.functions.Yaql'>}

```
functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>,
'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join':
<class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap':
<class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class
'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class
'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class
'heat.engine.hot.functions.Removed'>, 'Fn::Split': <class
'heat.engine.hot.functions.Removed'>, 'Ref': <class
'heat.engine.hot.functions.Removed'>, 'contains': <class
'heat.engine.hot.functions.Contains'>, 'digest': <class
'heat.engine.hot.functions.Digest'>, 'filter': <class
'heat.engine.hot.functions.Filter'>, 'get_attr': <class
'heat.engine.hot.functions.GetAttAllAttributes'>, 'get_file': <class
'heat.engine.hot.functions.GetFile'>, 'get_param': <class
'heat.engine.hot.functions.GetParam'>, 'get_resource': <class
'heat.engine.hot.functions.GetResource'>, 'if': <class
'heat.engine.hot.functions.If'>, 'list_concat': <class
'heat.engine.hot.functions.ListConcat'>, 'list_concat_unique': <class
'heat.engine.hot.functions.ListConcatUnique'>, 'list_join': <class
'heat.engine.hot.functions.JoinMultiple'>, 'make_url': <class
'heat.engine.hot.functions.MakeURL'>, 'map_merge': <class
'heat.engine.hot.functions.MapMerge'>, 'map_replace': <class
'heat.engine.hot.functions.MapReplace'>, 'repeat': <class
'heat.engine.hot.functions.RepeatWithNestedLoop'>, 'resource_facade':
<class 'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class
'heat.engine.hot.functions.ReplaceJson'>, 'str_replace_strict': <class
'heat.engine.hot.functions.ReplaceJsonStrict'>, 'str_replace_vstrict':
<class 'heat.engine.hot.functions.ReplaceJsonVeryStrict'>, 'str_split':
<class 'heat.engine.hot.functions.StrSplit'>, 'yaql': <class
'heat.engine.hot.functions.Yaql'>}
```

class heat.engine.hot.template.**HOTemplate20210416**(*template*, **args*, ***kwargs*)

Bases: *HOTemplate20180831*

```
functions = {'Fn::Base64': <class 'heat.engine.hot.functions.Removed'>,
'Fn::GetAZs': <class 'heat.engine.hot.functions.Removed'>, 'Fn::Join':
<class 'heat.engine.hot.functions.Removed'>, 'Fn::MemberListToMap':
<class 'heat.engine.hot.functions.Removed'>, 'Fn::Replace': <class
'heat.engine.hot.functions.Removed'>, 'Fn::ResourceFacade': <class
'heat.engine.hot.functions.Removed'>, 'Fn::Select': <class
'heat.engine.hot.functions.Removed'>, 'Fn::Split': <class
'heat.engine.hot.functions.Removed'>, 'Ref': <class
'heat.engine.hot.functions.Removed'>, 'contains': <class
'heat.engine.hot.functions.Contains'>, 'digest': <class
'heat.engine.hot.functions.Digest'>, 'filter': <class
'heat.engine.hot.functions.Filter'>, 'get_attr': <class
'heat.engine.hot.functions.GetAttAllAttributes'>, 'get_file': <class
'heat.engine.hot.functions.GetFile'>, 'get_param': <class
'heat.engine.hot.functions.GetParam'>, 'get_resource': <class
'heat.engine.hot.functions.GetResource'>, 'if': <class
'heat.engine.hot.functions.IfNullable'>, 'list_concat': <class
'heat.engine.hot.functions.ListConcat'>, 'list_concat_unique': <class
'heat.engine.hot.functions.ListConcatUnique'>, 'list_join': <class
'heat.engine.hot.functions.JoinMultiple'>, 'make_url': <class
'heat.engine.hot.functions.MakeURL'>, 'map_merge': <class
'heat.engine.hot.functions.MapMerge'>, 'map_replace': <class
'heat.engine.hot.functions.MapReplace'>, 'repeat': <class
'heat.engine.hot.functions.RepeatWithNestedLoop'>, 'resource_facade':
<class 'heat.engine.hot.functions.ResourceFacade'>, 'str_replace': <class
'heat.engine.hot.functions.ReplaceJson'>, 'str_replace_strict': <class
'heat.engine.hot.functions.ReplaceJsonStrict'>, 'str_replace_vstrict':
<class 'heat.engine.hot.functions.ReplaceJsonVeryStrict'>, 'str_split':
<class 'heat.engine.hot.functions.StrSplit'>, 'yaql': <class
'heat.engine.hot.functions.Yaql'>}
```

Module contents

heat.engine.notification package

Submodules

heat.engine.notification.autoscaling module

`heat.engine.notification.autoscaling.`**`send`**(*stack*, *adjustment=None*, *adjustment_type=None*, *capacity=None*, *groupname=None*, *message='error'*, *suffix=None*)

Send autoscaling notifications to the configured notification driver.

heat.engine.notification.stack module

`heat.engine.notification.stack.`**`send`**(*stack*)

Send usage notifications to the configured notification driver.

Module contents

heat.engine.notification.**get_default_level**()

heat.engine.notification.**list_opts**()

heat.engine.notification.**notify**(*context*, *event_type*, *level*, *body*)

heat.engine.resources package

Submodules

heat.engine.resources.alarm_base module

class heat.engine.resources.alarm_base.**BaseAlarm**(*name*, *definition*, *stack*)

Bases: *Resource*

Base Alarm Manager.

QF_FIELD = 'field'

QF_OP = 'op'

QF_OP_VALS = <heat.engine.constraints.AllowedValues object>

QF_TYPE = 'type'

QF_TYPE_VALS = <heat.engine.constraints.AllowedValues object>

QF_VALUE = 'value'

QUERY_FACTOR_FIELDS = ('field', 'op', 'value', 'type')

actions_to_urls(*props*)

alarm_type = 'threshold'

default_client_name = 'aodh'

entity = 'alarm'

handle_check()

handle_resume()

handle_suspend()

heat.engine.resources.scheduler_hints module

class heat.engine.resources.scheduler_hints.**SchedulerHintsMixin**

Bases: object

Utility class to encapsulate Scheduler Hint related logic.

HEAT_PATH_IN_STACK = 'heat_path_in_stack'

HEAT_RESOURCE_NAME = 'heat_resource_name'

HEAT_RESOURCE_UUID = 'heat_resource_uuid'

HEAT_ROOT_STACK_ID = 'heat_root_stack_id'

HEAT_STACK_ID = 'heat_stack_id'

HEAT_STACK_NAME = 'heat_stack_name'

heat.engine.resources.server_base module

class heat.engine.resources.server_base.**BaseServer**(*name*, *definition*, *stack*)

Bases: *StackUser*

Base Server resource.

property access_key

check_create_complete(*server_id*)

check_delete_complete(*prg*)

entity = 'servers'

handle_delete()

Default implementation; should be overridden by resources.

handle_snapshot_delete(*state*)

handle_update(*json_snippet*, *tmpl_diff*, *prop_diff*)

metadata_update(*new_metadata=None*)

Refresh the metadata if new_metadata is None.

property password

physical_resource_name_limit = 53

property secret_key

transport_poll_server_cfn(*props*)

transport_poll_server_heat(*props*)

transport_poll_temp_url(*props*)

transport_zaqar_message(*props*)

heat.engine.resources.signal_responder module

class heat.engine.resources.signal_responder.**SignalResponder**(*name*, *definition*, *stack*)

Bases: *StackUser*

ATTRIBUTES = ('signal',)

PROPERTIES = ('signal_transport',)

SIGNAL_ATTR = 'signal'

SIGNAL_TRANSPORT = 'signal_transport'

handle_delete()

Default implementation; should be overridden by resources.

property password

requires_deferred_auth = True

heat.engine.resources.stack_resource module

class heat.engine.resources.stack_resource.**StackResource**(*name*, *definition*, *stack*)

Bases: *Resource*

Allows entire stack to be managed as a resource in a parent stack.

An abstract Resource subclass that allows the management of an entire Stack as a resource in a parent stack.

check_adopt_complete(*cookie=None*)

check_check_complete(*cookie=None*)

check_create_complete(*cookie=None*)

check_delete_complete(*cookie=None*)

check_resume_complete(*cookie=None*)

check_suspend_complete(*cookie=None*)

check_update_complete(*cookie=None*)

child_definition(*child_template=None*, *user_params=None*, *nested_identifier=None*)

child_params()

Default implementation to get the child params.

Resources that inherit from StackResource should override this method with specific details about the parameters used by them.

child_template()

Default implementation to get the child template.

Resources that inherit from StackResource should override this method with specific details about the template used by them.

child_template_files(*child_env*)

Default implementation to get the files map for child template.

create_with_template(*child_template*, *user_params=None*, *timeout_mins=None*, *adopt_data=None*)

Create the nested stack with the given template.

delete_nested()

Delete the nested stack.

get_nested_parameters_stack()

Return a stack for schema validation.

This returns a stack to be introspected for building parameters schema. It can be customized by subclass to return a restricted version of what will be running.

get_output(*op*)

Return the specified Output value from the nested stack.

If the output key does not exist, raise a NotFound exception.

handle_check()

handle_create_cancel(*cookie*)

handle_delete()

Default implementation; should be overridden by resources.

handle_preempt()

Pre-empt an in-progress update when a new update is available.

This method is called when a previous convergence update is in progress but a new update for the resource is available. By default it does nothing, but subclasses may override it to cancel the in-progress update if it is safe to do so.

Note that this method does not run in the context of the in-progress update and has no access to runtime information about it; nor is it safe to make changes to the Resource in the database. If implemented, this method should cause the existing update to complete by external means. If this leaves the resource in a FAILED state, that should be taken into account in needs_replace_failed().

handle_resume()

handle_suspend()

handle_update_cancel(*cookie*)

has_nested()

Return True if the resource has an existing nested stack.

nested()

Return a Stack object representing the nested (child) stack.

If we catch NotFound exception when loading, return None.

nested_identifier()

prepare_abandon()

preview()

Preview a StackResource as resources within a Stack.

This method overrides the original Resource.preview to return a preview of all the resources contained in this Stack. For this to be possible, the specific resources need to override both `child_template` and `child_params` with specific information to allow the stack to be parsed correctly. If any of these methods is missing, the entire StackResource will be returned as if it were a regular Resource.

requires_deferred_auth = True

property template_url

Template url for the stack resource.

When stack resource is a TemplateResource, its the template location. For group resources like ResourceGroup where the template is constructed dynamically, its just a placeholder.

translate_remote_exceptions(*ex*)

update_with_template(*child_template*, *user_params=None*, *timeout_mins=None*)

Update the nested stack with the new template.

validate()

Validate the resource.

This may be overridden by resource plugins to add extra validation logic specific to the resource implementation.

validate_nested_stack()

heat.engine.resources.stack_user module

class heat.engine.resources.stack_user.**StackUser**(*name*, *definition*, *stack*)

Bases: *Resource*

handle_create()

handle_delete()

Default implementation; should be overridden by resources.

handle_resume()

handle_suspend()

heat.engine.resources.template_resource module

class heat.engine.resources.template_resource.**TemplateResource**(*name*, *definition*, *stack*)

Bases: *StackResource*

A resource implemented by a nested stack.

This implementation passes resource properties as parameters to the nested stack. Outputs of the nested stack are exposed as attributes of this resource.

child_params()

Override method of child_params for the resource.

Returns

parameter values for our nested stack based on our properties

child_template()

Default implementation to get the child template.

Resources that inherit from StackResource should override this method with specific details about the template used by them.

get_attribute(*key*, **path*)

Default implementation for function get_attr and Fn::GetAtt.

This may be overridden by resource plugins to add extra logic specific to the resource implementation.

get_reference_id()

Default implementation for function get_resource.

This may be overridden by resource plugins to add extra logic specific to the resource implementation.

static get_schemas(*tmpl*, *param_defaults*)

static get_template_file(*template_name*, *allowed_schemes*)

handle_adopt(*resource_data=None*)

handle_create()

handle_update(*json_snippet*, *tmpl_diff*, *prop_diff*)

metadata_update(*new_metadata=None*)

Refresh the metadata if new_metadata is None.

regenerate_info_schema(*definition*)

Default implementation; should be overridden by resources.

Should be overridden by resources that would require schema refresh during update, ex. TemplateResource.

Definition
Resource Definition

template_data()

property template_url

Template url for the stack resource.

When stack resource is a TemplateResource, its the template location. For group resources like ResourceGroup where the template is constructed dynamically, its just a placeholder.

validate()

Validate the resource.

This may be overridden by resource plugins to add extra validation logic specific to the resource implementation.

validate_template()

Validate structural/syntax aspects of the resource definition.

Resource plugins should not override this, because this interface is expected to be called pre-create so things normally valid in an overridden validate() such as accessing properties may not work.

heat.engine.resources.template_resource.**generate_class_from_template**(*name*, *data*, *param_defaults*)

heat.engine.resources.volume_base module

class heat.engine.resources.volume_base.**BaseVolume**(*name*, *definition*, *stack*)

Bases: *Resource*

Base Volume Manager.

check_create_complete(*vol_id*)

check_delete_complete(*prg*)

default_client_name = 'cinder'

handle_check()

handle_create()

handle_delete()

Default implementation; should be overridden by resources.

handle_snapshot_delete(*state*)

classmethod validate_deletion_policy(*policy*)

class heat.engine.resources.volume_base.**BaseVolumeAttachment**(*name*, *definition*, *stack*)

Bases: *Resource*

Base Volume Attachment Manager.

check_create_complete(*volume_id*)

check_delete_complete(*prg*)

default_client_name = 'cinder'

handle_create()

handle_delete()

Default implementation; should be overridden by resources.

heat.engine.resources.wait_condition module

class heat.engine.resources.wait_condition.**BaseWaitConditionHandle**(*name*, *definition*, *stack*)

Bases: *SignalResponder*

Base WaitConditionHandle resource.

The main point of this class is to : - have no dependencies (so the instance can reference it) - create credentials to allow for signalling from the instance. - handle signals from the instance, validate and store result

STATUS_FAILURE = 'FAILURE'

STATUS_SUCCESS = 'SUCCESS'

WAIT_STATUSES = ('FAILURE', 'SUCCESS')

get_status()

Return a list of the Status values for the handle signals.

get_status_reason(*status*)

Return a list of reasons associated with a particular status.

handle_create()

handle_signal(*details=None*)

normalise_signal_data(*signal_data*, *latest_metadata*)

properties_schema = {}

exception heat.engine.resources.wait_condition.**WaitConditionFailure**(*wait_condition*, *handle*)

Bases: *Error*

exception heat.engine.resources.wait_condition.**WaitConditionTimeout**(*wait_condition*, *handle*)

Bases: *Error*

Module contents

heat.engine.resources.**global_env**()

heat.engine.resources.**initialise**()

heat.engine.resources.**list_opts**()

Submodules

heat.engine.api module

heat.engine.api.**extract_args**(*params*)

Extract arguments passed as parameters and return them as a dictionary.

Extract any arguments passed as parameters through the API and return them as a dictionary. This allows us to filter the passed args and do type conversion where appropriate

heat.engine.api.**format_event**(*event*, *stack_identifier*, *root_stack_identifier=None*, *include_rsrc_prop_data=True*)

heat.engine.api.**format_notification_body**(*stack*)

heat.engine.api.**format_resource_attributes**(*resource*, *with_attr=None*)

heat.engine.api.**format_resource_properties**(*resource*)

heat.engine.api.**format_resource_snapshot**(*rsrc_snapshot*)

heat.engine.api.**format_snapshot**(*snapshot*)

heat.engine.api.**format_software_config**(*sc*, *detail=True*, *include_project=False*)

heat.engine.api.**format_software_deployment**(*sd*)

heat.engine.api.**format_stack**(*stack*, *preview=False*, *resolve_outputs=True*)

Return a representation of the given stack.

Return a representation of the given stack that matches the API output expectations.

heat.engine.api.**format_stack_db_object**(*stack*)

Return a summary representation of the given stack.

Given a stack versioned DB object, return a representation of the given stack for a stack listing.

heat.engine.api.**format_stack_output**(*output_defn*, *resolve_value=True*)

heat.engine.api.**format_stack_outputs**(*outputs*, *resolve_value=False*)

Return a representation of the given output template.

Return a representation of the given output template for the given stack that matches the API output expectations.

heat.engine.api.**format_stack_preview**(*stack*)

heat.engine.api.**format_stack_resource**(*resource*, *detail=True*, *with_props=False*, *with_attr=None*)

Return a representation of the given resource.

Return a representation of the given resource that matches the API output expectations.

heat.engine.api.**format_validate_parameter**(*param*)

Format a template parameter for validate template API call.

Formats a template parameter and its schema information from the engines internal representation (i.e. a Parameter object and its associated Schema object) to a representation expected by the current API (for example to be compatible to CFN syntax).

heat.engine.api.**format_watch**(*watch*)

heat.engine.api.**format_watch_data**(*wd*, *rule_names*)

heat.engine.api.**translate_filters**(*params*)

Translate filter names to their corresponding DB field names.

Parameters

params A dictionary containing keys from engine.api.STACK_KEYS and other keys previously leaked to users.

Returns

A dict containing only valid DB filed names.

heat.engine.attributes module

class heat.engine.attributes.**Attribute**(*attr_name*, *schema*)

Bases: `object`

An Attribute schema.

as_output(*resource_name*, *template_type='cfn'*)

Output entry for a provider template with the given resource name.

Parameters

- **resource_name** the logical name of the provider resource
- **template_type** the template type to generate

Returns

This attribute as a template Output entry for cfn template and output entry for hot template

support_status()

class heat.engine.attributes.**Attributes**(*res_name*, *schema*, *resolver*)

Bases: Mapping

Models a collection of Resource Attributes.

static as_outputs(*resource_name*, *resource_class*, *template_type='cfn'*)

Dict of Output entries for a provider template with resource name.

Parameters

- **resource_name** logical name of the resource
- **resource_class** resource implementation class

Returns

The attributes of the specified resource_class as a template Output map

property cached_attrs

get_cache_mode(*attribute_name*)

Return the cache mode for the specified attribute.

If the attribute is not defined in the schema, the default cache mode (CACHE_LOCAL) is returned.

has_new_cached_attrs()

Returns True if cached_attrs have changed

Allows the caller to determine if this instances cached_attrs have been updated since they were initially set (if at all).

reset_resolved_values()

static schema_from_outputs(*json_snippet*)

set_cached_attr(*key*, *value*)

set_schema(*schema*)

class heat.engine.attributes.**Schema**(*description=None*, *support_status=<heat.engine.support.SupportStatus object>*, *cache_mode='cache_local'*, *type=None*)

Bases: *Schema*

Simple schema class for attributes.

Schema objects are serializable to dictionaries following a superset of the HOT input Parameter schema using dict().

BOOLEAN = 'Boolean'

CACHE_LOCAL = 'cache_local'

CACHE_MODES = ('cache_local', 'cache_none')

CACHE_NONE = 'cache_none'

DESCRIPTION = 'description'

INTEGER = 'Integer'

KEYS = ('description', 'type')

LIST = 'List'

MAP = 'Map'

STRING = 'String'

TYPE = 'type'

TYPES = (None, 'String', 'Map', 'List', 'Integer', 'Boolean')

UNKNOWN = None

classmethod from_attribute(*schema_dict*)

Return a Property Schema corresponding to a Attribute Schema.

heat.engine.attributes.**schemata**(*schema*)

Return dictionary of Schema objects for given dictionary of schemata.

heat.engine.attributes.**select_from_attribute**(*attribute_value*, *path*)

Select an element from an attribute value.

Parameters

- **attribute_value** the attribute value.
- **path** a list of path components to select from the attribute.

Returns

the selected attribute component value.

heat.engine.check_resource module

exception heat.engine.check_resource.**CancelOperation**

Bases: BaseException

Exception to cancel an in-progress operation on a resource.

This exception is raised when operations on a resource are cancelled.

class heat.engine.check_resource.**CheckResource**(*engine_id*, *rpc_client*, *thread_group_mgr*, *msg_queue*, *input_data*)

Bases: object

check(*cnxt*, *resource_id*, *current_traversal*, *resource_data*, *is_update*, *adopt_stack_data*, *rsrc*, *stack*, *skip_propagate=False*, *accumulated_failures=None*)

Process a node in the dependency graph.

The node may be associated with either an update or a cleanup of its associated resource.

check_delete_snapshot(*cnxt*, *rsrc*, *snapshot*)

retrigger_check_resource(*cnxt*, *resource_id*, *stack*)

heat.engine.check_resource.**check_resource_cleanup**(*rsrc*, *template_id*, *engine_id*, *stack*, *msg_queue*)

Delete the Resource if appropriate.

heat.engine.check_resource.**check_resource_update**(*rsrc*, *template_id*, *requires*, *engine_id*, *stack*, *msg_queue*)

Create or update the Resource if appropriate.

heat.engine.check_resource.**check_snapshot_complete**(*cnxt*, *snapshot*, *sender_id*, *deps*, *is_update*)

Mark the snapshot complete if the update is complete.

heat.engine.check_resource.**check_stack_complete**(*cnxt*, *stack*, *current_traversal*, *sender_id*, *deps*, *is_update*, *rsrc_failure=None*, *node_type='resource'*)

Mark the stack complete if the update is complete.

Complete is currently in the sense that all desired resources are in service, not that superfluous ones have been cleaned up.

heat.engine.check_resource.**do_snapshot_delete**(*rsrc*, *snapshot*, *msg_queue*, *engine_id*)

heat.engine.check_resource.**load_resource**(*cnxt*, *resource_id*, *resource_data*, *current_traversal*, *is_update*)

heat.engine.check_resource.**load_resource_from_snapshot**(*cnxt*, *rsrc_name*, *snapshot_id*, *thread_group_mgr*, *is_stack_delete*, *current_traversal*, *start_time*)

heat.engine.check_resource.**propagate_check_resource**(*cnxt*, *rpc_client*, *next_res_id*, *current_traversal*, *predecessors*, *sender_key*, *sender_data*, *is_update*, *adopt_stack_data*, *is_skip=False*, *rsrc_failure=None*, *converge=False*, *node_type='resource'*)

Trigger processing of node if all of its dependencies are satisfied.

heat.engine.conditions module

class heat.engine.conditions.**Conditions**(*conditions_dict*)

Bases: object

is_enabled(*condition_name*)

validate()

heat.engine.constraints module

class heat.engine.constraints.**AllowedPattern**(*pattern*, *description=None*)

Bases: *Constraint*

Constrain values to a predefined regular expression pattern.

Serializes to JSON as:

```
{
    'allowed_pattern': <pattern>,
    'description': <description>
}
```

valid_types = ('STRING',)

class heat.engine.constraints.**AllowedValues**(*allowed*, *description=None*)

Bases: *Constraint*

Constrain values to a predefined set.

Serializes to JSON as:

```
{
    'allowed_values': [<allowed1>, <allowed2>, ...],
    'description': <description>
}
```

valid_types = ('STRING', 'INTEGER', 'NUMBER', 'BOOLEAN', 'LIST')

class heat.engine.constraints.**AnyIndexDict**(*value*)

Bases: Mapping

A Mapping that returns the same value for any integer index.

Used for storing the schema for a list. When converted to a dictionary, it contains a single item with the key *.

ANYTHING = '*'

class heat.engine.constraints.**BaseCustomConstraint**

Bases: object

A base class for validation using API clients.

It will provide a better error message, and reduce a bit of duplication. Subclass must provide *expected_exceptions* and implement *validate_with_client*.

error(*value*)

expected_exceptions = (<class 'heat.common.exception.EntityNotFound'>,)

resource_client_name = None

resource_getter_name = None

validate(*value*, *context*)

validate_with_client(*client*, *resource_id*)

class heat.engine.constraints.**Constraint**(*description=None*)

Bases: Mapping

Parent class for constraints on allowable values for a Property.

Constraints are serializable to dictionaries following the HOT input Parameter constraints schema using dict().

DESCRIPTION = 'description'

validate(*value*, *schema=None*, *context=None*)

class heat.engine.constraints.**CustomConstraint**(*name*, *description=None*, *environment=None*)

Bases: *Constraint*

A constraint delegating validation to an external class.

property custom_constraint

valid_types = ('STRING', 'INTEGER', 'NUMBER', 'BOOLEAN', 'LIST')

class heat.engine.constraints.**Length**(*min=None*, *max=None*, *description=None*)

Bases: *Range*

Constrain the length of values within a range.

Serializes to JSON as:

```
{
    'length': {'min': <min>, 'max': <max>},
    'description': <description>
}
```

valid_types = ('STRING', 'LIST', 'MAP')

class heat.engine.constraints.**Modulo**(*step=None*, *offset=None*, *description=None*)

Bases: *Constraint*

Constrain values to modulo.

Serializes to JSON as:

```
{
    'modulo': {'step': <step>, 'offset': <offset>},
    'description': <description>
}
```

OFFSET = 'offset'

STEP = 'step'

valid_types = ('INTEGER', 'NUMBER')

class heat.engine.constraints.**Range**(*min=None*, *max=None*, *description=None*)

Bases: *Constraint*

Constrain values within a range.

Serializes to JSON as:

```
{
    'range': {'min': <min>, 'max': <max>},
    'description': <description>
}
```

MAX = 'max'

MIN = 'min'

valid_types = ('INTEGER', 'NUMBER')

class heat.engine.constraints.**Schema**(*data_type*, *description=None*, *default=None*, *schema=None*, *required=False*, *constraints=None*, *label=None*, *immutable=False*)

Bases: Mapping

Schema base class for validating properties or parameters.

Schema objects are serializable to dictionaries following a superset of the HOT input Parameter schema using dict().

Serialises to JSON in the form:

```
{
    'type': 'list',
    'required': False
    'constraints': [
        {
            'length': {'min': 1},
            'description': 'List must not be empty'
        }
    ],
    'schema': {
        '*': {
            'type': 'string'
        }
    },
    'description': 'An example list property.'
}
```

ANY = 'Any'

BOOLEAN = 'Boolean'

BOOLEAN_TYPE = 'BOOLEAN'

CONSTRAINTS = 'constraints'

DEFAULT = 'default'

DESCRIPTION = 'description'

IMMUTABLE = 'immutable'

INTEGER = 'Integer'

INTEGER_TYPE = 'INTEGER'

KEYS = ('type', 'description', 'default', 'schema', 'required', 'constraints', 'immutable')

LIST = 'List'

LIST_TYPE = 'LIST'

MAP = 'Map'

MAP_TYPE = 'MAP'

NUMBER = 'Number'

NUMBER_TYPE = 'NUMBER'

REQUIRED = 'required'

SCHEMA = 'schema'

STRING = 'String'

STRING_TYPE = 'STRING'

TYPE = 'type'

TYPES = ('Integer', 'String', 'Number', 'Boolean', 'Map', 'List', 'Any')

TYPE_KEYS = ('INTEGER', 'STRING', 'NUMBER', 'BOOLEAN', 'MAP', 'LIST')

set_default(*default=None*)

Set the default value for this Schema object.

static str_to_num(*value*)

Convert a string representation of a number into a numeric type.

to_schema_type(*value*)

Returns the value in the schemas data type.

validate(*context=None*)

Validates the schema.

This method checks if the schema itself is valid, and if the default value - if present - complies to the schemas constraints.

validate_constraints(*value*, *context=None*, *skipped=None*)

heat.engine.dependencies module

class heat.engine.dependencies.**Dependencies**(*edges=None*)

Bases: object

Helper class for calculating a dependency graph.

graph(*reverse=False*)

Return a copy of the underlying dependency graph.

leaves()

Return an iterator over all of the leaf nodes in the graph.

required_by(*last*)

List the keys that require the specified node.

requires(*source*)

List the keys that the specified node requires.

roots()

Return an iterator over all of the root nodes in the graph.

translate(*transform*)

Translate all of the nodes using a transform function.

Returns a new Dependencies object.

class heat.engine.dependencies.**Graph**(**args*)

Bases: defaultdict

A mutable mapping of objects to nodes in a dependency graph.

copy()

Return a copy of the graph.

edges()

Return an iterator over all of the edges in the graph.

map(*func*)

Map the supplied function onto each node in the graph.

Return a dictionary derived from mapping the supplied function onto each node in the graph.

reverse_copy()

Return a copy of the graph with the edges reversed.

static toposort(*graph*)

Return a topologically sorted iterator over a dependency graph.

This is a destructive operation for the graph.

class heat.engine.dependencies.**Node**(*requires=None*, *required_by=None*)

Bases: object

A node in a dependency graph.

copy()

Return a copy of the node.

disjoint()

Return True if this node is both a leaf and a stem.

require

required_by(*source=None*)

List the keys that require this node, and optionally add new one.

requires(*target=None*)

List the keys that this node requires, and optionally add a new one.

reverse_copy()

Return a copy of the node with the edge directions reversed.

satisfy

stem()

Return True if this node is a stem (required by nothing).

heat.engine.environment module

class heat.engine.environment.**ClassResourceInfo**(*registry*, *path*, *value*)

Bases: *ResourceInfo*

Store the mapping of resource name to python class implementation.

description = 'Plugin'

get_class(*files=None*)

class heat.engine.environment.**Environment**(*env=None*, *user_env=True*)

Bases: object

env_as_dict()

Get the entire environment as a dict.

get_class(*resource_type*, *resource_name=None*, *files=None*)

get_class_to_instantiate(*resource_type*, *resource_name=None*)

get_constraint(*name*)

get_event_sinks()

get_resource_info(*resource_type*, *resource_name=None*, *registry_type=None*, *ignore=None*)

get_stack_lifecycle_plugins()

get_types(*cnxt=None*, *support_status=None*, *type_name=None*, *version=None*, *with_description=False*)

load(*env_snippet*)

register_class(*resource_type*, *resource_class*, *path=None*)

register_constraint(*constraint_name*, *constraint*)

register_event_sink(*event_sink_name*, *event_sink_class*)

register_stack_lifecycle_plugin(*stack_lifecycle_name*, *stack_lifecycle_class*)

user_env_as_dict()

Get the environment as a dict, only user-allowed keys.

class heat.engine.environment.**GlobResourceInfo**(*registry*, *path*, *value*)

Bases: *MapResourceInfo*

Store the mapping (with wild cards) of one resource type to another.

like: OS::Networking::* -> OS::Neutron::*

Also supports many-to-one mapping (mostly useful together with special OS::Heat::None resource)

like: OS::* -> OS::Heat::None

description = 'Wildcard Mapping'

get_resource_info(*resource_type=None*, *resource_name=None*)

matches(*resource_type*)

class heat.engine.environment.**MapResourceInfo**(*registry*, *path*, *value*)

Bases: *ResourceInfo*

Store the mapping of one resource type to another.

like: OS::Networking::FloatingIp -> OS::Neutron::FloatingIp

description = 'Mapping'

get_class(*files=None*)

get_resource_info(*resource_type=None*, *resource_name=None*)

class heat.engine.environment.**ResourceInfo**(*registry*, *path*, *value*)

Bases: object

Base mapping of resource type to implementation.

get_class()

get_class_to_instantiate()

get_resource_info(*resource_type=None*, *resource_name=None*)

matches(*resource_type*)

name

path

property registry

user_resource

value

class heat.engine.environment.**ResourceRegistry**(*global_registry*, *param_defaults*)

Bases: object

By looking at the environment, find the resource implementation.

as_dict()

Return user resources in a dict format.

get_class(*resource_type*, *resource_name=None*, *files=None*)

get_class_to_instantiate(*resource_type*, *resource_name=None*)

get_resource_info(*resource_type*, *resource_name=None*, *registry_type=None*, *ignore=None*)

Find possible matches to the resource type and name.

Chain the results from the global and user registry to find a match.

get_rsrc_restricted_actions(*resource_name*)

Returns a set of restricted actions.

For a given resource we get the set of restricted actions.

Actions are set in this format via *resources*:

```
{
    "restricted_actions": [update, replace]
}
```

A restricted_actions value is either *update*, *replace* or a list of those values. Resources support wildcard matching. The asterisk sign matches everything.

get_types(*cnxt=None*, *support_status=None*, *type_name=None*, *version=None*, *with_description=False*)

Return a list of valid resource types.

iterable_by(*resource_type*, *resource_name=None*)

load(*json_snippet*)

log_resource_info(*show_all=False*, *prefix=None*)

matches_hook(*resource_name*, *hook*)

Return whether a resource have a hook set in the environment.

For a given resource and a hook type, we check to see if the passed group of resources has the right hook associated with the name.

Hooks are set in this format via *resources*:

```
{
    "res_name": {
        "hooks": [pre-create, pre-update]
```

(continues on next page)

(continued from previous page)

```
    },
    "*_suffix": {
        "hooks": pre-create
    },
    "prefix_*": {
        "hooks": pre-update
    }
}
```

A hook value is either *pre-create*, *pre-update* or a list of those values. Resources support wildcard matching. The asterisk sign matches everything.

register_class(*resource_type*, *resource_class*, *path=None*)

remove_item(*info*)

remove_resources_except(*resource_name*)

class heat.engine.environment.**TemplateResourceInfo**(*registry*, *path*, *value*)

Bases: *ResourceInfo*

Store the info needed to start a TemplateResource.

description = 'Template'

get_class(*files=None*)

get_class_to_instantiate()

template_name

heat.engine.environment.**get_child_environment**(*parent_env*, *child_params*, *item_to_remove=None*, *child_resource_name=None*)

Build a child environment using the parent environment and params.

This is built from the child_params and the parent env so some resources can use user-provided parameters as if they come from an environment.

1. resource_registry must be merged (child env should be loaded after the parent env to take precedence).
2. child parameters must overwrite the parents as they wont be relevant in the child template.

If *child_resource_name* is provided, resources in the registry will be replaced with the contents of the matching child resource plus anything that passes a wildcard match.

heat.engine.environment.**is_hook_definition**(*key*, *value*)

heat.engine.environment.**is_valid_restricted_action**(*key*, *value*)

heat.engine.environment.**read_global_environment**(*env*, *env_dir=None*)

heat.engine.environment.**valid_hook_type**(*hook*)

heat.engine.environment.**valid_restricted_actions**(*action*)

heat.engine.event module

class heat.engine.event.**Event**(*context*, *stack*, *action*, *status*, *reason*, *physical_resource_id*, *resource_prop_data_id*, *resource_properties*, *resource_name*, *resource_type*, *uuid=None*, *timestamp=None*, *id=None*)

Bases: object

Class representing a Resource state change.

as_dict()

identifier()

Return a unique identifier for the event.

store()

Store the Event in the database.

heat.engine.function module

class heat.engine.function.**Function**(*stack*, *fn_name*, *args*)

Bases: object

Abstract base class for template functions.

all_dep_attrs()

Return resource, attribute name pairs of all attributes referenced.

Return an iterator over the resource name, attribute name tuples of all attributes that this function references.

The special value heat.engine.attributes.ALL_ATTRIBUTES may be used to indicate that all attributes of the resource are required.

By default this calls the dep_attrs() method, but subclasses can override to provide a more efficient implementation.

dep_attrs(*resource_name*)

Return the attributes of the specified resource that are referenced.

Return an iterator over any attributes of the specified resource that this function references.

The special value heat.engine.attributes.ALL_ATTRIBUTES may be used to indicate that all attributes of the resource are required.

dependencies(*path*)

abstract result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

property stack

validate()

Validate arguments without resolving the function.

Function subclasses must override this method to validate their args.

class heat.engine.function.**Invalid**(*stack*, *fn_name*, *args*)

Bases: *Function*

A function for checking condition functions and to force failures.

This function is used to force failures for functions that are not supported in condition definition.

result()

Return the result of resolving the function.

Function subclasses must override this method to calculate their results.

class heat.engine.function.**Macro**(*stack*, *fn_name*, *raw_args*, *parse_func*, *template*)

Bases: *Function*

Abstract base class for template macros.

A macro differs from a function in that it controls how the template is parsed. As such, it operates on the syntax tree itself, not on the parsed output.

all_dep_attrs()

Return resource, attribute name pairs of all attributes referenced.

Return an iterator over the resource name, attribute name tuples of all attributes that this function references.

The special value heat.engine.attributes.ALL_ATTRIBUTES may be used to indicate that all attributes of the resource are required.

By default this calls the dep_attrs() method, but subclasses can override to provide a more efficient implementation.

dep_attrs(*resource_name*)

Return the attributes of the specified resource that are referenced.

Return an iterator over any attributes of the specified resource that this function references.

The special value heat.engine.attributes.ALL_ATTRIBUTES may be used to indicate that all attributes of the resource are required.

dependencies(*path*)

abstract parse_args(*parse_func*)

Parse the macro using the supplied parsing function.

Macro subclasses should override this method to control parsing of the arguments.

result()

Return the resolved result of the macro contents.

property template

validate()

Validate arguments without resolving the result.

heat.engine.function.**all_dep_attrs**(*snippet*)

Iterator over resource, attribute name pairs referenced in a snippet.

The snippet should be already parsed to insert Function objects where appropriate.

Returns
an iterator over the resource name, attribute name tuples of all attributes that are referenced in the template snippet.

`heat.engine.function.dep_attrs(snippet, resource_name)`

Iterator over dependent attrs of a resource in a template snippet.

The snippet should be already parsed to insert Function objects where appropriate.

Returns
an iterator over the attributes of the specified resource that are referenced in the template snippet.

`heat.engine.function.dependencies(snippet, path="")`

Return an iterator over Resource dependencies in a template snippet.

The snippet should be already parsed to insert Function objects where appropriate.

`heat.engine.function.resolve(snippet, nullable=False)`

`heat.engine.function.validate(snippet, path=None)`

heat.engine.lifecycle_plugin module

class `heat.engine.lifecycle_plugin.LifecyclePlugin`

Bases: `object`

Base class for pre-op and post-op work on a stack.

Implementations should extend this class and override the methods.

`do_post_op(cnxt, stack, current_stack=None, action=None, is_stack_failure=False)`

Method to be run by heat after stack operations, including failures.

On failure to execute all the registered pre_ops, this method will be called if and only if the corresponding pre_op was successfully called. On failures of the actual stack operation, this method will be called if all the pre operations were successfully called.

`do_pre_op(cnxt, stack, current_stack=None, action=None)`

Method to be run by heat before stack operations.

`get_ordinal()`

Get the sort order for pre and post operation execution.

The values returned by get_ordinal are used to create a partial order for pre and post operation method invocations. The default ordinal value of 100 may be overridden. If class1inst.ordinal() < class2inst.ordinal(), then the method on class1inst will be executed before the method on class2inst. If class1inst.ordinal() > class2inst.ordinal(), then the method on class1inst will be executed after the method on class2inst. If class1inst.ordinal() == class2inst.ordinal(), then the order of method invocation is indeterminate.

heat.engine.node_data module

class `heat.engine.node_data.NodeData(primary_key, resource_name, uuid, reference_id, attributes, action, status)`

Bases: `object`

Data about a node in the graph, to be passed along to other nodes.

action

as_dict()

Return a dict representation of the data.

This is the format that is serialised and stored in the databases SyncPoints.

attribute(*attr_name*)

Return the specified attribute value.

attribute_names()

Iterate over valid top-level attribute names.

attributes()

Return a dict of all available top-level attribute values.

classmethod from_dict(*node_data*)

Create a new NodeData object from deserialised data.

This reads the format that is stored in the database, and is the inverse of as_dict().

name

primary_key

reference_id()

Return the reference ID of the resource.

i.e. the result that the {get_resource: } intrinsic function should return for this resource.

status

uuid

heat.engine.node_data.**load_resources_data**(*data*)

Return the data for all of the resources that meet at a SyncPoint.

The input is the input_data dict from a SyncPoint received over RPC. The keys (which are ignored) are resource primary keys.

The output is a dict of NodeData objects with the resource names as the keys.

heat.engine.output module

class heat.engine.output.**OutputDefinition**(*name*, *value*, *description=None*)

Bases: object

A definition of a stack output, independent of any template format.

dep_attrs(*resource_name*, *load_all=False*)

Iterate over attributes of a given resource that this references.

Return an iterator over dependent attributes for specified resource_name in the outputs value field.

description()

Return a description of the output.

get_value()

Resolve the value of the output.

render_hot()

required_resource_names()

validate()

Validate the output value without resolving it.

heat.engine.parameter_groups module

class heat.engine.parameter_groups.**ParameterGroups**(*tmpl*)

Bases: object

The ParameterGroups specified by the stacks template.

validate()

Validate the parameter group.

Validate that each parameter belongs to only one Parameter Group and that each parameter name in the group references a valid parameter.

heat.engine.parameters module

class heat.engine.parameters.**BooleanParam**(*name*, *schema*, *value=None*)

Bases: *Parameter*

A template parameter of type Boolean.

value()

Get the parameter value, optionally sanitising it for output.

class heat.engine.parameters.**CommaDelimitedListParam**(*name*, *schema*, *value=None*)

Bases: *ParsedParameter*, Sequence

A template parameter of type CommaDelimitedList.

default_parsed()

parse(*value*)

value()

Get the parameter value, optionally sanitising it for output.

class heat.engine.parameters.**JsonParam**(*name*, *schema*, *value=None*)

Bases: *ParsedParameter*

A template parameter whos value is map or list.

default_parsed()

parse(*value*)

value()

Get the parameter value, optionally sanitising it for output.

class heat.engine.parameters.**NumberParam**(*name*, *schema*, *value=None*)

Bases: *Parameter*

A template parameter of type Number.

value()

Get the parameter value, optionally sanitising it for output.

class heat.engine.parameters.**Parameter**(*name*, *schema*, *value=None*)

Bases: object

A template parameter.

default()

Return the default value of the parameter.

description()

Return the description of the parameter.

has_default()

Return whether the parameter has a default value.

has_value()

Parameter has a user or default value.

hidden()

Return whether the parameter is hidden.

Hidden parameters should be sanitised in any output to the user.

label()

Return the label or param name.

name

schema

set_default(*value*)

tags()

Return the tags associated with the parameter

user_default

user_value

validate(*validate_value=True*, *context=None*)

Validates the parameter.

This method validates if the parameters schema is valid, and if the default value - if present - or the user-provided value for the parameter comply with the schema.

value()

Get the parameter value, optionally sanitising it for output.

class heat.engine.parameters.**Parameters**(*stack_identifier*, *tmpl*, *user_params=None*, *param_defaults=None*)

Bases: Mapping

Parameters of a stack.

The parameters of a stack, with type checking, defaults, etc. specified by the stacks template.

immutable_params_modified(*new_parameters*, *input_params*)

map(*func*, *filter_func=<function Parameters.<lambda>*)

Map the supplied function onto each Parameter.

Map the supplied function onto each Parameter (with an optional filter function) and return the resulting dictionary.

set_stack_id(*stack_identifier*)

Set the StackId pseudo parameter value.

validate(*validate_value=True*, *context=None*)

Validates all parameters.

This method validates if all user-provided parameters are actually defined in the template, and if all parameters are valid.

class heat.engine.parameters.**ParsedParameter**(*name*, *schema*, *value=None*)

Bases: *Parameter*

A template parameter with cached parsed value.

property parsed

class heat.engine.parameters.**Schema**(*data_type*, *description=None*, *default=None*, *schema=None*, *constraints=None*, *hidden=False*, *label=None*, *immutable=False*, *tags=None*)

Bases: *Schema*

Parameter schema.

BOOLEAN = 'Boolean'

CONSTRAINTS = 'Constraints'

DEFAULT = 'Default'

DESCRIPTION = 'Description'

HIDDEN = 'NoEcho'

IMMUTABLE = 'Immutable'

KEYS = ('Type', 'Description', 'Default', 'Schema', 'Constraints', 'NoEcho', 'Label', 'Immutable', 'Tags')

LABEL = 'Label'

LIST = 'CommaDelimitedList'

MAP = 'Json'

NUMBER = 'Number'

PARAMETER_KEYS = ('Type', 'Default', 'NoEcho', 'AllowedValues', 'AllowedPattern', 'MaxLength', 'MinLength', 'MaxValue', 'MinValue', 'Description', 'ConstraintDescription', 'Label')

SCHEMA = 'Schema'

STRING = 'String'

TAGS = 'Tags'

TYPE = 'Type'

TYPES = ('String', 'Number', 'CommaDelimitedList', 'Json', 'Boolean')

classmethod from_dict(*param_name*, *schema_dict*)

Return a Parameter Schema object from a legacy schema dictionary.

Parameters

param_name (*str*) name of the parameter owning the schema; used for more verbose logging

static get_num(*key*, *context*)

set_default(*default=None*)

Set the default value for this Schema object.

validate_value(*value*, *context=None*)

class heat.engine.parameters.**StringParam**(*name*, *schema*, *value=None*)

Bases: *Parameter*

A template parameter of type String.

value()

Get the parameter value, optionally sanitising it for output.

heat.engine.parent_rsrc module

class heat.engine.parent_rsrc.**ParentResourceProxy**(*context*, *parent_resource_name*, *parent_stack_id*)

Bases: object

Proxy for the TemplateResource that owns a provider stack.

This is an interface through which the Fn::ResourceFacade/resource_facade intrinsic functions in a stack can access data about the TemplateResource in the parent stack for which it was created.

This API can be considered stable by third-party Function plugins, and no part of it should be changed or removed without an appropriate deprecation process.

metadata_get()

Return the resource metadata.

property t

The resource definition.

heat.engine.parent_rsrc.**use_parent_stack**(*parent_proxy*, *stack*)

heat.engine.plugin_manager module

class heat.engine.plugin_manager.**PluginManager**(**extra_packages*)

Bases: object

A class for managing plugin modules.

map_to_modules(*function*)

Iterate over the results of calling a function on every module.

class heat.engine.plugin_manager.**PluginMapping**(*names*, **args*, ***kwargs*)

Bases: object

A class for managing plugin mappings.

load_all(*plugin_manager*)

Iterate over the mappings from all modules in the plugin manager.

Mappings are returned as a list of (key, value) tuples.

load_from_module(*module*)

Return the mapping specified in the given module.

If no such mapping is specified, an empty dictionary is returned.

heat.engine.properties module

class heat.engine.properties.**Properties**(*schema*, *data*, *resolver=<function _default_resolver>*, *parent_name=None*, *context=None*, *section=None*, *translation=None*, *rsrc_description=None*)

Bases: Mapping

get_user_value(*key*)

static schema_from_params(*params_snippet*)

Create properties schema from the parameters section of a template.

Parameters

params_snippet parameter definition from a template

Returns

equivalent properties schemata for the specified parameters

classmethod schema_to_parameters_and_properties(*schema*, *template_type='cfn'*)

Convert a schema to template parameters and properties.

This can be used to generate a provider template that matches the given properties schemata.

Parameters

schema A resource types properties_schema

Returns
A tuple of params and properties dicts

ex: input: {foo: {Type: List}}

output: {foo: {Type: CommaDelimitedList}},
{foo: {Fn::Split: {Ref: foo}}}

ex: input: {foo: {Type: String}, bar: {Type: Map}}

output: {foo: {Type: String}, bar: {Type: Json}},
{foo: {Ref: foo}, bar: {Ref: bar}}

`update_translation`(*rules*, *client_resolve=True*, *ignore_resolve_error=False*)

`validate`(*with_value=True*)

`class heat.engine.properties.Property`(*schema*, *name=None*, *context=None*, *path=None*)

Bases: `object`

`default`()

`get_value`(*value*, *validate=False*, *translation=None*)

Get value from raw value and sanitize according to data type.

`has_default`()

`immutable`()

`implemented`()

`make_path`(*name*, *path=None*)

`required`()

`support_status`()

`type`()

`update_allowed`()

`class heat.engine.properties.Schema`(*data_type*, *description=None*, *default=None*, *schema=None*, *required=False*, *constraints=None*, *implemented=True*, *update_allowed=False*, *immutable=False*, *support_status=<heat.engine.support.SupportStatus object>*, *allow_conversion=False*)

Bases: *Schema*

Schema class for validating resource properties.

This class is used for defining schema constraints for resource properties. It inherits generic validation features from the base Schema class and add processing that is specific to resource properties.

`CONSTRAINTS = 'constraints'`

`DEFAULT = 'default'`

DESCRIPTION = 'description'

IMMUTABLE = 'immutable'

KEYS = ('type', 'description', 'default', 'schema', 'required', 'constraints', 'update_allowed', 'immutable')

REQUIRED = 'required'

SCHEMA = 'schema'

TYPE = 'type'

UPDATE_ALLOWED = 'update_allowed'

allowed_param_prop_type()

Return allowed type of Property Schema converted from parameter.

Especially, when generating Schema from parameter, Integer Property Schema will be supplied by Number parameter.

classmethod from_legacy(*schema_dict*)

Return a Property Schema object from a legacy schema dictionary.

classmethod from_parameter(*param*)

Return a Property Schema corresponding to a Parameter Schema.

Convert a parameter schema from a provider template to a property Schema for the corresponding resource facade.

validate(*context=None*)

Validates the schema.

This method checks if the schema itself is valid, and if the default value - if present - complies to the schemas constraints.

heat.engine.properties.**schemata**(*schema_dicts*)

Return dictionary of Schema objects for given dictionary of schemata.

The input schemata are converted from the legacy (dictionary-based) format to Schema objects where necessary.

heat.engine.properties_group module

class heat.engine.properties_group.**PropertiesGroup**(*schema, properties=None*)

Bases: object

A class for specifying properties relationships.

Properties group allows to specify relations between properties or other properties groups with operators AND, OR and XOR by one-key dict with list value. For example, if there are two properties: subprop1, which is child of property prop1, and property prop2, and they should not be specified together, then properties group for them should be next:

```
{XOR: [["prop1", "subprop1"], ["prop2"]]}
```

where each property name should be set as list of strings. Also, if these properties are exclusive with properties prop3 and prop4, which should be specified both, then properties group will be defined such way:

```
{XOR: [ ["prop1", "subprop1"], ["prop2"],
        {AND: [ ["prop3"], ["prop4"] ]} ]}
```

where one-key dict with key AND is nested properties group.

validate_schema(*current_schema*)

heat.engine.resource module

exception heat.engine.resource.**NoActionRequired**(*res_name='Unknown'*, *reason=''*)

Bases: Exception

Exception raised when a signal is ignored.

Resource subclasses should raise this exception from handle_signal() to suppress recording of an event corresponding to the signal.

exception heat.engine.resource.**PollDelay**(*period*)

Bases: Exception

Exception to delay polling of the resource.

This exception may be raised by a Resource subclasss check_*_complete() methods to indicate that it need not be polled again immediately. If this exception is raised, the check_*_complete() method will not be called again until the nth time that the resource becomes eligible for polling. A PollDelay period of 1 is equivalent to returning False.

class heat.engine.resource.**Resource**(*name*, *definition*, *stack*)

Bases: *ResourceStatus*

BASE_ATTRIBUTES = ('show',)

FnGetAtt(*key*, **path*)

For the intrinsic function Fn::GetAtt.

Parameters

- **key** the attribute key.
- **path** a list of path components to select from the attribute.

Returns
the attribute value.

FnGetRefId()

For the intrinsic function Ref.

Results
the id or name of the resource.

LOCK_ACQUIRE = 1

LOCK_ACTIONS = (None, 1, -1, 0)

LOCK_NONE = None

LOCK_RELEASE = -1

LOCK_RESPECT = 0

SHOW = 'show'

action_handler_task(*action*, *args=None*, *action_prefix=None*)

A task to call the Resource subclasss handler methods for action.

Calls the handle_<ACTION>() method for the given action and then calls the check_<ACTION>_complete() method with the result in a loop until it returns True. If the methods are not provided, the call is omitted.

Any args provided are passed to the handler.

If a prefix is supplied, the handler method handle_<PREFIX>_<ACTION>() is called instead.

add_dependencies(*deps*)

Add implicit dependencies specific to the resource type.

Some resource types may have implicit dependencies on other resources in the same stack that are not linked by a property value (that would be set using get_resource or get_attr for example, thus creating an explicit dependency). Such dependencies are opaque to the user and should be avoided wherever possible, however in some circumstances they are required due to magic in the underlying API.

The deps parameter is a Dependencies object to which dependency pairs may be added.

add_explicit_dependencies(*deps*)

Add all dependencies explicitly specified in the template.

The deps parameter is a Dependencies object to which dependency pairs are added.

adopt(*resource_data*)

Adopt the existing resource.

Resource subclasses can provide a handle_adopt() method to customise adopt.

always_replace_on_check_failed = True

attributes_schema = {}

base_attributes_schema = {'show': <heat.engine.attributes.Schema object>}

static build_template_dict(*res_name*, *res_type*, *tmpl_type*, *params*, *props*, *outputs*, *description*)

calc_update_allowed(*props*)

cancel_grace_period()

check()

Checks that the physical resource is in its expected state.

Gets the current status of the physical resource and updates the database accordingly. If check is not supported by the resource, default action is to fail and revert the resources status to its original state with the added message that check was not performed.

check_convergence(*engine_id*, *timeout*, *progress_callback=None*)

Check the resource synchronously.

classmethod check_is_substituted(*new_res_type*)

cinder()

clear_hook(*hook*)

clear_stored_attributes()

client(*name=None*, *version=None*)

client_plugin(*name=None*)

create()

Create the resource.

Subclasses should provide a handle_create() method to customise creation.

create_convergence(*template_id*, *requires*, *engine_id*, *timeout*, *progress_callback=None*)

Creates the resource by invoking the scheduler TaskRunner.

data()

Return the resource data for this resource.

Use methods data_set and data_delete to modify the resource data for this resource.

Returns
a dict representing the resource data for this resource.

data_delete(*key*)

Remove a key from the resource data.

Returns
True if the key existed to delete.

data_set(*key*, *value*, *redact=False*)

Set a key in the resource data.

default_client_name = None

delete()

A task to delete the resource.

Subclasses should provide a handle_delete() method to customise deletion.

delete_convergence(*template_id*, *engine_id*, *timeout*, *progress_callback=None*)

Destroys the resource if it doesnt belong to given template.

The given template is suppose to be the current template being provisioned.

Also, since this resource is visited as part of clean-up phase, the needed_by should be updated. If this resource was replaced by more recent resource, then delete this and update the replacement resources replaces field.

delete_snapshot(*data*)

delete_snapshot_convergence(*engine_id*, *timeout*, *progress_callback=None*)

delete snapshot for the resource synchronously.

destroy()

A task to delete the resource and remove it from the database.

entity = None

property external_id

frozen_definition()

Return a frozen ResourceDefinition with stored property values.

The returned definition will contain the property values read from the database, and will have all intrinsic functions resolved (note that this makes it useless for calculating dependencies).

frozen_properties()

Context manager to use the frozen property values from the database.

The live property values are always substituted back when the context ends.

get_attribute(*key*, **path*)

Default implementation for function get_attr and Fn::GetAtt.

This may be overridden by resource plugins to add extra logic specific to the resource implementation.

get_live_resource_data()

Default implementation; can be overridden by resources.

Get resource data and handle it with exceptions.

get_live_state(*resource_properties*)

Default implementation; should be overridden by resources.

Parameters

resource_properties resources object of Properties class.

Returns

dict of resources real state of properties.

get_nested_parameters_stack()

Return the nested stack for schema validation.

Regular resources dont have such a thing.

get_reference_id()

Default implementation for function get_resource.

This may be overridden by resource plugins to add extra logic specific to the resource implementation.

classmethod getdoc()

glance()

handle_adopt(*resource_data=None*)

handle_delete()

Default implementation; should be overridden by resources.

handle_metadata_reset()

Default implementation; should be overridden by resources.

Now we override this method to reset the metadata for scale-policy and scale-group resources, because their metadata might hang in a wrong state (scaling_in_progress is always True) if engine restarts while scaling.

handle_preempt()

Pre-empt an in-progress update when a new update is available.

This method is called when a previous convergence update is in progress but a new update for the resource is available. By default it does nothing, but subclasses may override it to cancel the in-progress update if it is safe to do so.

Note that this method does not run in the context of the in-progress update and has no access to runtime information about it; nor is it safe to make changes to the Resource in the database. If implemented, this method should cause the existing update to complete by external means. If this leaves the resource in a FAILED state, that should be taken into account in needs_replace_failed().

handle_update(*json_snippet*, *tmpl_diff*, *prop_diff*)

has_hook(*hook*)

has_interface(*resource_type*)

Check if resource is mapped to resource_type or is resource_type.

Check to see if this resource is either mapped to resource_type or is a resource_type.

has_nested()

Return True if the resource has an existing nested stack.

For most resource types, this will always return False. StackResource subclasses return True when appropriate. Resource subclasses that may return True must also provide a nested_identifier() method to return the identifier of the nested stack, and a nested() method to return a Stack object for the nested stack.

heat()

identifier()

Return an identifier for this resource.

classmethod is_service_available(*context*)

keystone()

classmethod load(*context*, *resource_id*, *current_traversal*, *is_update*, *data*)

Load a specified resource from the database to check.

Returns a tuple of the Resource, the StackDefinition corresponding to the resources ResourceDefinition (i.e. the one the resource was last updated to if it has already been created, or the one it will be created with if it hasnt been already), and the Stack containing the latest StackDefinition (i.e. the one that the latest traversal is updating to.

The latter two must remain in-scope, because the Resource holds weak references to them.

lock(*engine_id*)

make_replacement(*new_tmpl_id*, *requires*)

Create a replacement resource in the database.

Returns the DB ID of the new resource, or None if the new resource cannot be created (generally because the template ID does not exist). Raises UpdateInProgress if another traversal has already locked the current resource.

metadata_get(*refresh=False*)

metadata_set(*metadata*, *merge_metadata=None*)

Write new metadata to the database.

The caller may optionally provide a merge_metadata() function, which takes two arguments - the metadata passed to metadata_set() and the current metadata of the resource - and returns the merged metadata to write. If merge_metadata is not provided, the metadata passed to metadata_set() is written verbatim, overwriting any existing metadata.

If a race condition is detected, the write will be retried with the new result of merge_metadata() (if it is supplied) or the verbatim data (if it is not).

metadata_update(*new_metadata=None*)

No-op for resources which dont explicitly override this method.

needs_replace(*after_props*)

Mandatory replace based on certain properties.

needs_replace_failed()

Needs replace if resource is in `*_FAILED`.

needs_replace_with_prop_diff(*changed_properties_set*, *after_props*, *before_props*)

Needs replace based on prop_diff.

needs_replace_with_tmpl_diff(*tmpl_diff*)

Needs replace based on tmpl_diff.

neutron()

no_signal_actions = ('SUSPEND', 'DELETE')

node_data(*stk_defn=None*, *for_resources=True*, *for_outputs=False*)

Return a NodeData object representing the resource.

The NodeData object returned contains basic data about the resource, including its name, ID and state, as well as its reference ID and any attribute values that are used.

By default, those attribute values that are referenced by other resources are included. These can be ignored by setting the for_resources parameter to False. If the for_outputs parameter is True, those attribute values that are referenced by stack outputs are included. If the for_outputs parameter is an iterable of output names, only those attribute values referenced by the specified stack outputs are included.

The set of referenced attributes is calculated from the StackDefinition object provided, or from the stacks current definition if none is passed.

After calling this method, the resources attribute cache is populated with any cacheable attribute values referenced by stack outputs, even if they are not also referenced by other resources.

nova()

parse_live_resource_data(*resource_properties*, *resource_data*)

Default implementation; can be overridden by resources.

Parse resource data for using it in updating properties with live state. :param resource_properties: properties of stored resource plugin. :param resource_data: data from current live state of a resource.

static pause()

physical_resource_name()

physical_resource_name_limit = 255

physical_resource_name_or_FnGetRefId()

prepare_abandon()

prepare_for_replace()

Prepare resource for replacing.

Some resources requires additional actions before replace them. If resource need to be changed before replacing, this method should be implemented in resource class.

preview()

Default implementation of Resource.preview.

This method should be overridden by child classes for specific behavior.

preview_update(*after*, *before*, *after_props*, *before_props*, *prev_resource*, *check_init_complete=False*)

Simulates update without actually updating the resource.

Raises UpdateReplace, if replacement is required or returns True, if in-place update is required.

static reduce_physical_resource_name(*name*, *limit*)

Reduce length of physical resource name to a limit.

The reduced name will consist of the following:

- the first 2 characters of the name
- a hyphen
- the end of the name, truncated on the left to bring the name length within the limit

Parameters

- **name** The name to reduce the length of
- **limit** The max length limit

Returns

A name whose length is less than or equal to the limit

referenced_attrs(*stk_defn=None*, *in_resources=True*, *in_outputs=True*, *load_all=False*)

Return the set of all attributes referenced in the template.

This enables the resource to calculate which of its attributes will be used. By default, attributes referenced in either other resources or outputs will be included. Either can be excluded by setting the *in_resources* or *in_outputs* parameters to False. To limit to a subset of outputs, pass an iterable of the output names to examine for the *in_outputs* parameter.

The set of referenced attributes is calculated from the StackDefinition object provided, or from the stacks current definition if none is passed.

regenerate_info_schema(*definition*)

Default implementation; should be overridden by resources.

Should be overridden by resources that would require schema refresh during update, ex. TemplateResource.

Definition
Resource Definition

reparse(*client_resolve=True*)

Reparse the resource properties.

Optional translate flag for property translation and client_resolve flag for resolving properties by doing client lookup.

required_by()

List of resources that require this one as a dependency.

Returns a list of names of resources that depend on this resource directly.

required_service_extension = None

requires_deferred_auth = False

resource_id_set(*inst*)

resource_snapshot_set(*snapshot_id=None*)

classmethod resource_to_template(*resource_type*, *template_type='cfn'*)

Generate a provider template that mirrors the resource.

Parameters

- **resource_type** The resource type to be displayed in the template
- **template_type** the template type to generate, cfn or hot.

Returns
A template where the resources properties_schema is mapped as parameters, and the resources attributes_schema is mapped as outputs

restore_prev_rsrc(*convergence=False*)

Restore resource after rollback.

Some resources requires additional actions after rollback. If resource need to be changed during rollback, this method should be implemented in resource class.

resume()

Return a task to resume the resource.

Subclasses should provide a handle_resume() method to implement resume.

resume_convergence(*engine_id*, *timeout*, *progress_callback=None*)

resume the resource synchronously.

rpc_client()

Return a client for making engine RPC calls.

signal(*details=None*, *need_check=True*)

Signal the resource.

Returns True if the metadata for all resources in the stack needs to be regenerated as a result of the signal, False if it should not be.

Subclasses should provide a handle_signal() method to implement the signal. The base-class raise an exception if no handler is implemented.

signal_needs_metadata_updates = True

snapshot()

Snapshot the resource and return the created data, if any.

snapshot_convergence(*engine_id*, *timeout*, *progress_callback=None*)

snapshot the resource synchronously.

property stack

property state

Returns state, tuple of action, status.

state_reset()

Reset state to (INIT, COMPLETE).

state_set(*action*, *status*, *reason='state changed'*, *lock=None*)

store(*set_metadata=False*, *lock=None*)

Create the resource in the database.

If self.id is set, we update the existing stack.

store_attributes()

strict_dependency = True

support_status = <heat.engine.support.SupportStatus object>

suspend()

Return a task to suspend the resource.

Subclasses should provide a handle_suspend() method to implement suspend.

suspend_convergence(*engine_id*, *timeout*, *progress_callback=None*)

suspend the resource synchronously.

swift()

translate_properties(*properties*, *client_resolve=True*, *ignore_resolve_error=False*)

Set resource specific rules for properties translation.

The properties parameter is a properties object and the optional client_resolve flag is to specify whether to do RESOLVE translation with client lookup.

translation_rules(*properties*)

Return specified rules for resource.

trigger_hook(*hook*)

trove()

type()

update(*after*, *before=None*, *prev_resource=None*, *new_template_id=None*, *new_requires=None*)

Return a task to update the resource.

Subclasses should provide a handle_update() method to customise update, the base-class handle_update will fail by default.

update_allowed_properties = ()

update_convergence(*template_id*, *new_requires*, *engine_id*, *timeout*, *new_stack*, *progress_callback=None*)

Update the resource synchronously.

Persist the resources current_template_id to template_id and resources requires to list of the required resource ids from the given resource_data and existing resources requires, then updates the resource by invoking the scheduler TaskRunner.

update_policy_schema = {}

update_template_diff(*after*, *before*)

Returns the difference between the before and after json snippets.

If something has been removed in after which exists in before we set it to None.

update_template_diff_properties(*after_props*, *before_props*)

Return changed Properties between the before and after properties.

If any property having immutable as True is updated, raises NotSupported error. If any properties have changed which are not in update_allowed_properties, raises UpdateReplace.

validate()

Validate the resource.

This may be overridden by resource plugins to add extra validation logic specific to the resource implementation.

classmethod validate_deletion_policy(*policy*)

validate_external()

validate_template()

Validate structural/syntax aspects of the resource definition.

Resource plugins should not override this, because this interface is expected to be called pre-create so things normally valid in an overridden validate() such as accessing properties may not work.

heat.engine.rsrc_defn module

class heat.engine.rsrc_defn.**ResourceDefinition**(*name*, *resource_type*, *properties=None*, *metadata=None*, *depends=None*, *deletion_policy=None*, *update_policy=None*, *description=None*, *external_id=None*, *condition=None*)

Bases: object

A definition of a resource, independent of any template format.

DELETE = 'Delete'

DELETION_POLICIES = ('Delete', 'Retain', 'Snapshot')

class Diff(*old_defn*, *new_defn*)

Bases: object

A diff between two versions of the same resource definition.

metadata_changed()

Return True if the resource metadata has changed.

properties_changed()

Return True if the resource properties have changed.

update_policy_changed()

Return True if the resource update policy has changed.

RETAIN = 'Retain'

SNAPSHOT = 'Snapshot'

condition()

Return the name of the conditional inclusion rule, if any.

Returns None if the resource is included unconditionally.

deletion_policy()

Return the deletion policy for the resource.

The policy will be one of those listed in DELETION_POLICIES.

dep_attrs(*resource_name*, *load_all=False*)

Iterate over attributes of a given resource that this references.

Return an iterator over dependent attributes for specified resource_name in resources properties and metadata fields.

dependencies(*stack*)

Return the Resource objects in given stack on which this depends.

external_id()

Return the external resource id.

freeze(***overrides*)

Return a frozen resource definition, with all functions resolved.

This return a new resource definition with fixed data (containing no intrinsic functions). Named arguments passed to this method override the values passed as arguments to the constructor.

metadata()

Return the resource metadata.

properties(*schema*, *context=None*)

Return a Properties object representing the resource properties.

The Properties object is constructed from the given schema, and may require a context to validate constraints.

render_hot()

Return a HOT snippet for the resource definition.

reparse(*stack*, *template*)

Reinterpret the resource definition in the context of a new stack.

This returns a new resource definition, with all of the functions parsed in the context of the specified stack and template.

Any conditions are *not* included - it is assumed that the resource is being interpreted in any context that it should be enabled in that context.

required_resource_names()

Return a set of names of all resources on which this depends.

Note that this is done entirely in isolation from the rest of the template, so the resource names returned may refer to resources that dont actually exist, or would have strict_dependency=False. Use the dependencies() method to get validated dependencies.

set_translation_rules(*rules=None*, *client_resolve=True*)

Helper method to update properties with translation rules.

update_policy(*schema*, *context=None*)

Return a Properties object representing the resource update policy.

The Properties object is constructed from the given schema, and may require a context to validate constraints.

validate()

Validate intrinsic functions that appear in the definition.

heat.engine.scheduler module

class heat.engine.scheduler.**DependencyTaskGroup**(*dependencies*, *task=<function DependencyTaskGroup.<lambda>*, *reverse=False*, *name=None*, *error_wait_time=None*, *aggregate_exceptions=False*)

Bases: object

Task which manages group of subtasks that have ordering dependencies.

cancel_all(*grace_period=None*)

exception heat.engine.scheduler.**ExceptionGroup**(*exceptions=None*)

Bases: Exception

Container for multiple exceptions.

This exception is used by DependencyTaskGroup when the flag aggregate_exceptions is set to True and its re-raised again when all tasks are finished. This way it can be caught later on so that the individual exceptions can be acted upon.

class heat.engine.scheduler.**TaskRunner**(*task*, **args*, ***kwargs*)

Bases: object

Wrapper for a resumable task (co-routine).

as_task(*timeout=None*, *progress_callback=None*, *post_func=None*)

Return a task that drives the TaskRunner.

cancel(*grace_period=None*)

Cancel the task and mark it as done.

done()

Return True if the task is complete.

run_to_completion(*wait_time=1*, *progress_callback=None*)

Run the task to completion.

The task will sleep for *wait_time* seconds between steps. To avoid sleeping, pass *None* for *wait_time*.

start(*timeout=None*)

Initialise the task and run its first step.

If a timeout is specified, any attempt to step the task after that number of seconds has elapsed will result in a Timeout being raised inside the task.

started()

Return True if the task has been started.

step()

Run another step of the task.

Return True if the task is complete; False otherwise.

exception heat.engine.scheduler.**TimedCancel**(*task_runner*, *timeout*)

Bases: *Timeout*

trigger(*generator*)

Trigger the timeout on a given generator.

exception heat.engine.scheduler.**Timeout**(*task_runner*, *timeout*)

Bases: BaseException

Raised when task has exceeded its allotted (wallclock) running time.

This allows the task to perform any necessary cleanup, as well as use a different exception to notify the controlling task if appropriate. If the task suppresses the exception altogether, it will be cancelled but the controlling task will not be notified of the timeout.

earlier_than(*other*)

expired()

trigger(*generator*)

Trigger the timeout on a given generator.

heat.engine.scheduler.**task_description**(*task*)

Return a human-readable string description of a task.

The description is used to identify the task when logging its status.

heat.engine.service module

class heat.engine.service.**EngineListener**(*host*, *engine_id*, *thread_group_mgr*)

Bases: object

Listen on an AMQP queue named for the engine.

Allows individual engines to communicate with each other for multi-engine support.

ACTIONS = ('stop_stack', 'send')

SEND = 'send'

STOP_STACK = 'stop_stack'

listening(*ctxt*)

Respond to a watchdog request.

Respond affirmatively to confirm that the engine performing the action is still alive.

send(*ctxt*, *stack_identity*, *message*)

start()

stop()

stop_stack(*ctxt*, *stack_identity*)

Stop any active threads on a stack.

class heat.engine.service.**EngineService**(*host*, *topic*)

Bases: ServiceBase

Manages the running instances from creation to destruction.

All the methods in here are called from the RPC backend. This is all done dynamically so if a call is made via RPC that does not have a corresponding method here, an exception will be thrown when it attempts to call into this class. Arguments to these methods are also dynamically added and will be named as keyword arguments by the RPC caller.

RPC_API_VERSION = '1.36'

abandon_stack(*cnxt*, *stack_identity*, *abandon=True*)

Abandon a given stack.

Parameters

- **cnxt** RPC context.
- **stack_identity** Name of the stack you want to abandon.
- **abandon** Delete Heat stack but not physical resources.

authenticated_to_backend(*cnxt*)

Validate the credentials in the RPC context.

Verify that the credentials in the RPC context are valid for the current cloud backend.

check_software_deployment(*cnxt*, *deployment_id*, *timeout*)

count_stacks(*cnxt*, *filters=None*, *tenant_safe=True*, *show_deleted=False*, *show_nested=False*, *show_hidden=False*, *tags=None*, *tags_any=None*, *not_tags=None*, *not_tags_any=None*)

Return the number of stacks that match the given filters.

Parameters

- **cnxt** RPC context.
- **filters** a dict of ATTR:VALUE to match against stacks
- **tenant_safe** DEPRECATED, if true, scope the request by the current tenant
- **show_deleted** if true, count will include the deleted stacks
- **show_nested** if true, count will include nested stacks
- **show_hidden** if true, count will include hidden stacks
- **tags** count stacks containing these tags. If multiple tags are passed, they will be combined using the boolean AND expression
- **tags_any** count stacks containing these tags. If multiple tags are passed, they will be combined using the boolean OR expression
- **not_tags** count stacks not containing these tags. If multiple tags are passed, they will be combined using the boolean AND expression
- **not_tags_any** count stacks not containing these tags. If multiple tags are passed, they will be combined using the boolean OR expression

Returns

an integer representing the number of matched stacks

create_software_config(*cnxt*, *group*, *name*, *config*, *inputs*, *outputs*, *options*)

create_software_deployment(*cnxt*, *server_id*, *config_id*, *input_values*, *action*, *status*, *status_reason*, *stack_user_project_id*, *deployment_id=None*)

create_stack(*cnxt*, *stack_name*, *template*, *params*, *files*, *args*, *environment_files=None*, *files_container=None*, *owner_id=None*, *nested_depth=0*, *user_creds_id=None*, *stack_user_project_id=None*, *parent_resource_name=None*, *template_id=None*)

Create a new stack using the template provided.

Note that at this stage the template has already been fetched from the heat-api process if using a template-url.

Parameters

- **cnxt** RPC context.
- **stack_name** Name of the stack you want to create.
- **template** Template of stack you want to create.
- **params** Stack Input Params
- **files** Files referenced from the template
- **args** Request parameters/args passed from API
- **environment_files** (*list or None*) optional ordered list of environment file names included in the files dict
- **files_container** optional swift container name
- **owner_id** parent stack ID for nested stacks, only expected when called from another heat-engine (not a user option)
- **nested_depth** the nested depth for nested stacks, only expected when called from another heat-engine
- **user_creds_id** the parent user_creds record for nested stacks
- **stack_user_project_id** the parent stack_user_project_id for nested stacks
- **parent_resource_name** the parent resource name
- **template_id** the ID of a pre-stored template in the DB

delete_snapshot(*cnxt*, *stack_identity*, *snapshot_id*)

delete_software_config(*cnxt*, *config_id*)

delete_software_deployment(*cnxt*, *deployment_id*)

delete_stack(*cnxt*, *stack_identity*)

Delete a given stack.

Parameters

- **cnxt** RPC context.
- **stack_identity** Name of the stack you want to delete.

describe_stack_resource(*cnxt*, *stack_identity*, *resource_name*, *with_attr=None*)

describe_stack_resources(*cnxt*, *stack_identity*, *resource_name*)

export_stack(*cnxt*, *stack_identity*)

Exports the stack data json.

Intended to be used to safely retrieve the stack data before performing the abandon action.

Parameters

- **cnxt** RPC context.
- **stack_identity** Name of the stack you want to export.

find_physical_resource(*cnxt*, *physical_resource_id*)

Return an identifier for the specified resource.

Parameters

- **cnxt** RPC context.
- **physical_resource_id** The physical resource ID to look up.

generate_template(*cnxt*, *type_name*, *template_type='cfn'*)

Generate a template based on the specified type.

Parameters

- **cnxt** RPC context.
- **type_name** Name of the resource type to generate a template for.
- **template_type** the template type to generate, cfn or hot.

get_environment(*cnxt*, *stack_identity*)

Returns the environment for an existing stack.

Parameters

- **cnxt** RPC context
- **stack_identity** identifies the stack

Return type
dict

get_files(*cnxt*, *stack_identity*)

Returns the files for an existing stack.

Parameters

- **cnxt** RPC context
- **stack_identity** identifies the stack

Return type
dict

get_revision(*cnxt*)

get_template(*cnxt*, *stack_identity*)

Get the template.

Parameters

- **cnxt** RPC context.
- **stack_identity** Name of the stack you want to see.

identify_stack(*cnxt*, *stack_name*)

The full stack identifier for a single, live stack with stack_name.

Parameters

- **cnxt** RPC context.
- **stack_name** Name or UUID of the stack to look up.

list_events(*cnxt*, *stack_identity*, *filters=None*, *limit=None*, *marker=None*, *sort_keys=None*, *sort_dir=None*, *nested_depth=None*)

Lists all events associated with a given stack.

It supports pagination (`limit` and `marker`), sorting (`sort_keys` and `sort_dir`) and filtering(filters) of the results.

Parameters

- **cnxt** RPC context.
- **stack_identity** Name of the stack you want to get events for
- **filters** a dict with attribute:value to filter the list
- **limit** the number of events to list (integer or string)
- **marker** the ID of the last event in the previous page
- **sort_keys** an array of fields used to sort the list
- **sort_dir** the direction of the sort (asc or desc).
- **nested_depth** Levels of nested stacks to list events for.

list_outputs(*cntx*, *stack_identity*)

Get a list of stack outputs.

Parameters

- **cntx** RPC context.
- **stack_identity** Name of the stack you want to see.

Returns

list of stack outputs in defined format.

list_resource_types(*cnxt*, *support_status=None*, *type_name=None*, *heat_version=None*, *with_description=False*)

Get a list of supported resource types.

Parameters

- **cnxt** RPC context.
- **support_status** Support status of resource type

- **type_name** Resource types name (regular expression allowed)
- **heat_version** Heat version
- **with_description** Either return resource type description or not

list_services(*cnxt*)

list_software_configs(*cnxt*, *limit=None*, *marker=None*, *tenant_safe=True*)

list_software_deployments(*cnxt*, *server_id*)

list_stack_resources(*cnxt*, *stack_identity*, *nested_depth=0*, *with_detail=False*, *filters=None*)

list_stacks(*cnxt*, *limit=None*, *marker=None*, *sort_keys=None*, *sort_dir=None*, *filters=None*, *tenant_safe=True*, *show_deleted=False*, *show_nested=False*, *show_hidden=False*, *tags=None*, *tags_any=None*, *not_tags=None*, *not_tags_any=None*)

Returns attributes of all stacks.

It supports pagination (`limit` and `marker`), sorting (`sort_keys` and `sort_dir`) and filtering (`filters`) of the results.

Parameters

- **cnxt** RPC context
- **limit** the number of stacks to list (integer or string)
- **marker** the ID of the last item in the previous page
- **sort_keys** an array of fields used to sort the list
- **sort_dir** the direction of the sort (asc or desc)
- **filters** a dict with attribute:value to filter the list
- **tenant_safe** DEPRECATED, if true, scope the request by the current tenant
- **show_deleted** if true, show soft-deleted stacks
- **show_nested** if true, show nested stacks
- **show_hidden** if true, show hidden stacks
- **tags** show stacks containing these tags. If multiple tags are passed, they will be combined using the boolean AND expression
- **tags_any** show stacks containing these tags. If multiple tags are passed, they will be combined using the boolean OR expression
- **not_tags** show stacks not containing these tags. If multiple tags are passed, they will be combined using the boolean AND expression
- **not_tags_any** show stacks not containing these tags. If multiple tags are passed, they will be combined using the boolean OR expression

Returns

a list of formatted stacks

list_template_functions(*cnxt*, *template_version*, *with_condition=False*)

list_template_versions(*cnxt*)

metadata_software_deployments(*cnxt*, *server_id*)

migrate_convergence_1(*ctxt*, *stack_id*)

preview_stack(*cnxt*, *stack_name*, *template*, *params*, *files*, *args*, *environment_files=None*, *files_container=None*)

Simulate a new stack using the provided template.

Note that at this stage the template has already been fetched from the heat-api process if using a template-url.

Parameters

- **cnxt** RPC context.
- **stack_name** Name of the stack you want to create.
- **template** Template of stack you want to create.
- **params** Stack Input Params
- **files** Files referenced from the template
- **args** Request parameters/args passed from API
- **environment_files** (*list or None*) optional ordered list of environment file names included in the files dict
- **files_container** optional swift container name

preview_update_stack(*cnxt*, *stack_identity*, *template*, *params*, *files*, *args*, *environment_files=None*, *files_container=None*)

Shows the resources that would be updated.

The preview_update_stack method shows the resources that would be changed with an update to an existing stack based on the provided template and parameters. See update_stack for description of parameters.

This method *cannot* guarantee that an update will have the actions specified because resource plugins can influence changes/replacements at runtime.

Note that at this stage the template has already been fetched from the heat-api process if using a template-url.

reset()

Reset service.

Called in case a service running in daemon mode receives SIGHUP.

reset_stack_status()

resource_mark_unhealthy(*cnxt*, *stack_identity*, *resource_name*, *mark_unhealthy*, *resource_status_reason=None*)

Mark the resource as healthy or unhealthy.

Put the resource in CHECK_FAILED state if mark_unhealthy is true. Put the resource in CHECK_COMPLETE if mark_unhealthy is false and the resource is in CHECK_FAILED state. Otherwise, make no change.

Parameters

- **resource_name** either the logical name of the resource or the physical resource ID.
- **mark_unhealthy** indicates whether the resource is unhealthy.
- **resource_status_reason** reason for health change.

resource_schema(*cnxt*, *type_name*, *with_description=False*)

Return the schema of the specified type.

Parameters

- **cnxt** RPC context.
- **type_name** Name of the resource type to obtain the schema of.
- **with_description** Return result with description or not.

resource_signal(*cnxt*, *stack_identity*, *resource_name*, *details*, *sync_call=False*)

Calls resources signal for the specified resource.

Parameters

sync_call indicates whether a synchronized call behavior is expected. This is reserved for CFN WaitCondition implementation.

service_manage_cleanup()

service_manage_report()

show_output(*cntx*, *stack_identity*, *output_key*)

Returns dict with specified output key, value and description.

Parameters

- **cntx** RPC context.
- **stack_identity** Name of the stack you want to see.
- **output_key** key of desired stack output.

Returns

dict with output key, value and description in defined format.

show_snapshot(*cnxt*, *stack_identity*, *snapshot_id*)

show_software_config(*cnxt*, *config_id*)

show_software_deployment(*cnxt*, *deployment_id*)

show_stack(*cnxt*, *stack_identity*, *resolve_outputs=True*)

Return detailed information about one or all stacks.

Parameters

- **cnxt** RPC context.

- **stack_identity** Name of the stack you want to show, or None to show all
- **resolve_outputs** If True, outputs for given stack/stacks will be resolved

signal_software_deployment(*cnxt*, *deployment_id*, *details*, *updated_at*)

stack_cancel_update(*cnxt*, *stack_identity*, *cancel_with_rollback=True*)

Cancel currently running stack update.

Parameters

- **cnxt** RPC context.
- **stack_identity** Name of the stack for which to cancel update.
- **cancel_with_rollback** Force rollback when cancel update.

stack_check(*cnxt*, *stack_identity*)

Handle request to perform a check action on a stack.

stack_list_snapshots(*cnxt*, *stack_identity*)

stack_restore(*cnxt*, *stack_identity*, *snapshot_id*)

stack_resume(*cnxt*, *stack_identity*)

Handle request to perform a resume action on a stack.

stack_snapshot(*cnxt*, *stack_identity*, *name*)

stack_suspend(*cnxt*, *stack_identity*)

Handle request to perform suspend action on a stack.

start()

Start service.

stop()

Stop service.

update_software_deployment(*cnxt*, *deployment_id*, *config_id*, *input_values*, *output_values*, *action*, *status*, *status_reason*, *updated_at*)

update_stack(*cnxt*, *stack_identity*, *template*, *params*, *files*, *args*, *environment_files=None*, *files_container=None*, *template_id=None*)

Update an existing stack based on the provided template and params.

Note that at this stage the template has already been fetched from the heat-api process if using a template-url.

Parameters

- **cnxt** RPC context.
- **stack_identity** Name of the stack you want to create.
- **template** Template of stack you want to create.
- **params** Stack Input Params
- **files** Files referenced from the template
- **args** Request parameters/args passed from API

- **environment_files** (*list or None*) optional ordered list of environment file names included in the files dict
- **files_container** optional swift container name
- **template_id** the ID of a pre-stored template in the DB

validate_template(*cnxt*, *template*, *params=None*, *files=None*, *environment_files=None*, *files_container=None*, *show_nested=False*, *ignorable_errors=None*)

Check the validity of a template.

Checks, so far as we can, that a template is valid, and returns information about the parameters suitable for producing a user interface through which to specify the parameter values.

Parameters

- **cnxt** RPC context.
- **template** Template of stack you want to create.
- **params** Stack Input Params
- **files** Files referenced from the template
- **environment_files** (*list or None*) optional ordered list of environment file names included in the files dict
- **files_container** optional swift container name
- **show_nested** if True, any nested templates will be checked
- **ignorable_errors** List of error_code to be ignored as part of validation

wait()

Wait for service to complete.

class heat.engine.service.**NotifyEvent**

Bases: object

signal()

Signal the event.

signalled()

wait(*timeout=None*)

Wait for the event with optional timeout.

class heat.engine.service.**ThreadGroup**(*thread_pool_size=10*)

Bases: ThreadGroup

add_thread(*func*, **args*, ***kwargs*)

Start a new thread in the group.

class heat.engine.service.**ThreadGroupManager**

Bases: object

add_msg_queue(*stack_id*, *msg_queue*)

add_timer(*stack_id*, *func*, **args*, ***kwargs*)

Define a periodic task in the stack threadgroups.

The task is run in a separate thread.

Periodicity is cfg.CONF.periodic_interval

remove_msg_queue(*gt*, *stack_id*, *msg_queue*)

send(*stack_id*, *message*)

start(*stack_id*, *func*, **args*, ***kwargs*)

Run the given method in a sub-thread.

start_with_acquired_lock(*stack*, *lock*, *func*, **args*, ***kwargs*)

Run the given method in a sub-thread with an existing stack lock.

Release the provided lock when the thread finishes.

Parameters

- **stack** (*heat.engine.parser.Stack*) Stack to be operated on
- **lock** (heat.engine.stack_lock.StackLock) The acquired stack lock
- **func** (*function or instancemethod*) Callable to be invoked in sub-thread
- **args** Args to be passed to func
- **kwargs** Keyword-args to be passed to func

start_with_lock(*cnxt*, *stack*, *engine_id*, *func*, **args*, ***kwargs*)

Run the method in sub-thread after acquiring the stack lock.

Release the lock when the thread finishes.

Parameters

- **cnxt** RPC context
- **stack** (*heat.engine.parser.Stack*) Stack to be operated on
- **engine_id** The UUID of the engine/worker acquiring the lock
- **func** (*function or instancemethod*) Callable to be invoked in sub-thread
- **args** Args to be passed to func
- **kwargs** Keyword-args to be passed to func.

stop(*stack_id*, *graceful=False*)

Stop any active threads on a stack.

stop_timers(*stack_id*)

stopall()

class heat.engine.service.**ThreadWithCallback**(*target*, *args=()*, *kwargs=None*)

Bases: Thread

Thread that supports callback functions on completion.

link(*callback*, **args*, ***kwargs*)

Add a callback to be executed when thread completes.

run()

Method representing the threads activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the objects constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

heat.engine.service_software_config module

class heat.engine.service_software_config.**SoftwareConfigService**

Bases: object

check_software_deployment(*cnxt*, *deployment_id*, *timeout*)

count_software_config(*cnxt*)

count_software_deployment(*cnxt*)

create_software_config(*cnxt*, *group*, *name*, *config*, *inputs*, *outputs*, *options*)

create_software_deployment(*cnxt*, *server_id*, *config_id*, *input_values*, *action*, *status*, *status_reason*, *stack_user_project_id*, *deployment_id=None*)

delete_software_config(*cnxt*, *config_id*)

delete_software_deployment(*cnxt*, *deployment_id*)

list_software_configs(*cnxt*, *limit=None*, *marker=None*)

list_software_deployments(*cnxt*, *server_id*)

metadata_software_deployments(*cnxt*, *server_id*)

show_software_config(*cnxt*, *config_id*)

show_software_deployment(*cnxt*, *deployment_id*)

signal_software_deployment(*cnxt*, *deployment_id*, *details*, *updated_at*)

update_software_deployment(*cnxt*, *deployment_id*, *config_id*, *input_values*, *output_values*, *action*, *status*, *status_reason*, *updated_at*)

heat.engine.snapshots module

class heat.engine.snapshots.**Snapshot**(*context*, *snapshot_id*, *stack_id*, *start_time*, *thread_group_mgr*, *resources=None*, *action=None*, *timeout_mins=None*, *is_stack_delete=False*, *current_traversal=None*)

Bases: `object`

Operate snapshot actions under convergence.

ACTIONS = 'delete_snapshot'

DELETE_SNAPSHOT = 'delete_snapshot'

delete_snapshot()

delete_snapshot_objs()

do_delete_snapshot()

has_timed_out()

Returns True if this Snapshot has timed-out.

mark_complete(*predecessors=None*, *input_data=None*)

mark_failed(*rsrc_name*, *failure_reason*)

time_elapsed()

Time elapsed in seconds since the Snapshot operation started.

time_remaining()

Time left before Snapshot times out.

timeout_secs()

Return the action timeout in seconds.

property worker_client

Return a client for making engine RPC calls.

heat.engine.snapshots.**delete_snapshots**(*context*, *snapshot_ids*, *stack_id*, *current_traversal*, *start_time*, *thread_group_mgr*, *run_till_success=True*)

heat.engine.software_config_io module

APIs for dealing with input and output definitions for Software Configurations.

class heat.engine.software_config_io.**IOConfig**(***config*)

Bases: `object`

Base class for the configuration data for a single input or output.

as_dict()

Return a dict representation suitable for persisting.

name()

Return the name of the input or output.

class heat.engine.software_config_io.**InputConfig**(*value=<object object>*, ***config*)

Bases: *IOConfig*

Class representing the configuration data for a single input.

as_dict()

Return a dict representation suitable for persisting.

default()

Return the default value of the input.

input_data()

Return a name, value pair for the input.

replace_on_change()

schema = {'default': <heat.engine.properties.Schema object>, 'description': <heat.engine.properties.Schema object>, 'name': <heat.engine.properties.Schema object>, 'replace_on_change': <heat.engine.properties.Schema object>, 'type': <heat.engine.properties.Schema object>}

class heat.engine.software_config_io.**OutputConfig**(***config*)

Bases: *IOConfig*

Class representing the configuration data for a single output.

error_output()

Return True if the presence of the output indicates an error.

schema = {'description': <heat.engine.properties.Schema object>, 'error_output': <heat.engine.properties.Schema object>, 'name': <heat.engine.properties.Schema object>, 'type': <heat.engine.properties.Schema object>}

heat.engine.software_config_io.**check_io_schema_list**(*io_configs*)

Check that an input or output schema list is of the correct type.

Raises TypeError if the list itself is not a list, or if any of the members are not dicts.

heat.engine.stack module

class heat.engine.stack.**ConvergenceNode**(*rsrc_id*, *is_update*, *node_type='resource'*)

Bases: *ConvergenceNode*

Represents a node in the convergence dependency graph.

rsrc_id: Resource ID or Snapshot ID is_update: True for create/update, False for cleanup/delete node_type: resource (default) or snapshot

classmethod from_tuple(*t*)

Create ConvergenceNode from a tuple (2 or 3 elements).

to_tuple()

Convert to tuple for serialization.

exception heat.engine.stack.**ForcedCancel**(*with_rollback=True*)

Bases: Exception

Exception raised to cancel task execution.

class heat.engine.stack.**Stack**(*context*, *stack_name*, *tmpl*, *stack_id=None*, *action=None*, *status=None*, *status_reason=''*, *timeout_mins=None*, *disable_rollback=True*, *parent_resource=None*, *owner_id=None*, *adopt_stack_data=None*, *stack_user_project_id=None*, *created_time=None*, *updated_time=None*, *user_creds_id=None*, *tenant_id=None*, *use_stored_context=False*, *username=None*, *nested_depth=0*, *strict_validate=True*, *convergence=False*, *current_traversal=None*, *tags=None*, *prev_raw_template_id=None*, *current_deps=None*, *cache_data=None*, *deleted_time=None*, *converge=False*, *refresh_cred=False*)

Bases: Mapping

ACTIONS = ('CREATE', 'DELETE', 'UPDATE', 'ROLLBACK', 'SUSPEND', 'RESUME', 'ADOPT', 'SNAPSHOT', 'CHECK', 'RESTORE')

ADOPT = 'ADOPT'

CHECK = 'CHECK'

COMPLETE = 'COMPLETE'

CREATE = 'CREATE'

DELETE = 'DELETE'

FAILED = 'FAILED'

IN_PROGRESS = 'IN_PROGRESS'

RESTORE = 'RESTORE'

RESUME = 'RESUME'

ROLLBACK = 'ROLLBACK'

SNAPSHOT = 'SNAPSHOT'

STATUSES = ('IN_PROGRESS', 'FAILED', 'COMPLETE')

SUSPEND = 'SUSPEND'

UPDATE = 'UPDATE'

access_allowed(*credential_id*, *resource_name*)

Is credential_id authorised to access resource by resource_name.

add_resource(*resource*)

Insert the given resource into the stack.

adopt()

Adopt existing resources into a new stack.

check(*notify=None*)

converge_stack(*template*, *action='UPDATE'*, *new_stack=None*, *pre_converge=None*, *new_traversal_id=None*)

Update the stack template and trigger convergence for resources.

property convergence_dependencies

create(*msg_queue=None*)

Create the stack and all of the resources.

create_stack_user_project_id()

db_active_resources_get()

db_resource_get(*name*)

defer_state_persist()

Return whether to defer persisting the state.

If persistence is deferred, the new state will not be written to the database until the stack lock is released (by calling persist_state_and_release_lock()). This prevents races in the legacy path where an observer sees the stack COMPLETE but an engine still holds the lock.

delete(*action='DELETE'*, *backup=False*, *abandon=False*, *notify=None*)

Delete all of the resources, and then the stack itself.

The action parameter is used to differentiate between a user initiated delete and an automatic stack rollback after a failed create, which amount to the same thing, but the states are recorded differently.

Note abandon is a delete where all resources have been set to a RETAIN deletion policy, but we also dont want to delete anything required for those resources, e.g the stack_user_project.

delete_all_snapshots(*run_till_success=True*)

Remove all snapshots for this stack.

delete_snapshot(*snapshot*)

Remove a snapshot from the backends.

property dependencies

dependent_resource_ids(*resource_id*)

Return a set of resource IDs that are dependent on another.

Given a resource ID, return a set of all other resource IDs that are dependent on that one - that is to say, those that must be cleaned up before the given resource is cleaned up.

dispatch_event(*ev*)

property env

The stack environment

get_availability_zones()

get_kwargs_for_cloning(*keep_status=False*, *only_db=False*, *keep_tags=False*)

Get common kwargs for calling Stack() for cloning.

The point of this method is to reduce the number of places that we need to update when a kwarg to Stack.__init__() is modified. It is otherwise easy to forget an option and cause some unexpected error if this option is lost.

Note:

- This doesnt return the args(name, template) but only the kwargs.
- We often want to start fresh so dont want to maintain the old status, action and status_reason.
- We sometimes only want the DB attributes.

get_nested_parameters(*filter_func*)

Return nested parameters schema, if any.

This introspects the resources to return the parameters of the nested stacks. It uses the *get_nested_parameters_stack* API to build the stack.

has_timed_out()

Returns True if this stack has timed-out.

identifier()

Return an identifier for this stack.

iter_resources(*nested_depth=0*, *filters=None*)

Iterates over all the resources in a stack.

Iterating includes nested stacks up to *nested_depth* levels below.

classmethod load(*context*, *stack_id=None*, *stack=None*, *show_deleted=True*, *use_stored_context=False*, *force_reload=False*, *cache_data=None*, *load_template=True*, *check_refresh_cred=False*)

Retrieve a Stack from the database.

classmethod load_all(*context*, *limit=None*, *marker=None*, *sort_keys=None*, *sort_dir=None*, *filters=None*, *show_deleted=False*, *show_nested=False*, *show_hidden=False*, *tags=None*, *tags_any=None*, *not_tags=None*, *not_tags_any=None*)

mark_complete()

Mark the convergence update as complete.

mark_failed(*failure_reason*)

Mark the convergence update as failed.

migrate_to_convergence()

object_path_in_stack()

Return stack resources and stacks in path from the root stack.

If this is not nested return (None, self), else return stack resources and stacks in path from the root stack and including this stack.

Note that this is horribly inefficient, as it requires us to load every stack in the chain back to the root in memory at the same time.

Returns
a list of (stack_resource, stack) tuples.

property outputs

property parameters

property parent_resource

Dynamically load up the parent_resource.

Note: this should only be used by Fn::ResourceFacade

property parent_resource_name

path_in_stack()

Return tuples of names in path from the root stack.

If this is not nested return (None, self.name), else return tuples of names (stack_resource.name, stack.name) in path from the root stack and including this stack.

Returns
a list of (string, string) tuples.

persist_state_and_release_lock(*engine_id*)

Persist stack state to database and release stack lock

prepare_abandon(*no_resources=False*)

preview_resources()

Preview the stack with all of the resources.

purge_db()

Cleanup database after stack has completed/failed.

1. Delete the resources from DB.
2. If the stack failed, update the current_traversal to empty string so that the resource workers bail out.
3. Delete previous raw template if stack completes successfully.
4. Deletes all sync points. They are no longer needed after stack has completed/failed.
5. Delete the stack if the action is DELETE.

register_access_allowed_handler(*credential_id*, *handler*)

Register an authorization handler function.

Register a function which determines whether the credentials with a given ID can have access to a named resource.

remove_resource(*resource_name*)

Remove the resource with the specified name.

requires_deferred_auth()

Determine whether to perform API requests with deferred auth.

Returns whether this stack may need to perform API requests during its lifecycle using the configured deferred authentication method.

reset_dependencies()

reset_stack_and_resources_in_progress(*reason*)

resource_by_refid(*refid*)

Return the resource in this stack with the specified refid.

Returns
resource in this stack with the specified refid, or None if not found.

resource_get(*name*)

Return a stack resource, even if not in the current template.

property resources

restore(*snapshot*, *notify=None*)

Restore the given snapshot.

Invokes handle_restore on all resources.

restore_data(*snapshot*)

resume(*notify=None*)

Resume the stack.

Invokes handle_resume for all stack resources.

Waits for all resources to become RESUME_COMPLETE then declares the stack RESUME_COMPLETE. Note the default implementation for all resources is to do nothing other than move to RESUME_COMPLETE, so the resources must implement handle_resume for this to have any effect.

rollback()

root_stack_id()

set_parent_stack(*parent_stack*)

set_stack_user_project_id(*project_id*)

snapshot(*save_snapshot_func*)

Snapshot the stack, invoking handle_snapshot on all resources.

stack_task(*action*, *reverse=False*, *post_func=None*, *aggregate_exceptions=False*, *pre_completion_func=None*, *notify=None*)

A task to perform an action on the stack.

All of the resources are traversed in forward or reverse dependency order.

Parameters

- **action** action that should be executed with stack resources
- **reverse** define if action on the resources need to be executed in reverse dependency order
- **post_func** function that need to be executed after action complete on the stack
- **aggregate_exceptions** define if exceptions should be aggregated

- **pre_completion_func** function that need to be executed right before action completion; uses stack, action, status and reason as input parameters

property state

Returns state, tuple of action, status.

state_set(*action*, *status*, *reason*)

Update the stack state.

store(*backup=False*, *exp_trvsl=None*, *ignore_traversal_check=False*, *use_stored_context=False*)

Store the stack in the database and return its ID.

If self.id is set, we update the existing stack.

stored_context()

supports_check_action()

suspend(*notify=None*)

Suspend the stack.

Invokes handle_suspend for all stack resources.

Waits for all resources to become SUSPEND_COMPLETE then declares the stack SUSPEND_COMPLETE. Note the default implementation for all resources is to do nothing other than move to SUSPEND_COMPLETE, so the resources must implement handle_suspend for this to have any effect.

property t

The stack template.

property tags

time_elapsed()

Time elapsed in seconds since the stack operation started.

time_remaining()

Time left before stack times out.

timeout_secs()

Return the stack action timeout in seconds.

total_resources(*stack_id=None*)

Return the total number of resources in a stack.

Includes nested stacks below.

update(*newstack*, *msg_queue=None*, *notify=None*)

Update the stack.

Compare the current stack with newstack, and where necessary create/update/delete the resources until this stack aligns with newstack.

Note update of existing stack resources depends on update being implemented in the underlying resource types

Update will fail if it exceeds the specified timeout. The default is 60 minutes, set in the constructor

update_snapshot(*reason*, *status*)

update_task(*newstack*, *action='UPDATE'*, *msg_queue=None*, *notify=None*)

validate(*ignorable_errors=None*, *validate_res_tmpl_only=False*)

Validates the stack.

property worker_client

Return a client for making engine RPC calls.

heat.engine.stack.**reset_state_on_error**(*func*)

heat.engine.stack_lock module

class heat.engine.stack_lock.**StackLock**(*context*, *stack_id*, *engine_id*)

Bases: object

acquire(*retry=True*)

Acquire a lock on the stack.

Parameters

retry (*boolean*) When True, retry if lock was released while stealing.

get_engine_id()

Return the ID of the engine which currently holds the lock.

Returns None if there is no lock held on the stack.

release()

Release a stack lock.

thread_lock(*retry=True*)

Acquire a lock and release it only if there is an exception.

The release method still needs to be scheduled to be run at the end of the thread using the Thread.link method.

Parameters

retry (*boolean*) When True, retry if lock was released while stealing.

try_acquire()

Try to acquire a stack lock.

Dont raise an ActionInProgress exception or try to steal lock.

try_thread_lock()

Similar to thread_lock, but acquire the lock using try_acquire.

Only release it upon any exception after a successful acquisition.

heat.engine.status module

class heat.engine.status.**ResourceStatus**

Bases: object

ACTIONS = ('INIT', 'CREATE', 'DELETE', 'UPDATE', 'ROLLBACK', 'SUSPEND', 'RESUME', 'ADOPT', 'SNAPSHOT', 'CHECK')

ADOPT = 'ADOPT'

CHECK = 'CHECK'

COMPLETE = 'COMPLETE'

CREATE = 'CREATE'

DELETE = 'DELETE'

FAILED = 'FAILED'

INIT = 'INIT'

IN_PROGRESS = 'IN_PROGRESS'

RESUME = 'RESUME'

ROLLBACK = 'ROLLBACK'

SNAPSHOT = 'SNAPSHOT'

STATUSES = ('IN_PROGRESS', 'FAILED', 'COMPLETE')

SUSPEND = 'SUSPEND'

UPDATE = 'UPDATE'

heat.engine.stk_defn module

class heat.engine.stk_defn.**ResourceProxy**(*name*, *definition*, *resource_data*)

Bases: *ResourceStatus*

A lightweight API for essential data about a resource.

This is the interface through which template functions will access data about particular resources in the stack definition, such as the resource definition and current values of reference IDs and attributes.

Resource proxies for some or all resources in the stack will potentially be loaded for every check resource operation, so it is essential that this API is implemented efficiently, using only the data received over RPC and without reference to the resource data stored in the database.

This API can be considered stable by third-party Template or Function plugins, and no part of it should be changed or removed without an appropriate deprecation process.

FnGetAtt(*attr*, **path*)

For the intrinsic function get_attr.

FnGetAtts()

For the intrinsic function get_attr when getting all attributes.

Returns

a dict of all of the resources attribute values, excluding the show attribute.

FnGetRefId()

For the intrinsic function get_resource.

property action

The current action of the resource.

property attributes_schema

A set of the valid top-level attribute names.

This is provided for backwards-compatibility for functions that require a container with all of the valid attribute names in order to validate the template. Other operations on it are invalid because we dont actually have access to the attributes schema here; hence we return a set instead of a dict.

property external_id

The external ID of the resource.

name

property state

The current state (action, status) of the resource.

property status

The current status of the resource.

property t

The resource definition.

class heat.engine.stk_defn.**StackDefinition**(*context*, *template*, *stack_identifier*, *resource_data*, *parent_info=None*)

Bases: object

Class representing the definition of a Stack, but not its current state.

This is the interface through which template functions will access data about the stack definition, including the template and current values of resource reference IDs and attributes.

This API can be considered stable by third-party Template or Function plugins, and no part of it should be changed or removed without an appropriate deprecation process.

all_resource_types()

Return the set of types of all resources in the template.

all_rsrc_names()

Return the set of names of all resources in the template.

This includes resources that are disabled due to false conditionals.

clone_with_new_template(*new_template*, *stack_identifier*, *clear_resource_data=False*)

Create a new StackDefinition with a different template.

enabled_output_names()

Return the set of names of all enabled outputs in the template.

enabled_rsrc_names()

Return the set of names of all enabled resources in the template.

property env

The stacks environment.

get_availability_zones()

Return the list of Nova availability zones.

output_definition(*output_name*)

Return the definition of the given output.

property parent_resource

Return a proxy for the parent resource.

Returns None if the stack is not a provider stack for a TemplateResource.

resource_definition(*resource_name*)

Return the definition of the given resource.

property t

The stacks template.

heat.engine.stk_defn.**add_resource**(*stack_definition*, *resource_definition*)

Insert the given resource definition into the stack definition.

Add the resource to the template and store any temporary data.

heat.engine.stk_defn.**remove_resource**(*stack_definition*, *resource_name*)

Remove the named resource from the stack definition.

Remove the resource from the template and eliminate references to it.

heat.engine.stk_defn.**update_resource_data**(*stack_definition*, *resource_name*, *resource_data*)

Store new resource state data for the specified resource.

This function enables the legacy (non-convergence) path to store updated NodeData as resources are created/updated in a single StackDefinition that lasts for the entire lifetime of the stack operation.

heat.engine.support module

class heat.engine.support.**SupportStatus**(*status='SUPPORTED'*, *message=None*, *version=None*, *previous_status=None*, *substitute_class=None*)

Bases: object

is_substituted(*substitute_class*)

to_dict()

validate()

heat.engine.support.**is_valid_status**(*status*)

heat.engine.sync_point module

heat.engine.sync_point.**create**(*context*, *entity_id*, *traversal_id*, *is_update*, *stack_id*)

Creates a sync point entry in DB.

heat.engine.sync_point.**delete_all**(*context*, *stack_id*, *traversal_id*)

Deletes all sync points of a stack associated with a traversal_id.

heat.engine.sync_point.**deserialize_extra_data**(*db_extra_data*)

heat.engine.sync_point.**deserialize_input_data**(*db_input_data*)

heat.engine.sync_point.**get**(*context*, *entity_id*, *traversal_id*, *is_update*)

Retrieves a sync point entry from DB.

heat.engine.sync_point.**make_key**(**components*)

heat.engine.sync_point.**serialize_extra_data**(*extra_data*)

heat.engine.sync_point.**serialize_input_data**(*input_data*)

heat.engine.sync_point.**str_pack_tuple**(*t*)

heat.engine.sync_point.**sync**(*cnxt*, *entity_id*, *current_traversal*, *is_update*, *propagate*, *predecessors*, *new_data*, *new_resource_failures=None*, *is_skip=False*)

Synchronize resource state and propagate when all predecessors done.

This function updates the sync point with new data and resource failures, and calls the propagate callback when all predecessors have reported.

heat.engine.sync_point.**update_input_data**(*context*, *entity_id*, *current_traversal*, *is_update*, *atomic_key*, *input_data=None*, *extra_data=None*)

heat.engine.sync_point.**update_sync_point**(*cnxt*, *entity_id*, *current_traversal*, *is_update*, *predecessors*, *new_data*, *new_resource_failures=None*, *is_skip=False*)

Update a sync point atomically with new data and failures.

Retry waits up to 60 seconds at most, with exponentially increasing amounts of jitter per resource still outstanding.

heat.engine.template module

class heat.engine.template.**Template**(*template*, **args*, ***kwargs*)

Bases: Mapping

Abstract base class for template format plugins.

All template formats (both internal and third-party) should derive from Template and implement the abstract functions to provide resource definitions and other data.

This is a stable third-party API. Do not add implementations that are specific to internal template formats. Do not add new abstract methods.

add_output(*definition*)

Add an output to the template.

The output is passed as a OutputDefinition object.

abstract add_resource(*definition*, *name=None*)

Add a resource to the template.

The resource is passed as a ResourceDefinition object. If no name is specified, the name from the ResourceDefinition should be used.

all_param_schemata(*files*)

condition_functions = {}

conditions(*stack*)

Return a dictionary of resolved conditions.

classmethod create_empty_template(*version=('heat_template_version', '2015-04-30')*, *from_template=None*)

Create an empty template.

Creates a new empty template with given version. If version is not provided, a new empty HOT template of version 2015-04-30 is returned.

Parameters

version A tuple containing version header of the template version key and value, e.g. `('heat_template_version', '2015-04-30')`

Returns

A new empty template.

property files

functions = {}

abstract get_section_name(*section*)

Get the name of a field within a resource or output definition.

Return the name of the given field (specified by the constants given in heat.engine.rsrc_defn and heat.engine.output) in the template format. This is used in error reporting to help users find the location of errors in the template.

Note that section here does not refer to a top-level section of the template (like parameters, resources, &c.) as it does everywhere else.

classmethod load(*context*, *template_id*, *t=None*)

Retrieve a Template with the given ID from the database.

merge_snippets(*other*)

abstract outputs(*stack*)

Return a dictionary of OutputDefinition objects.

abstract param_schemata(*param_defaults=None*)

Return a dict of parameters.Schema objects for the parameters.

abstract parameters(*stack_identifier*, *user_params*, *param_defaults=None*)

Return a parameters.Parameters object for the stack.

parse(*stack*, *snippet*, *path=''*)

parse_condition(*stack*, *snippet*, *path=”*)

remove_all_resources()

Remove all the resources from the template.

remove_resource(*name*)

Remove a resource from the template.

abstract resource_definitions(*stack*)

Return a dictionary of ResourceDefinition objects.

store(*context*)

Store the Template in the database and return its ID.

validate()

Validate the template.

Validates the top-level sections of the template as well as syntax inside select sections. Some sections are not checked here but in code parts that are responsible for working with the respective sections (e.g. parameters are check by parameters schema class).

heat.engine.template_common module

class heat.engine.template_common.**CommonTemplate**(*template*, **args*, ***kwargs*)

Bases: *Template*

A class of the common implementation for HOT and CFN templates.

This is *not* a stable interface, and any third-parties who create derived classes from it do so at their own risk.

conditions(*stack*)

Return a dictionary of resolved conditions.

outputs(*stack*)

Return a dictionary of OutputDefinition objects.

heat.engine.template_files module

class heat.engine.template_files.**ReadOnlyDict**

Bases: dict

class heat.engine.template_files.**TemplateFiles**(*files*)

Bases: Mapping

store(*ctxt*)

update(*files*)

heat.engine.template_files.**get_files_from_container**(*cnxt*, *files_container*, *files=None*)

heat.engine.timestamp module

class heat.engine.timestamp.**Timestamp**(*db_fetch*, *attribute*)

Bases: object

A descriptor for writing a timestamp to the database.

heat.engine.translation module

class heat.engine.translation.**Translation**(*properties=None*)

Bases: object

Mechanism for translating one properties to other.

Mechanism allows to handle properties - update deprecated/hidden properties to new, resolve values, remove unnecessary. It uses list of TranslationRule objects as rules for translation.

add(*key*, *add_rule*, *prop_value=None*, *prop_data=None*, *validate=False*)

cast_key_to_rule(*key*)

has_translation(*key*)

is_deleted(*key*)

is_replaced(*key*)

replace(*key*, *replace_rule*, *prop_value=None*, *prop_data=None*, *validate=False*)

set_rules(*rules*, *client_resolve=True*, *ignore_resolve_error=False*)

translate(*key*, *prop_value=None*, *prop_data=None*, *validate=False*)

class heat.engine.translation.**TranslationRule**(*properties*, *rule*, *translation_path*, *value=None*, *value_name=None*, *value_path=None*, *client_plugin=None*, *finder=None*, *entity=None*, *custom_value_path=None*)

Bases: object

Translating mechanism one properties to another.

Mechanism uses list of rules, each defines by this class, and can be executed. Working principe: during resource creating after properties defining resource take list of rules, specified by method translation_rules, which should be overloaded for each resource, if its needed, and execute each rule using translate_properties method. Next operations are allowed:

- **ADD. This rule allows to add some value to list-type properties. Only** list-type values can be added to such properties. Using for other cases is prohibited and will be returned with error.
- **REPLACE. This rule allows to replace some property value to another. Used** for all types of properties. Note, that if property has list type, then value will be replaced for all elements of list, where it needed. If element in such property must be replaced by value of another element of this property, value_name must be defined.
- **DELETE. This rule allows to delete some property. If property has list** type, then deleting affects value in all list elements.
- **RESOLVE. This rule allows to resolve some property using client and** the finder function. Finders may require an additional entity key.

ADD = 'Add'

DELETE = 'Delete'

REPLACE = 'Replace'

RESOLVE = 'Resolve'

RULE_KEYS = ('Add', 'Replace', 'Delete', 'Resolve')

get_value_absolute_path(*full_value_name=False*)

validate()

heat.engine.translation.**get_value**(*path*, *props*, *validate=False*)

heat.engine.translation.**resolve_and_find**(*value*, *cplugin*, *finder*, *entity=None*, *ignore_resolve_error=False*)

heat.engine.update module

class heat.engine.update.**StackUpdate**(*existing_stack*, *new_stack*, *previous_stack*, *rollback=False*)

Bases: object

A Task to perform the update of an existing stack to a new template.

dependencies()

Return the Dependencies graph for the update.

Returns a Dependencies object representing the dependencies between update operations to move from an existing stack definition to a new one.

preview()

heat.engine.worker module

class heat.engine.worker.**WorkerService**(*host*, *topic*, *engine_id*, *thread_group_mgr*)

Bases: object

Service that has worker actor in convergence.

This service is dedicated to handle internal messages to the worker (a.k.a. converger) actor in convergence. Messages on this bus will use the cast rather than call method to anycast the message to an engine that will handle it asynchronously. It wont wait for or expect replies from these messages.

RPC_API_VERSION = '1.8'

cancel_check_resource(*cnxt*, *stack_id*)

Cancel check_resource for given stack.

All the workers running for the given stack will be cancelled.

check_resource(*cnxt*, *resource_id*, *current_traversal*, *data*, *is_update*, *adopt_stack_data*, *converge=False*, *skip_propagate=False*, *accumulated_failures=None*, *node_type='resource'*)

Process a node in the dependency graph.

The node may be associated with either an update or a cleanup of its associated resource, or a snapshot deletion.

check_resource_delete_snapshot(*cnxt*, *snapshot_id*, *resource_name*, *start_time=None*, *is_stack_delete=False*, *current_traversal=None*)

start()

stop()

stop_all_workers(*stack*)

Cancel all existing worker threads for the stack.

Threads will stop running at their next yield point, whether or not the resource operations are complete.

stop_traversal(*stack*)

Update current traversal to stop workers from propagating.

Marks the stack as FAILED due to cancellation, but, allows all in_progress resources to complete normally; no worker is stopped abruptly.

Any in-progress traversals are also stopped on all nested stacks that are descendants of the one passed.

heat.engine.worker.**log_exceptions**(*func*)

Module contents

heat.objects package

Submodules

heat.objects.base module

Heat common internal object model

class heat.objects.base.**HeatObject**(*context=None*, ***kwargs*)

Bases: VersionedObject

OBJ_PROJECT_NAMESPACE = 'heat'

VERSION = '1.0'

class heat.objects.base.**HeatObjectRegistry**(**args*, ***kwargs*)

Bases: VersionedObjectRegistry

heat.objects.event module

Event object.

class heat.objects.event.**Event**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat

classmethod count_all_by_stack(*context*, *stack_id*)

classmethod create(*context*, *values*)

```
fields = {'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'id':
Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'physical_resource_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'resource_action': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'resource_name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'resource_status': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'resource_status_reason': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'resource_type': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'rsrc_prop_data_id': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'stack_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'uuid':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}
```

classmethod get_all_by_stack(*context*, *stack_id*, ***kwargs*)

classmethod get_all_by_tenant(*context*, ***kwargs*)

identifier(*stack_identifier*)

Return a unique identifier for the event.

property resource_properties

heat.objects.fields module

class heat.objects.fields.**Json**

Bases: FieldType

coerce(*obj*, *attr*, *value*)

This is called to coerce (if possible) a value on assignment.

This method should convert the value given into the designated type, or throw an exception if this is not possible.

Param:obj
The VersionedObject on which an attribute is being set

Param:attr
The name of the attribute being set

Param:value
The value being set

Returns
A properly-typed value

from_primitive(*obj*, *attr*, *value*)

This is called to deserialize a value.

This method should deserialize a value from the form given by to_primitive() to the designated type.

Param:obj
The VersionedObject on which the value is to be set

Param:attr
The name of the attribute which will hold the value

Param:value
The serialized form of the value

Returns
The natural form of the value

to_primitive(*obj*, *attr*, *value*)

This is called to serialize a value.

This method should serialize a value to the form expected by from_primitive().

Param:obj
The VersionedObject on which the value is set

Param:attr
The name of the attribute holding the value

Param:value
The natural form of the value

Returns
The serialized form of the value

class heat.objects.fields.**JsonField**(***kwargs*)

Bases: AutoTypedField

AUTO_TYPE = <heat.objects.fields.Json object>

class heat.objects.fields.**ListField**(***kwargs*)

Bases: AutoTypedField

AUTO_TYPE = <oslo_versionedobjects.fields.List object>

heat.objects.raw_template module

RawTemplate object.

class heat.objects.raw_template.**RawTemplate**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

VERSION = '1.1'

classmethod create(*context*, *values*)

classmethod delete(*context*, *template_id*)

classmethod encrypt_hidden_parameters(*tmpl*)

property environment

fields = {'environment': Json(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'files': Json(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'files_id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'template': Json(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

property files

property files_id

static from_db_object(*context*, *tpl*, *db_tpl*)

classmethod get_by_id(*context*, *template_id*)

property id

property template

classmethod update_by_id(*context*, *template_id*, *values*)

heat.objects.raw_template_files module

RawTemplateFiles object.

class heat.objects.raw_template_files.**RawTemplateFiles**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

VERSION = '1.0'

classmethod create(*context*, *values*)

fields = {'files': Json(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

property files

property id

heat.objects.resource module

Resource object.

class heat.objects.resource.**Resource**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

property attr_data

classmethod attr_data_delete(*context*, *resource_id*, *attr_id*)

convert_to_convergence(*current_template_id*, *requires*)

classmethod create(*context*, *values*)

classmethod delete(*context*, *resource_id*)

static encrypt_properties_data(*data*)

classmethod exchange_stacks(*context*, *resource_id1*, *resource_id2*)

```
fields = {'action': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'atomic_key': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'attr_data': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'attr_data_id': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'current_template_id': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'data': List(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'engine_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id':
Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'needed_by': List(default=None,nullable=True), 'physical_resource_id':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'replaced_by': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'replaces': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'requires': List(default=None,nullable=True), 'root_stack_id':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'rsrc_metadata': Json(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'rsrc_prop_data_id': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'stack_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'status': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'status_reason': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'uuid':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}
```

classmethod get_all(*context*)

classmethod get_all_active_by_stack(*context*, *stack_id*)

classmethod get_all_by_physical_resource_id(*context*, *physical_resource_id*)

classmethod get_all_by_root_stack(*context*, *stack_id*, *filters*, *cache=False*)

classmethod get_all_by_stack(*context*, *stack_id*, *filters=None*)

classmethod get_all_stack_ids_by_root_stack(*context*, *stack_id*)

classmethod get_by_name_and_stack(*context*, *resource_name*, *stack_id*)

classmethod get_obj(*context*, *resource_id*, *refresh=False*, *fields=None*)

property properties_data

classmethod purge_deleted(*context*, *stack_id*)

refresh()

classmethod replacement(*context*, *existing_res_id*, *new_res_values*, *atomic_key=0*, *expected_engine_id=None*)

select_and_update(*values*, *expected_engine_id=None*, *atomic_key=0*)

classmethod select_and_update_by_id(*context*, *resource_id*, *values*, *expected_engine_id=None*, *atomic_key=0*)

classmethod store_attributes(*context*, *resource_id*, *atomic_key*, *attr_data*, *attr_id*)

update_and_save(*values*)

classmethod update_by_id(*context*, *resource_id*, *values*)

update_metadata(*metadata*)

class heat.objects.resource.**ResourceCache**

Bases: object

delete_all()

set_by_stack_id(*resources*)

heat.objects.resource.**retry_on_conflict**(*func*)

heat.objects.resource_data module

ResourceData object.

class heat.objects.resource_data.**ResourceData**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

classmethod delete(*resource*, *key*)

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'decrypt_method': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'key': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'redact': Boolean(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'resource_id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'value': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

classmethod get_all(*resource*, **args*, ***kwargs*)

classmethod get_by_key(*context*, *resource_id*, *key*)

classmethod get_obj(*resource*, *key*)

classmethod get_val(*resource*, *key*)

classmethod set(*resource*, *key*, *value*, **args*, ***kwargs*)

heat.objects.resource_properties_data module

ResourcePropertiesData object.

class heat.objects.resource_properties_data.**ResourcePropertiesData**(*context=None*, ***kwargs*)

Bases: VersionedObject, VersionedObjectDictCompat, ComparableVersionedObject

classmethod create(*context*, *data*)

classmethod create_or_update(*context*, *data*, *rpd_id=None*)

static encrypt_properties_data(*data*)

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'data': Json(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

static get_by_id(*context*, *id*)

heat.objects.resource_snapshot module

ResourceSnapshot object.

class heat.objects.resource_snapshot.**ResourceSnapshot**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

classmethod create(*context*, *values*)

classmethod delete(*context*, *resource_snapshot_id*)

classmethod delete_all_by_snapshot(*context*, *snapshot_id*)

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'data': Json(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'resource_name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'snapshot_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get(*context*, *resource_snapshot_id*)

classmethod get_all(*context*, *snapshot_id*)

classmethod get_by_snapshot_and_rsrc(*context*, *snapshot_id*, *resource_name*)

classmethod update(*context*, *resource_snapshot_id*, *values*)

heat.objects.service module

Service object.

class heat.objects.service.**Service**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

classmethod active_service_count(*context*)

Return the number of services reportedly active.

classmethod create(*context*, *values*)

classmethod delete(*context*, *service_id*, *soft_delete=True*)

```
fields = {'binary': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'deleted_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'engine_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'host': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'hostname': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'id':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'report_interval': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'topic': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}
```

classmethod get_all(*context*)

classmethod get_all_by_args(*context*, *host*, *binary*, *hostname*)

classmethod get_by_id(*context*, *service_id*)

classmethod update_by_id(*context*, *service_id*, *values*)

heat.objects.snapshot module

Snapshot object.

class heat.objects.snapshot.**Snapshot**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

classmethod count_all_by_stack(*context*, *stack_id*)

classmethod create(*context*, *values*)

classmethod delete(*context*, *snapshot_id*)

```
fields = {'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'data': Json(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'stack_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'status': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'status_reason': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'tenant': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}
```

classmethod get_all(*context*, *stack_id*, *load_rsrc_snapshot=False*)

classmethod get_all_by_stack(*context*, *stack_id*)

classmethod get_snapshot(*context*, *snapshot_id*, *load_rsrc_snapshot=False*)

classmethod get_snapshot_by_stack(*context*, *snapshot_id*, *stack*, *load_rsrc_snapshot=False*)

classmethod load_rsrc_snapshots(*context*, *snapshot_id*, *show_meta=False*)

classmethod update(*context*, *snapshot_id*, *values*)

heat.objects.software_config module

SoftwareConfig object.

class heat.objects.software_config.**SoftwareConfig**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

classmethod count_all(*context*, ***kwargs*)

classmethod create(*context*, *values*)

classmethod delete(*context*, *config_id*)

```
fields = {'config': Json(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'group': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'tenant': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}
```

classmethod get_all(*context*, ***kwargs*)

classmethod get_by_id(*context*, *config_id*)

heat.objects.software_deployment module

SoftwareDeployment object.

class heat.objects.software_deployment.**SoftwareDeployment**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

classmethod count_all(*context*)

classmethod create(*context*, *values*)

classmethod delete(*context*, *deployment_id*)

```
fields = {'action': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'config': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'config_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'id':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'input_values': Json(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'output_values': Json(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'server_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'stack_user_project_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'status': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'status_reason': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'tenant': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}
```

classmethod get_all(*context*, *server_id=None*)

classmethod get_by_id(*context*, *deployment_id*)

classmethod update_by_id(*context*, *deployment_id*, *values*)

Note this is a bit unusual as it returns the object.

Other update_by_id methods return a bool (was it updated).

heat.objects.stack module

Stack object.

class heat.objects.stack.**Stack**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

classmethod count_all(*context*, ***kwargs*)

classmethod count_total_resources(*context*, *stack_id*)

classmethod create(*context*, *values*)

classmethod delete(*context*, *stack_id*)

classmethod encrypt_hidden_parameters(*tmpl*)

```
fields = {'action': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'backup': Boolean(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'convergence': Boolean(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'current_deps': Json(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'current_traversal': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'deleted_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'disable_rollback': Boolean(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'id':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'nested_depth': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'owner_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'parent_resource_name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'prev_raw_template': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'prev_raw_template_id': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'raw_template_id': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'raw_template_obj': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'stack_user_project_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'status': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'status_reason': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'tenant': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'timeout': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'user_creds_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'username': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}
```

classmethod get_all(*context*, *limit=None*, *sort_keys=None*, *marker=None*, *sort_dir=None*, *filters=None*, *show_deleted=False*, *show_nested=False*, *show_hidden=False*, *tags=None*, *tags_any=None*, *not_tags=None*, *not_tags_any=None*, *eager_load=False*)

classmethod get_all_by_owner_id(*context*, *owner_id*)

classmethod get_all_by_root_owner_id(*context*, *root_owner_id*)

classmethod get_by_id(*context*, *stack_id*, ***kwargs*)

classmethod get_by_name(*context*, *stack_name*)

classmethod get_by_name_and_owner_id(*context*, *stack_name*, *owner_id*)

classmethod get_root_id(*context*, *stack_id*)

classmethod get_status(*context*, *stack_id*)

Return action and status for the given stack.

identifier()

Return an identifier for this stack.

classmethod persist_state_and_release_lock(*context*, *stack_id*, *engine_id*, *values*)

property raw_template

refresh()

classmethod select_and_update(*context*, *stack_id*, *values*, *exp_trvsl=None*)

Update the stack by selecting on traversal ID.

Uses UPDATE WHERE (compare and swap) to catch any concurrent update problem.

If the stack is found with given traversal, it is updated.

If there occurs a race while updating, only one will succeed and other will get return value of False.

property tags

update_and_save(*values*)

classmethod update_by_id(*context*, *stack_id*, *values*)

Update and return (boolean) if it was updated.

Note: the underlying stack_update filters by current_traversal and stack_id.

heat.objects.stack_lock module

StackLock object.

class heat.objects.stack_lock.**StackLock**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

classmethod create(*context*, *stack_id*, *engine_id*)

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'engine_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'stack_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

classmethod get_engine_id(*context*, *stack_id*)

classmethod release(*context*, *stack_id*, *engine_id*)

classmethod steal(*context*, *stack_id*, *old_engine_id*, *new_engine_id*)

heat.objects.stack_tag module

StackTag object.

class heat.objects.stack_tag.**StackTag**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'stack_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'tag': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

classmethod get_obj(*context*, *tag*)

class heat.objects.stack_tag.**StackTagList**(**args*, ***kwargs*)

Bases: *HeatObject*, ObjectListBase

classmethod delete(*context*, *stack_id*)

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod from_db_object(*context*, *db_tags*)

classmethod get(*context*, *stack_id*)

classmethod set(*context*, *stack_id*, *tags*)

heat.objects.sync_point module

SyncPoint object.

class heat.objects.sync_point.**SyncPoint**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

classmethod create(*context*, *values*)

classmethod delete_all_by_stack_and_traversal(*context*, *stack_id*, *traversal_id*)

fields = {'atomic_key': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'entity_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'extra_data': Json(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'input_data': Json(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'is_update': Boolean(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'stack_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'traversal_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

classmethod get_by_key(*context*, *entity_id*, *traversal_id*, *is_update*)

classmethod update_input_data(*context*, *entity_id*, *traversal_id*, *is_update*, *atomic_key*, *input_data*, *extra_data=None*)

heat.objects.user_creds module

UserCreds object.

class heat.objects.user_creds.**UserCreds**(*context=None*, ***kwargs*)

Bases: *HeatObject*, VersionedObjectDictCompat, ComparableVersionedObject

property auth_url

classmethod create(*context*)

property created_at

property decrypt_method

classmethod delete(*context*, *user_creds_id*)

```
fields = {'auth_url': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'decrypt_method': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'password': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'region_name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'tenant': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'tenant_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'trust_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'trustor_user_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'username': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}
```

classmethod get_by_id(*context*, *user_creds_id*)

property id

property password

property region_name

property tenant

property tenant_id

property trust_id

property trustor_user_id

property updated_at

property username

Module contents

heat.policies package

Submodules

heat.policies.actions module

heat.policies.actions.**list_rules**()

heat.policies.base module

heat.policies.base.**list_rules**()

heat.policies.build_info module

heat.policies.build_info.**list_rules**()

heat.policies.cloudformation module

heat.policies.cloudformation.**list_rules**()

heat.policies.events module

heat.policies.events.**list_rules**()

heat.policies.resource module

heat.policies.resource.**list_rules**()

heat.policies.resource_types module

heat.policies.resource_types.**list_rules**()

heat.policies.service module

heat.policies.service.**list_rules**()

heat.policies.software_configs module

heat.policies.software_configs.**list_rules**()

heat.policies.software_deployments module

heat.policies.software_deployments.**list_rules**()

heat.policies.stacks module

heat.policies.stacks.**list_rules**()

Module contents

heat.policies.**list_rules**()

heat.rpc package

Submodules

heat.rpc.api module

heat.rpc.client module

Client side of the heat engine RPC API.

class heat.rpc.client.**EngineClient**

Bases: object

Client side of the heat engine rpc API.

API version history:

```
1.0 - Initial version.
1.1 - Add support_status argument to list_resource_types()
1.4 - Add support for service list
1.9 - Add template_type option to generate_template()
1.10 - Add support for software config list
1.11 - Add support for template versions list
1.12 - Add with_detail option for stack resources list
1.13 - Add support for template functions list
1.14 - Add cancel_with_rollback option to stack_cancel_update
1.15 - Add preview_update_stack() call
1.16 - Adds version, type_name to list_resource_types()
1.17 - Add files to validate_template
1.18 - Add show_nested to validate_template
1.19 - Add show_output and list_outputs for returning stack outputs
1.20 - Add resolve_outputs to stack show
1.21 - Add deployment_id to create_software_deployment
1.22 - Add support for stack export
1.23 - Add environment_files to create/update/preview/validate
1.24 - Adds ignorable_errors to validate_template
1.25 - list_stack_resource filter update
1.26 - Add mark_unhealthy
1.27 - Add check_software_deployment
1.28 - Add get_environment call
1.29 - Add template_id to create_stack/update_stack
1.30 - Add possibility to resource_type_* return descriptions
1.31 - Add nested_depth to list_events, when nested_depth is specified
       add root_stack_id to response
1.32 - Add get_files call
1.33 - Remove tenant_safe from list_stacks, count_stacks
       and list_software_configs
1.34 - Add migrate_convergence_1 call
1.35 - Add with_condition to list_template_functions
1.36 - Add files_container to create/update/preview/validate
```

BASE_RPC_API_VERSION = '1.0'

abandon_stack(*ctxt*, *stack_identity*)

Deletes a given stack but resources would not be deleted.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack you want to abandon.

authenticated_to_backend(*ctxt*)

Validate the credentials in the RPC context.

Verify that the credentials in the RPC context are valid for the current cloud backend.

Parameters

ctxt RPC context.

call(*ctxt*, *msg*, *version=None*, *timeout=None*)

cast(*ctxt*, *msg*, *version=None*)

check_software_deployment(*cnxt*, *deployment_id*, *timeout*)

count_stacks(*ctxt*, *filters=None*, *show_deleted=False*, *show_nested=False*, *show_hidden=False*, *tags=None*, *tags_any=None*, *not_tags=None*, *not_tags_any=None*)

Returns the number of stacks that match the given filters.

Parameters

- **ctxt** RPC context.
- **filters** a dict of ATTR:VALUE to match against stacks
- **show_deleted** if true, count will include the deleted stacks
- **show_nested** if true, count will include nested stacks
- **show_hidden** if true, count will include hidden stacks
- **tags** count stacks containing these tags. If multiple tags are passed, they will be combined using the boolean AND expression
- **tags_any** count stacks containing these tags. If multiple tags are passed, they will be combined using the boolean OR expression
- **not_tags** count stacks not containing these tags. If multiple tags are passed, they will be combined using the boolean AND expression
- **not_tags_any** count stacks not containing these tags. If multiple tags are passed, they will be combined using the boolean OR expression

Returns

an integer representing the number of matched stacks

create_software_config(*cnxt*, *group*, *name*, *config*, *inputs=None*, *outputs=None*, *options=None*)

create_software_deployment(*cnxt*, *server_id*, *config_id=None*, *input_values=None*, *action='INIT'*, *status='COMPLETE'*, *status_reason=''*, *stack_user_project_id=None*, *deployment_id=None*)

create_stack(*ctxt*, *stack_name*, *template*, *params*, *files*, *args*, *environment_files=None*, *files_container=None*)

Creates a new stack using the template provided.

Note that at this stage the template has already been fetched from the heat-api process if using a template-url.

Parameters

- **ctxt** RPC context.
- **stack_name** Name of the stack you want to create.
- **template** Template of stack you want to create.
- **params** Stack Input Params/Environment
- **files** files referenced from the environment.
- **args** Request parameters/args passed from API
- **environment_files** (*list or None*) optional ordered list of environment file names included in the files dict
- **files_container** name of swift container

delete_snapshot(*cnxt*, *stack_identity*, *snapshot_id*)

delete_software_config(*cnxt*, *config_id*)

delete_software_deployment(*cnxt*, *deployment_id*)

delete_stack(*ctxt*, *stack_identity*, *cast=False*)

Deletes a given stack.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack you want to delete.
- **cast** cast the message instead of using call (default: False)

You probably never want to use cast(). If you do, youll never hear about any exceptions the call might raise.

describe_stack_resource(*ctxt*, *stack_identity*, *resource_name*, *with_attr=False*)

Get detailed resource information about a particular resource.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack.
- **resource_name** the Resource.

describe_stack_resources(*ctxt*, *stack_identity*, *resource_name*)

Get detailed resource information about one or more resources.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack.
- **resource_name** the Resource.

export_stack(*ctxt*, *stack_identity*)

Exports the stack data in JSON format.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack you want to export.

find_physical_resource(*ctxt*, *physical_resource_id*)

Return an identifier for the resource.

Parameters

- **ctxt** RPC context.
- **physcial_resource_id** The physical resource ID to look up.

generate_template(*ctxt*, *type_name*, *template_type='cfn'*)

Generate a template based on the specified type.

Parameters

- **ctxt** RPC context.
- **type_name** The resource type name to generate a template for.
- **template_type** the template type to generate, cfn or hot.

get_environment(*context*, *stack_identity*)

Returns the environment for an existing stack.

Parameters

- **context** RPC context
- **stack_identity** identifies the stack

Return type
dict

get_files(*context*, *stack_identity*)

Returns the files for an existing stack.

Parameters

- **context** RPC context
- **stack_identity** identifies the stack

Return type
dict

get_revision(*ctxt*)

get_template(*ctxt*, *stack_identity*)

Get the template.

Parameters

- **ctxt** RPC context.
- **stack_name** Name of the stack you want to see.

identify_stack(*ctxt*, *stack_name*)

Returns the full stack identifier for a single, live stack.

Parameters

- **ctxt** RPC context.
- **stack_name** Name of the stack you want to see, or None to see all

ignore_error_by_name(*name*)

Returns a context manager that filters exceptions with a given name.

Parameters

name Name to compare the local exception name to.

list_events(*ctxt*, *stack_identity*, *filters=None*, *limit=None*, *marker=None*, *sort_keys=None*, *sort_dir=None*, *nested_depth=None*)

Lists all events associated with a given stack.

It supports pagination (`limit` and `marker`), sorting (`sort_keys` and `sort_dir`) and filtering(filters) of the results.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack you want to get events for
- **filters** a dict with attribute:value to filter the list
- **limit** the number of events to list (integer or string)
- **marker** the ID of the last event in the previous page
- **sort_keys** an array of fields used to sort the list
- **sort_dir** the direction of the sort (asc or desc).
- **nested_depth** Levels of nested stacks to list events for.

list_outputs(*cntx*, *stack_identity*)

list_resource_types(*ctxt*, *support_status=None*, *type_name=None*, *heat_version=None*, *with_description=False*)

Get a list of valid resource types.

Parameters

- **ctxt** RPC context.
- **support_status** Support status of resource type
- **type_name** Resource types name (regular expression allowed)
- **heat_version** Heat version
- **with_description** Either return resource type description or not

list_services(*cnxt*)

list_software_configs(*cnxt*, *limit=None*, *marker=None*)

list_software_deployments(*cnxt*, *server_id=None*)

list_stack_resources(*ctxt*, *stack_identity*, *nested_depth=0*, *with_detail=False*, *filters=None*)

List the resources belonging to a stack.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack.
- **nested_depth** Levels of nested stacks of which list resources.
- **with_detail** show detail for resources in list.
- **filters** a dict with attribute:value to search the resources

list_stacks(*ctxt*, *limit=None*, *marker=None*, *sort_keys=None*, *sort_dir=None*, *filters=None*, *show_deleted=False*, *show_nested=False*, *show_hidden=False*, *tags=None*, *tags_any=None*, *not_tags=None*, *not_tags_any=None*)

Returns attributes of all stacks.

It supports pagination (`limit` and `marker`), sorting (`sort_keys` and `sort_dir`) and filtering (`filters`) of the results.

Parameters

- **ctxt** RPC context.
- **limit** the number of stacks to list (integer or string)
- **marker** the ID of the last item in the previous page
- **sort_keys** an array of fields used to sort the list
- **sort_dir** the direction of the sort (asc or desc)
- **filters** a dict with attribute:value to filter the list
- **show_deleted** if true, show soft-deleted stacks
- **show_nested** if true, show nested stacks
- **show_hidden** if true, show hidden stacks
- **tags** show stacks containing these tags. If multiple tags are passed, they will be combined using the boolean AND expression
- **tags_any** show stacks containing these tags. If multiple tags are passed, they will be combined using the boolean OR expression
- **not_tags** show stacks not containing these tags. If multiple tags are passed, they will be combined using the boolean AND expression
- **not_tags_any** show stacks not containing these tags. If multiple tags are passed, they will be combined using the boolean OR expression

Returns

a list of stacks

list_template_functions(*ctxt*, *template_version*, *with_condition=False*)

Get a list of available functions in a given template type.

Parameters

- **ctxt** RPC context
- **template_version** template format/version tuple for which you want to get the list of functions.
- **with_condition** return includes condition functions.

list_template_versions(*ctxt*)

Get a list of available template versions.

Parameters

ctxt RPC context.

local_error_name(*error*)

Returns the name of the error with any _Remote postfix removed.

Parameters

error Remote raised error to derive the name from.

static make_msg(*method*, ***kwargs*)

metadata_software_deployments(*cnxt*, *server_id*)

migrate_convergence_1(*ctxt*, *stack_id*)

Migrate the stack to convergence engine

Parameters

- **ctxt** RPC context
- **stack_name** Name of the stack you want to migrate

preview_stack(*ctxt*, *stack_name*, *template*, *params*, *files*, *args*, *environment_files=None*, *files_container=None*)

Simulates a new stack using the provided template.

Note that at this stage the template has already been fetched from the heat-api process if using a template-url.

Parameters

- **ctxt** RPC context.
- **stack_name** Name of the stack you want to create.
- **template** Template of stack you want to create.
- **params** Stack Input Params/Environment
- **files** files referenced from the environment.
- **args** Request parameters/args passed from API
- **environment_files** (*list or None*) optional ordered list of environment file names included in the files dict
- **files_container** name of swift container

preview_update_stack(*ctxt*, *stack_identity*, *template*, *params*, *files*, *args*, *environment_files=None*, *files_container=None*)

Returns the resources that would be changed in an update.

Based on the provided template and parameters.

Requires RPC version 1.15 or above.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack you wish to update.
- **template** New template for the stack.
- **params** Stack Input Params/Environment
- **files** files referenced from the environment.
- **args** Request parameters/args passed from API
- **environment_files** (*list or None*) optional ordered list of environment file names included in the files dict
- **files_container** name of swift container

resource_mark_unhealthy(*ctxt*, *stack_identity*, *resource_name*, *mark_unhealthy*, *resource_status_reason=None*)

Mark the resource as unhealthy or healthy.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack.
- **resource_name** the Resource.
- **mark_unhealthy** indicates whether the resource is unhealthy.
- **resource_status_reason** reason for health change.

resource_schema(*ctxt*, *type_name*, *with_description=False*)

Get the schema for a resource type.

Parameters

- **ctxt** RPC context.
- **with_description** Return resource with description or not.

resource_signal(*ctxt*, *stack_identity*, *resource_name*, *details*, *sync_call=False*)

Generate an alarm on the resource.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack.
- **resource_name** the Resource.
- **details** the details of the signal.

show_output(*cntx*, *stack_identity*, *output_key*)

show_snapshot(*cnxt*, *stack_identity*, *snapshot_id*)

show_software_config(*cnxt*, *config_id*)

show_software_deployment(*cnxt*, *deployment_id*)

show_stack(*ctxt*, *stack_identity*, *resolve_outputs=True*)

Returns detailed information about one or all stacks.

Parameters

- **ctxt** RPC context.
- **stack_identity** Name of the stack you want to show, or None to show all
- **resolve_outputs** If True, stack outputs will be resolved

signal_software_deployment(*cnxt*, *deployment_id*, *details*, *updated_at=None*)

stack_cancel_update(*ctxt*, *stack_identity*, *cancel_with_rollback=True*)

stack_check(*ctxt*, *stack_identity*)

stack_list_snapshots(*cnxt*, *stack_identity*)

stack_restore(*cnxt*, *stack_identity*, *snapshot_id*)

stack_resume(*ctxt*, *stack_identity*)

stack_snapshot(*ctxt*, *stack_identity*, *name*)

stack_suspend(*ctxt*, *stack_identity*)

update_software_deployment(*cnxt*, *deployment_id*, *config_id=None*, *input_values=None*, *output_values=None*, *action=None*, *status=None*, *status_reason=None*, *updated_at=None*)

update_stack(*ctxt*, *stack_identity*, *template*, *params*, *files*, *args*, *environment_files=None*, *files_container=None*)

Updates an existing stack based on the provided template and params.

Note that at this stage the template has already been fetched from the heat-api process if using a template-url.

Parameters

- **ctxt** RPC context.
- **stack_name** Name of the stack you want to create.
- **template** Template of stack you want to create.
- **params** Stack Input Params/Environment
- **files** files referenced from the environment.
- **args** Request parameters/args passed from API
- **environment_files** (*list or None*) optional ordered list of environment file names included in the files dict

- **files_container** name of swift container

validate_template(*ctxt*, *template*, *params=None*, *files=None*, *environment_files=None*, *files_container=None*, *show_nested=False*, *ignorable_errors=None*)

Uses the stack parser to check the validity of a template.

Parameters

- **ctxt** RPC context.
- **template** Template of stack you want to create.
- **params** Stack Input Params/Environment
- **files** files referenced from the environment/template.
- **environment_files** ordered list of environment file names included in the files dict
- **files_container** name of swift container
- **show_nested** if True nested templates will be validated
- **ignorable_errors** List of error_code to be ignored as part of validation

heat.rpc.listener_client module

Client side of the heat worker RPC API.

class heat.rpc.listener_client.**EngineListenerClient**(*engine_id*)

Bases: object

Client side of the heat listener RPC API.

API version history:

```
1.0 - Initial version.
```

BASE_RPC_API_VERSION = '1.0'

is_alive(*ctxt*)

heat.rpc.worker_api module

heat.rpc.worker_client module

Client side of the heat worker RPC API.

class heat.rpc.worker_client.**WorkerClient**

Bases: object

Client side of the heat worker RPC API.

API version history:

```
1.0 - Initial version.
1.1 - Added check_resource.
1.2 - Add adopt data argument to check_resource.
1.3 - Added cancel_check_resource API.
```

(continues on next page)

(continued from previous page)

```
1.4 - Add converge argument to check_resource
1.5 - Add skip_propagate argument to check_resource for CHECK action
1.6 - Add accumulated_failures argument to check_resource for CHECK
1.7 - Added check_resource_delete_snapshot
1.8 - Add node_type argument to check_resource for snapshot deletion
```

BASE_RPC_API_VERSION = '1.0'

cancel_check_resource(*ctxt*, *stack_id*, *engine_id*)

Send check-resource cancel message.

Sends a cancel message to given heat engine worker.

cast(*ctxt*, *msg*, *version=None*)

check_resource(*ctxt*, *resource_id*, *current_traversal*, *data*, *is_update*, *adopt_stack_data*, *converge=False*, *skip_propagate=False*, *accumulated_failures=None*, *node_type='resource'*)

check_resource_delete_snapshot(*ctxt*, *snapshot_id*, *resource_name*, *start_time*, *is_stack_delete=False*, *current_traversal=None*)

static make_msg(*method*, ***kwargs*)

Module contents

heat.scaling package

Submodules

heat.scaling.cooldown module

class heat.scaling.cooldown.**CooldownMixin**

Bases: object

Utility class to encapsulate Cooldown related logic.

This logic includes both cooldown timestamp comparing and scaling in progress checking.

handle_metadata_reset()

heat.scaling.lbutils module

heat.scaling.lbutils.**reconfigure_loadbalancers**(*load_balancers*, *id_list*)

Notify the LoadBalancer to reload its config.

This must be done after activation (instance in ACTIVE state), otherwise the instances IP addresses may not be available.

heat.scaling.rolling_update module

`heat.scaling.rolling_update.needs_update(targ_capacity, curr_capacity, num_up_to_date)`

Return whether there are more batch updates to do.

Inputs are the target size for the group, the current size of the group, and the number of members that already have the latest definition.

`heat.scaling.rolling_update.next_batch(targ_capacity, curr_capacity, num_up_to_date, batch_size, min_in_service)`

Return details of the next batch in a batched update.

The result is a tuple containing the new size of the group and the number of members that may receive the new definition (by a combination of creating new members and updating existing ones).

Inputs are the target size for the group, the current size of the group, the number of members that already have the latest definition, the batch size, and the minimum number of members to keep in service during a rolling update.

heat.scaling.scalingutil module

`heat.scaling.scalingutil.calculate_new_capacity(current, adjustment, adjustment_type, min_adjustment_step, minimum, maximum)`

Calculates new capacity from the given adjustments.

Given the current capacity, calculates the new capacity which results from applying the given adjustment of the given adjustment-type. The new capacity will be kept within the maximum and minimum bounds.

heat.scaling.template module

`heat.scaling.template.make_template(resource_definitions, version=('heat_template_version', '2015-04-30'), child_env=None)`

Return a Template object containing the given resource definitions.

By default, the template will be in the HOT format. A different format can be specified by passing a (version_type, version_string) tuple matching any of the available template format plugins.

`heat.scaling.template.member_definitions(old_resources, new_definition, num_resources, num_new, get_new_id, customise=<function _identity>)`

Iterate over resource definitions for a scaling group

Generates the definitions for the next change to the scaling group. Each item is a (name, definition) tuple.

The input is a list of (name, definition) tuples for existing resources in the group, sorted in the order that they should be replaced or removed (i.e. the resource that should be the first to be replaced (on update) or removed (on scale down) appears at the beginning of the list.) New resources are added or old resources removed as necessary to ensure a total of num_resources.

The number of resources to have their definition changed to the new one is controlled by num_new. This value includes any new resources to be added, with any shortfall made up by modifying the definitions of existing resources.

Module contents

Module contents

CHAPTER

FIVE

FOR CONTRIBUTORS

- If you are a new contributor to Heat please refer: *So You Want to Contribute*

5.1 Heat Contributor Guidelines

5.1.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc. Below will cover the more project specific information you need to get started with heat.

Communication

- IRC channel #heat at OFTC
- Mailing list (prefix subjects with `[heat]` for faster responses) http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-discuss

Contacting the Core Team

Please refer the heat Core Team contacts.

New Feature Planning

heat features are tracked on Storyboard.

Task Tracking

We track our tasks in Storyboard. If youre looking for some smaller, easier work item to pick up and get started on, search for the low-hanging-fruit tag.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so on Storyboard.

Getting Your Patch Merged

All changes proposed to the heat project require one or two +2 votes from heat core reviewers before one of the core reviewers can approve patch by giving `Workflow +1` vote.

Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search