



Red Hat OpenStack Platform 16.2

Autoscaling for Instances

Configure Autoscaling in Red Hat OpenStack Platform

Red Hat OpenStack Platform 16.2 Autoscaling for Instances

Configure Autoscaling in Red Hat OpenStack Platform

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Use Red Hat OpenStack Platform telemetry components and heat templates to automatically launch instances for workloads.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	3
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	4
CHAPTER 1. INTRODUCTION TO AUTOSCALING COMPONENTS	5
1.1. DATA COLLECTION SERVICE (CEILOMETER) FOR AUTOSCALING	5
1.1.1. Publishers	5
1.2. TIME-SERIES DATABASE SERVICE (GNOCCHI) FOR AUTOSCALING	5
1.3. ALARMING SERVICE (AODH)	6
1.4. ORCHESTRATION SERVICE (HEAT) FOR AUTOSCALING	6
CHAPTER 2. CONFIGURING AND DEPLOYING THE OVERCLOUD FOR AUTOSCALING	7
2.1. CONFIGURING THE OVERCLOUD FOR AUTOSCALING	7
2.2. DEPLOYING THE OVERCLOUD FOR AUTOSCALING	8
2.2.1. Deploying the overcloud for autoscaling by using director	9
2.2.2. Deploying the overcloud for autoscaling in a standalone environment	9
2.3. VERIFYING THE OVERCLOUD DEPLOYMENT FOR AUTOSCALING	10
CHAPTER 3. USING THE HEAT SERVICE FOR AUTOSCALING	14
3.1. CREATING THE GENERIC ARCHIVE POLICY FOR AUTOSCALING	14
3.2. CONFIGURING A HEAT TEMPLATE FOR AUTOMATICALLY SCALING INSTANCES	15
3.3. PREPARING THE STANDALONE DEPLOYMENT FOR AUTOSCALING	18
3.4. CREATING THE STACK DEPLOYMENT FOR AUTOSCALING	20
CHAPTER 4. TESTING AND TROUBLESHOOTING AUTOSCALING	24
4.1. TESTING AUTOMATIC SCALING UP OF INSTANCES	24
4.2. TESTING AUTOMATIC SCALING DOWN OF INSTANCES	25
4.3. TROUBLESHOOTING FOR AUTOSCALING	26
4.4. USING CPU TELEMETRY VALUES FOR AUTOSCALING THRESHOLD WHEN USING RATE:MEAN AGGREGATION	28
4.4.1. Calculating CPU telemetry values as a percentage	28
4.4.2. Displaying instance workload vCPU as a percentage	29
4.4.3. Retrieving available telemetry for an instance workload	31

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Providing documentation feedback in Jira

Use the [Create Issue](#) form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Platform Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.
2. Click the following link to open a the **Create Issue** page: [Create Issue](#)
3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
4. Click **Create**.

CHAPTER 1. INTRODUCTION TO AUTOSCALING COMPONENTS

Use telemetry components to collect data about your Red Hat OpenStack Platform (RHOSP) environment, such as CPU, storage, and memory usage. You can launch and scale instances in response to workload demand and resource availability. You can define the upper and lower bounds of telemetry data that control the scaling of instances in your Orchestration service (heat) templates.

Control automatic instance scaling with the following telemetry components:

- **Data collection:** Telemetry uses the data collection service (Ceilometer) to gather metric and event data.
- **Storage:** Telemetry stores metrics data in the time-series database service (gnocchi).
- **Alarm:** Telemetry uses the Alarming service (aodh) to trigger actions based on rules against metrics or event data collected by Ceilometer.

1.1. DATA COLLECTION SERVICE (CEILOMETER) FOR AUTOSCALING

You can use Ceilometer to collect data about metering and event information for Red Hat OpenStack Platform (RHOSP) components.

The Ceilometer service uses three agents to collect data from RHOSP components:

- **A compute agent (ceilometer-agent-compute):** Runs on each Compute node and polls for resource use statistics.
- **A central agent (ceilometer-agent-central):** Runs on the Controller nodes to poll for resource use statistics for resources that are not provided by Compute nodes.
- **A notification agent (ceilometer-agent-notification):** Runs on the Controller nodes and consumes messages from the message queues to build event and metering data.

The Ceilometer agents use publishers to send data to the corresponding end points, for example the time-series database service (gnocchi).

Additional resources

- [Ceilometer](#) in the *Operational Measurements* guide.

1.1.1. Publishers

In Red Hat OpenStack Platform (RHOSP), you can use several transport methods to transfer the collected data into storage or external systems, such as Service Telemetry Framework (STF).

When you enable the gnocchi publisher, the measurement and resource information is stored as time-series data.

1.2. TIME-SERIES DATABASE SERVICE (GNOCCHI) FOR AUTOSCALING

Gnocchi is a time-series database that you can use for storing metrics in SQL. The Alarming service (aodh) and Orchestration service (heat) use the data stored in gnocchi for autoscaling.

Additional resources

- [Storage with gnocchi](#).

1.3. ALARMING SERVICE (AODH)

You can configure the Alarming service (aodh) to trigger actions based on rules against metrics data collected by Ceilometer and stored in gnocchi. Alarms can be in one of the following states:

- **Ok**: The metric or event is in an acceptable state.
- **Firing**: The metric or event is outside of the defined **Ok** state.
- **insufficient data**: The alarm state is unknown, for example, if there is no data for the requested granularity, or the check has not been executed yet, and so on.

1.4. ORCHESTRATION SERVICE (HEAT) FOR AUTOSCALING

Director uses Orchestration service (heat) templates as the template format for the overcloud deployment. Heat templates are usually expressed in YAML format. The purpose of a template is to define and create a stack, which is a collection of resources that heat creates, and the configuration of the resources. Resources are objects in Red Hat OpenStack Platform (RHOSP) and can include compute resources, network configuration, security groups, scaling rules, and custom resources.

Additional resources

- [Understanding heat templates](#).

CHAPTER 2. CONFIGURING AND DEPLOYING THE OVERCLOUD FOR AUTOSCALING

You must configure the templates for the services on your overcloud that enable autoscaling.

Procedure

1. Create environment templates and a resource registry for autoscaling services before you deploy the overcloud for autoscaling. For more information, see [Section 2.1, “Configuring the overcloud for autoscaling”](#)
2. Deploy the overcloud. For more information, see [Section 2.2, “Deploying the overcloud for autoscaling”](#)

2.1. CONFIGURING THE OVERCLOUD FOR AUTOSCALING

Create the environment templates and resource registry that you need to deploy the services that provide autoscaling.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Create a directory for the autoscaling configuration files:

```
$ mkdir -p $HOME/templates/autoscaling/
```

3. Create the resource registry file for the definitions that the services require for autoscaling:

```
$ cat <<EOF > $HOME/templates/autoscaling/resources-autoscaling.yaml
resource_registry:
  OS::TripleO::Services::AodhApi: /usr/share/openstack-tripleo-heat-
templates/deployment/aodh/aodh-api-container-puppet.yaml
  OS::TripleO::Services::AodhEvaluator: /usr/share/openstack-tripleo-heat-
templates/deployment/aodh/aodh-evaluator-container-puppet.yaml
  OS::TripleO::Services::AodhListener: /usr/share/openstack-tripleo-heat-
templates/deployment/aodh/aodh-listener-container-puppet.yaml
  OS::TripleO::Services::AodhNotifier: /usr/share/openstack-tripleo-heat-
templates/deployment/aodh/aodh-notifier-container-puppet.yaml
  OS::TripleO::Services::CeilometerAgentCentral: /usr/share/openstack-tripleo-heat-
templates/deployment/ceilometer/ceilometer-agent-central-container-puppet.yaml
  OS::TripleO::Services::CeilometerAgentNotification: /usr/share/openstack-tripleo-heat-
templates/deployment/ceilometer/ceilometer-agent-notification-container-puppet.yaml
  OS::TripleO::Services::ComputeCeilometerAgent: /usr/share/openstack-tripleo-heat-
templates/deployment/ceilometer/ceilometer-agent-compute-container-puppet.yaml
  OS::TripleO::Services::GnocchiApi: /usr/share/openstack-tripleo-heat-
templates/deployment/gnocchi/gnocchi-api-container-puppet.yaml
  OS::TripleO::Services::GnocchiMetricd: /usr/share/openstack-tripleo-heat-
templates/deployment/gnocchi/gnocchi-metricd-container-puppet.yaml
  OS::TripleO::Services::GnocchiStatsd: /usr/share/openstack-tripleo-heat-
templates/deployment/gnocchi/gnocchi-statsd-container-puppet.yaml
  OS::TripleO::Services::HeatApi: /usr/share/openstack-tripleo-heat-
templates/deployment/heat/heat-api-container-puppet.yaml
  OS::TripleO::Services::HeatApiCfn: /usr/share/openstack-tripleo-heat-
```

```

templates/deployment/heat/heat-api-cfn-container-puppet.yaml
  OS::TripleO::Services::HeatApiCloudwatch: /usr/share/openstack-tripleo-heat-
templates/deployment/heat/heat-api-cloudwatch-disabled-puppet.yaml
  OS::TripleO::Services::HeatEngine: /usr/share/openstack-tripleo-heat-
templates/deployment/heat/heat-engine-container-puppet.yaml
  OS::TripleO::Services::Redis: /usr/share/openstack-tripleo-heat-
templates/deployment/database/redis-pacemaker-puppet.yaml
EOF

```

4. Create an environment template to configure the services required for autoscaling:

```

cat <<EOF > $HOME/templates/autoscaling/parameters-autoscaling.yaml
parameter_defaults:
  NotificationDriver: 'messagingv2'
  GnocchiDebug: false
  CeilometerEnableGnocchi: true
  ManagePipeline: true
  ManageEventPipeline: true

  EventPipelinePublishers:
    - gnocchi://?archive_policy=generic
  PipelinePublishers:
    - gnocchi://?archive_policy=generic

  ManagePolling: true
  ExtraConfig:
    ceilometer::agent::polling::polling_interval: 60
EOF

```

If you use Red Hat Ceph Storage as the data storage back end for the time-series database service, add the following parameters to your **parameters-autoscaling.yaml** file:

```

parameter_defaults:
  GnocchiRbdPoolName: 'metrics'
  GnocchiBackend: 'rbd'

```

You must create the defined archive policy **generic** before you can store metrics. You define this archive policy after the deployment. For more information, see [Section 3.1, “Creating the generic archive policy for autoscaling”](#).

5. Set the **polling_interval** parameter, for example, 60 seconds. The value of the **polling_interval** parameter must match the gnocchi granularity value that you defined when you created the archive policy. For more information, see [Section 3.1, “Creating the generic archive policy for autoscaling”](#).
6. Deploy the overcloud. For more information, see [Section 2.2, “Deploying the overcloud for autoscaling”](#).

2.2. DEPLOYING THE OVERCLOUD FOR AUTOSCALING

You can deploy the overcloud for autoscaling by using director or by using a standalone environment.

Prerequisites

- You have created the environment templates for deploying the services that provide autoscaling capabilities. For more information, see [Section 2.1, “Configuring the overcloud for autoscaling”](#).

Procedure

- [Section 2.2.1, “Deploying the overcloud for autoscaling by using director”](#)
- [Section 2.2.2, “Deploying the overcloud for autoscaling in a standalone environment”](#)

2.2.1. Deploying the overcloud for autoscaling by using director

Use director to deploy the overcloud. If you are using a standalone environment, see [Section 2.2.2, “Deploying the overcloud for autoscaling in a standalone environment”](#).

Prerequisites

- A deployed undercloud. For more information, see [Installing director on the undercloud](#).

Procedure

1. Log in to the undercloud as the **stack** user.
2. Source the **stackrc** undercloud credentials file:


```
[stack@director ~]$ source ~/stackrc
```
3. Add the autoscaling environment files to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
-e [your environment files] \
-e $HOME/templates/autoscaling/parameters-autoscaling.yaml \
-e $HOME/templates/autoscaling/resources-autoscaling.yaml
```

2.2.2. Deploying the overcloud for autoscaling in a standalone environment

To test the environment files in a pre-production environment, you can deploy the overcloud with the services required for autoscaling by using a standalone deployment.



NOTE

This procedure uses example values and commands that you must change to suit a production environment.

If you want to use director to deploy the overcloud for autoscaling, see [Section 2.2.1, “Deploying the overcloud for autoscaling by using director”](#).

Prerequisites

- An all-in-one RHOSP environment has been staged with the python3-tripleoclient. For more information, see [Installing the all-in-one Red Hat OpenStack Platform environment](#).

- An all-in-one RHOSP environment has been staged with the base configuration. For more information, see [Configuring the all-in-one Red Hat OpenStack Platform environment](#).

Procedure

1. Change to the user that manages your overcloud deployments, for example, the **stack** user:

```
[root@standalone ~]# su - stack
```

2. Replace or set the environment variables **\$IP**, **\$NETMASK** and **\$VIP** for the overcloud deployment:

```
$ export IP=192.168.25.2
$ export VIP=192.168.25.3
$ export NETMASK=24
```

3. Deploy the overcloud to test and verify the resource and parameter files:

```
$ sudo openstack tripleo deploy \
  --templates \
  --local-ip=$IP/$NETMASK \
  --control-virtual-ip=$VIP \
  -e /usr/share/openstack-tripleo-heat-templates/environments/standalone/standalone-
tripleo.yaml \
  -r /usr/share/openstack-tripleo-heat-templates/roles/Standalone.yaml \
  -e $HOME/containers-prepare-parameters.yaml \
  -e $HOME/standalone_parameters.yaml \
  -e $HOME/templates/autoscaling/resources-autoscaling.yaml \
  -e $HOME/templates/autoscaling/parameters-autoscaling.yaml \
  --output-dir $HOME \
  --standalone
```

4. Export the **OS_CLOUD** environment variable:

```
$ export OS_CLOUD=standalone
```

Additional resources

- [Director Installation and Usage](#) guide.
- [Standalone Deployment Guide](#).

2.3. VERIFYING THE OVERCLOUD DEPLOYMENT FOR AUTOSCALING

Verify that the autoscaling services are deployed and enabled. Verification output is from a standalone environment, but director-based environments provide similar output.

Prerequisites

- You have deployed the autoscaling services in an existing overcloud using standalone or director. For more information, see [Section 2.2, “Deploying the overcloud for autoscaling”](#).

Procedure

1. Log in to your environment as the **stack** user.
2. For standalone environments set the **OS_CLOUD** environment variable:

```
[stack@standalone ~]$ export OS_CLOUD=standalone
```

3. For director environments, source the **stackrc** undercloud credentials file:

```
[stack@undercloud ~]$ source ~/stackrc
```

Verification

1. Verify that the deployment was successful and ensure that the service API endpoints for autoscaling are available:

```
$ openstack endpoint list --service metric
```

```
+-----+-----+-----+-----+-----+-----+
| ID              | Region | Service Name | Service Type | Enabled | Interface | URL |
+-----+-----+-----+-----+-----+-----+
| 2956a12327b744b29abd4577837b2e6f | regionOne | gnocchi      | metric      | True   | internal  | http://192.168.25.3:8041 |
| 583453c58b064f69af3de3479675051a | regionOne | gnocchi      | metric      | True   | admin     | http://192.168.25.3:8041 |
| fa029da0e2c047fc9d9c50eb6b4876c6 | regionOne | gnocchi      | metric      | True   | public    | http://192.168.25.3:8041 |
+-----+-----+-----+-----+-----+-----+
```

```
$ openstack endpoint list --service alarming
```

```
+-----+-----+-----+-----+-----+-----+
| ID              | Region | Service Name | Service Type | Enabled | Interface | URL |
+-----+-----+-----+-----+-----+-----+
| 08c70ec137b44ed68590f4d5c31162bb | regionOne | aodh         | alarming     | True   | internal  | http://192.168.25.3:8042 |
| 194042887f3d4eb4b638192a0fe60996 | regionOne | aodh         | alarming     | True   | admin     | http://192.168.25.3:8042 |
| 2604b693740245ed8960b31dfea1f963 | regionOne | aodh         | alarming     | True   | public    | http://192.168.25.3:8042 |
+-----+-----+-----+-----+-----+-----+
```

```
$ openstack endpoint list --service orchestration
```

```
+-----+-----+-----+-----+-----+-----+
| ID              | Region | Service Name | Service Type | Enabled | Interface | URL |
+-----+-----+-----+-----+-----+-----+
```

```
| 00966a24dd4141349e12680307c11848 | regionOne | heat      | orchestration | True |
admin   | http://192.168.25.3:8004/v1/%(tenant_id)s |
| 831e411bb6d44f6db9f5103d659f901e | regionOne | heat      | orchestration | True |
public  | http://192.168.25.3:8004/v1/%(tenant_id)s |
| d5be22349add43ae95be4284a42a4a60 | regionOne | heat      | orchestration | True |
internal | http://192.168.25.3:8004/v1/%(tenant_id)s |
+-----+-----+-----+-----+-----+-----+
-----+
```

2. Verify that the services are running on the overcloud:

```
$ sudo podman ps --filter=name='heat|gnocchi|ceilometer|aodh'
CONTAINER ID IMAGE COMMAND CREATED
STATUS PORTS NAMES
31e75d62367f registry.redhat.io/rhosp-rhel9/openstack-aodh-api:17.0 kolla_start
27 minutes ago Up 27 minutes ago (healthy) aodh_api
77acf3487736 registry.redhat.io/rhosp-rhel9/openstack-aodh-listener:17.0 kolla_start
27 minutes ago Up 27 minutes ago (healthy) aodh_listener
29ec47b69799 registry.redhat.io/rhosp-rhel9/openstack-aodh-evaluator:17.0
kolla_start 27 minutes ago Up 27 minutes ago (healthy) aodh_evaluator
43efaa86c769 registry.redhat.io/rhosp-rhel9/openstack-aodh-notifier:17.0 kolla_start
27 minutes ago Up 27 minutes ago (healthy) aodh_notifier
0ac8cb2c7470 registry.redhat.io/rhosp-rhel9/openstack-aodh-api:17.0 kolla_start
27 minutes ago Up 27 minutes ago (healthy) aodh_api_cron
31b55e091f57 registry.redhat.io/rhosp-rhel9/openstack-ceilometer-central:17.0
kolla_start 27 minutes ago Up 27 minutes ago (healthy) ceilometer_agent_central
5f61331a17d8 registry.redhat.io/rhosp-rhel9/openstack-ceilometer-compute:17.0
kolla_start 27 minutes ago Up 27 minutes ago (healthy) ceilometer_agent_compute
7c5ef75d8f1b registry.redhat.io/rhosp-rhel9/openstack-ceilometer-notification:17.0
kolla_start 27 minutes ago Up 27 minutes ago (healthy)
ceilometer_agent_notification
88fa57cc1235 registry.redhat.io/rhosp-rhel9/openstack-gnocchi-api:17.0 kolla_start
23 minutes ago Up 23 minutes ago (healthy) gnocchi_api
0f05a58197d5 registry.redhat.io/rhosp-rhel9/openstack-gnocchi-metricd:17.0
kolla_start 23 minutes ago Up 23 minutes ago (healthy) gnocchi_metricd
6d806c285500 registry.redhat.io/rhosp-rhel9/openstack-gnocchi-statsd:17.0
kolla_start 23 minutes ago Up 23 minutes ago (healthy) gnocchi_statsd
7c02cac34c69 registry.redhat.io/rhosp-rhel9/openstack-heat-api:17.0 kolla_start
27 minutes ago Up 27 minutes ago (healthy) heat_api_cron
d3903df545ce registry.redhat.io/rhosp-rhel9/openstack-heat-api:17.0 kolla_start
27 minutes ago Up 27 minutes ago (healthy) heat_api
db1d33506e3d registry.redhat.io/rhosp-rhel9/openstack-heat-api-cfn:17.0 kolla_start
27 minutes ago Up 27 minutes ago (healthy) heat_api_cfn
051446294c70 registry.redhat.io/rhosp-rhel9/openstack-heat-engine:17.0 kolla_start
27 minutes ago Up 27 minutes ago (healthy) heat_engine
```

3. Verify that the time-series database service is available:

```
$ openstack metric status --fit-width
+-----+-----+
-----+
| Field                                | Value |
|                                     |       |
+-----+-----+
-----+
```

```

| metricd/processors | ['standalone-80.general.local.0.a94fbf77-1ac0-
49ed-bfe2-a89f014fde01',
| |
| | 'standalone-80.general.local.3.28ca78d7-a80e-4515-8060-
233360b410eb',
| |
| | 'standalone-80.general.local.1.7e8b5a5b-2ca1-49be-bc22-
25f51d67c00a',
| |
| | 'standalone-80.general.local.2.3c4fe59e-23cd-4742-833d-
42ff0a4cb692']
| storage/number of metric having measures to process | 0
|
| storage/total number of measures to process | 0
|
+-----+-----+
-----+

```

CHAPTER 3. USING THE HEAT SERVICE FOR AUTOSCALING

After you deploy the services required to provide autoscaling in the overcloud, you must configure the overcloud environment so that the Orchestration service (heat) can manage instances for autoscaling.

Prerequisites

- A deployed overcloud. For more information, see [Section 2.2, “Deploying the overcloud for autoscaling”](#).

Procedure

- [Section 3.1, “Creating the generic archive policy for autoscaling”](#)
- [Section 3.2, “Configuring a heat template for automatically scaling instances”](#)
- [Section 3.3, “Preparing the standalone deployment for autoscaling”](#)
- [Section 3.4, “Creating the stack deployment for autoscaling”](#)

3.1. CREATING THE GENERIC ARCHIVE POLICY FOR AUTOSCALING

After you deploy the services for autoscaling in the overcloud, you must configure the overcloud environment so that the Orchestration service (heat) can manage the instances for autoscaling.

Prerequisites

- You have deployed an overcloud that has autoscaling services. For more information, see [Section 2.1, “Configuring the overcloud for autoscaling”](#).

Procedure

1. Log in to your environment as the **stack** user.
2. For standalone environments, set the **OS_CLOUD** environment variable:

```
[stack@standalone ~]$ export OS_CLOUD=standalone
```

3. For director environments source the **stackrc** file:

```
[stack@undercloud ~]$ source ~/stackrc
```

4. Create the archive policy defined in **\$HOME/templates/autoscaling/parameters-autoscaling.yaml**:

```
$ openstack metric archive-policy create generic \
  --back-window 0 \
  --definition timespan:'4:00:00',granularity:'0:01:00',points:240 \
  --aggregation-method 'rate:mean' \
  --aggregation-method 'mean'
```

Verification

- Verify that the archive policy was created:

```
$ openstack metric archive-policy show generic
+-----+-----+
| Field          | Value                                     |
+-----+-----+
| aggregation_methods | mean, rate:mean                         |
| back_window       | 0                                       |
| definition        | - timespan: 4:00:00, granularity: 0:01:00, points: 240 |
| name             | generic                               |
+-----+-----+
```

3.2. CONFIGURING A HEAT TEMPLATE FOR AUTOMATICALLY SCALING INSTANCES

You can configure an Orchestration service (heat) template to create the instances, and configure alarms that create and scale instances when triggered.



NOTE

This procedure uses example values that you must change to suit your environment.

Prerequisites

- You have deployed the overcloud with the autoscaling services. For more information, see [Section 2.2, “Deploying the overcloud for autoscaling”](#).
- You have configured the overcloud with an archive policy for autoscaling telemetry storage. For more information, see [Section 3.1, “Creating the generic archive policy for autoscaling”](#).

Procedure

1. Log in to your environment as the **stack** user.

```
$ source ~/stackrc
```

2. Create a directory to hold the instance configuration for the autoscaling group:

```
$ mkdir -p $HOME/templates/autoscaling/vnf/
```

3. Create an instance configuration template, for example, **\$HOME/templates/autoscaling/vnf/instance.yaml**.

4. Add the following configuration to your **instance.yaml** file:

```
cat <<EOF > $HOME/templates/autoscaling/vnf/instance.yaml
heat_template_version: wallaby
description: Template to control scaling of VNF instance

parameters:
  metadata:
    type: json
  image:
```

```

    type: string
    description: image used to create instance
    default: fedora36
  flavor:
    type: string
    description: instance flavor to be used
    default: m1.small
  key_name:
    type: string
    description: keypair to be used
    default: default
  network:
    type: string
    description: project network to attach instance to
    default: private
  external_network:
    type: string
    description: network used for floating IPs
    default: public

  resources:
    vnf:
      type: OS::Nova::Server
      properties:
        flavor: {get_param: flavor}
        key_name: {get_param: key_name}
        image: { get_param: image }
        metadata: { get_param: metadata }
        networks:
          - port: { get_resource: port }

    port:
      type: OS::Neutron::Port
      properties:
        network: {get_param: network}
        security_groups:
          - basic

    floating_ip:
      type: OS::Neutron::FloatingIP
      properties:
        floating_network: {get_param: external_network }

    floating_ip_assoc:
      type: OS::Neutron::FloatingIPAssociation
      properties:
        floatingip_id: { get_resource: floating_ip }
        port_id: { get_resource: port }
  EOF

```

- The **parameters** parameter defines the custom parameters for this new resource.
- The **vnf** sub-parameter of the **resources** parameter defines the name of the custom sub-resource referred to in the **OS::Heat::AutoScalingGroup**, for example, **OS::Nova::Server::VNF**.

5. Create the resource to reference in the heat template:

```
$ cat <<EOF > $HOME/templates/autoscaling/vnf/resources.yaml
resource_registry:
  "OS::Nova::Server::VNF": $HOME/templates/autoscaling/vnf/instance.yaml
EOF
```

6. Create the deployment template for heat to control instance scaling:

```
$ cat <<EOF > $HOME/templates/autoscaling/vnf/template.yaml
heat_template_version: wallaby
description: Example auto scale group, policy and alarm
resources:
  scaleup_group:
    type: OS::Heat::AutoScalingGroup
    properties:
      max_size: 3
      min_size: 1
      #desired_capacity: 1
    resource:
      type: OS::Nova::Server::VNF
      properties:
        metadata: {"metering.server_group": {get_param: "OS::stack_id"}}

  scaleup_policy:
    type: OS::Heat::ScalingPolicy
    properties:
      adjustment_type: change_in_capacity
      auto_scaling_group_id: { get_resource: scaleup_group }
      cooldown: 60
      scaling_adjustment: 1

  scaledown_policy:
    type: OS::Heat::ScalingPolicy
    properties:
      adjustment_type: change_in_capacity
      auto_scaling_group_id: { get_resource: scaleup_group }
      cooldown: 60
      scaling_adjustment: -1

  cpu_alarm_high:
    type: OS::Aodh::GnocchiAggregationByResourcesAlarm
    properties:
      description: Scale up instance if CPU > 50%
      metric: cpu
      aggregation_method: rate:mean
      granularity: 60
      evaluation_periods: 3
      threshold: 60000000000.0
      resource_type: instance
      comparison_operator: gt
      alarm_actions:
        - str_replace:
            template: trust+url
            params:
              url: {get_attr: [scaleup_policy, signal_url]}
```

```

    query:
      list_join:
        - "
          - - {'=': {server_group: {get_param: "OS::stack_id"}}}

cpu_alarm_low:
  type: OS::Aodh::GnocchiAggregationByResourcesAlarm
  properties:
    description: Scale down instance if CPU < 20%
    metric: cpu
    aggregation_method: rate:mean
    granularity: 60
    evaluation_periods: 3
    threshold: 24000000000.0
    resource_type: instance
    comparison_operator: Lt
    alarm_actions:
      - str_replace:
          template: trust+url
          params:
            url: {get_attr: [scaledown_policy, signal_url]}
    query:
      list_join:
        - "
          - - {'=': {server_group: {get_param: "OS::stack_id"}}}

outputs:
  scaleup_policy_signal_url:
    value: {get_attr: [scaleup_policy, alarm_url]}

  scaledown_policy_signal_url:
    value: {get_attr: [scaledown_policy, alarm_url]}
EOF

```



NOTE

Outputs on the stack are informational and are not referenced in the ScalingPolicy or AutoScalingGroup. To view the outputs, use the **openstack stack show <stack_name>** command.

Additional resources

- [Advanced Overcloud Customization](#) guide.

3.3. PREPARING THE STANDALONE DEPLOYMENT FOR AUTOSCALING

To test the deployment of a stack for an autoscaled instance in a pre-production environment, you can deploy the stack by using a standalone deployment. You can use this procedure to test the deployment with a standalone environment. In a production environment, the deployment commands are different.

Procedure

1. Log in to your environment as the **stack** user.

2. Set the **OS_CLOUD** environment variable:

```
[stack@standalone ~]$ export OS_CLOUD=standalone
```

3. Configure the cloud to allow deployment of a simulated VNF workload that uses the Fedora 36 cloud image with attached private and public network interfaces. This example is a working configuration that uses a standalone deployment:

```
$ export GATEWAY=192.168.25.1
$ export STANDALONE_HOST=192.168.25.2
$ export PUBLIC_NETWORK_CIDR=192.168.25.0/24
$ export PRIVATE_NETWORK_CIDR=192.168.100.0/24
$ export PUBLIC_NET_START=192.168.25.3
$ export PUBLIC_NET_END=192.168.25.254
$ export DNS_SERVER=1.1.1.1
```

4. Create the flavor:

```
$ openstack flavor create --ram 2048 --disk 10 --vcpu 2 --public m1.small
```

5. Download and import the Fedora 36 x86_64 cloud image:

```
$ curl -L
'https://download.fedoraproject.org/pub/fedora/linux/releases/36/Cloud/x86_64/images/Fedora-Cloud-Base-36-1.5.x86_64.qcow2' -o $HOME/fedora36.qcow2
```

```
$ openstack image create fedora36 --container-format bare --disk-format qcow2 --public --file
$HOME/fedora36.qcow2
```

6. Generate and import the public key:

```
$ ssh-keygen -f $HOME/.ssh/id_rsa -q -N "" -t rsa -b 2048
```

```
$ openstack keypair create --public-key $HOME/.ssh/id_rsa.pub default
```

7. Create the **basic** security group that allows SSH, ICMP, and DNS protocols:

```
$ openstack security group create basic
```

```
$ openstack security group rule create basic --protocol tcp --dst-port 22:22 --remote-ip
0.0.0.0/0
```

```
$ openstack security group rule create --protocol icmp basic
```

```
$ openstack security group rule create --protocol udp --dst-port 53:53 basic
```

8. Create the external network (public):

```
$ openstack network create --external --provider-physical-network datacentre --provider-
network-type flat public
```

9. Create the private network:

```
$ openstack network create --internal private
```

```
openstack subnet create public-net \  
  --subnet-range $PUBLIC_NETWORK_CIDR \  
  --no-dhcp \  
  --gateway $GATEWAY \  
  --allocation-pool start=$PUBLIC_NET_START,end=$PUBLIC_NET_END \  
  --network public
```

```
$ openstack subnet create private-net \  
  --subnet-range $PRIVATE_NETWORK_CIDR \  
  --network private
```

10. Create the router:

```
$ openstack router create vrouter
```

```
$ openstack router set vrouter --external-gateway public
```

```
$ openstack router add subnet vrouter private-net
```

Additional resources

- [Standalone Deployment Guide](#).

3.4. CREATING THE STACK DEPLOYMENT FOR AUTOSCALING

Create the stack deployment for the worked VNF autoscaling example.

Prerequisites

- [Section 3.2, “Configuring a heat template for automatically scaling instances”](#)

Procedure

1. Create the stack:

```
$ openstack stack create \  
  -t $HOME/templates/autoscaling/vnf/template.yaml \  
  -e $HOME/templates/autoscaling/vnf/resources.yaml \  
  vnf
```

Verification

1. Verify that the stack was created successfully:

```
$ openstack stack show vnf -c id -c stack_status  
+-----+-----+  
| Field  | Value                                |
```

```
+-----+-----+
| id      | cb082cbd-535e-4779-84b0-98925e103f5e |
| stack_status | CREATE_COMPLETE          |
+-----+-----+
```

2. Verify that the stack resources were created, including alarms, scaling policies, and the autoscaling group:

```
$ export STACK_ID=$(openstack stack show vnf -c id -f value)
```

```
$ openstack stack resource list $STACK_ID
+-----+-----+-----+-----+-----+-----+
| resource_name | physical_resource_id | resource_type | resource_status | updated_time |
+-----+-----+-----+-----+-----+-----+
| cpu_alarm_high | d72d2e0d-1888-4f89-b888-02174c48e463 | OS::Aodh::GnocchiAggregationByResourcesAlarm | CREATE_COMPLETE | 2022-10-06T23:08:37Z |
| scaleup_policy | 1c4446b7242e479090bef4b8075df9d4 | OS::Heat::ScalingPolicy | CREATE_COMPLETE | 2022-10-06T23:08:37Z |
| cpu_alarm_low | b9c04ef4-8b57-4730-af03-1a71c3885914 | OS::Aodh::GnocchiAggregationByResourcesAlarm | CREATE_COMPLETE | 2022-10-06T23:08:37Z |
| scaledown_policy | a5af7faf5a1344849c3425cb2c5f18db | OS::Heat::ScalingPolicy | CREATE_COMPLETE | 2022-10-06T23:08:37Z |
| scaleup_group | 9609f208-6d50-4b8f-836e-b0222dc1e0b1 | OS::Heat::AutoScalingGroup | CREATE_COMPLETE | 2022-10-06T23:08:37Z |
+-----+-----+-----+-----+-----+-----+
```

3. Verify that an instance was launched by the stack creation:

```
$ openstack server list --long | grep $STACK_ID
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7 | vn-dvaxcqb-6bqh2qd2pif-hicmkm5dzjug-vnf-ywrydc5wqjjc | ACTIVE | None | Running | private=192.168.100.61, 192.168.25.99 | fedora36 | a6aa7b11-1b99-4c62-a43b-d0b7c77f4b72 | m1.small | 5cd46fec-50c2-43d5-89e8-ed3fa7660852 | nova | standalone-80.localdomain | metering.server_group='cb082cbd-535e-4779-84b0-98925e103f5e' |
```

4. Verify that the alarms were created for the stack:
 - a. List the alarm IDs. The state of the alarms might reside in the **insufficient data** state for a period of time. The minimal period of time is the polling interval of the data collection and data storage granularity setting:

```
$ openstack alarm list
+-----+-----+-----+-----+-----+-----+
| alarm_id | type | name | state | severity | enabled |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
```

```
| b9c04ef4-8b57-4730-af03-1a71c3885914 |
gnocchi_aggregation_by_resources_threshold | vnf-cpu_alarm_low-pve5eal6ykst |
alarm | low | True |
| d72d2e0d-1888-4f89-b888-02174c48e463 |
gnocchi_aggregation_by_resources_threshold | vnf-cpu_alarm_high-5xx7qvfsurxe | ok
| low | True |
+-----+-----+-----+-----+
-----+-----+-----+-----+
```

- b. List the resources for the stack and note the **physical_resource_id** values for the **cpu_alarm_high** and **cpu_alarm_low** resources.

```
$ openstack stack resource list $STACK_ID
+-----+-----+-----+-----+
+-----+-----+
| resource_name | physical_resource_id | resource_type |
resource_status | updated_time |
+-----+-----+-----+-----+
+-----+-----+
| cpu_alarm_high | d72d2e0d-1888-4f89-b888-02174c48e463 |
OS::Aodh::GnocchiAggregationByResourcesAlarm | CREATE_COMPLETE | 2022-10-
06T23:08:37Z |
| scaleup_policy | 1c4446b7242e479090bef4b8075df9d4 | OS::Heat::ScalingPolicy
| CREATE_COMPLETE | 2022-10-06T23:08:37Z |
| cpu_alarm_low | b9c04ef4-8b57-4730-af03-1a71c3885914 |
OS::Aodh::GnocchiAggregationByResourcesAlarm | CREATE_COMPLETE | 2022-10-
06T23:08:37Z |
| scaledown_policy | a5af7faf5a1344849c3425cb2c5f18db | OS::Heat::ScalingPolicy
| CREATE_COMPLETE | 2022-10-06T23:08:37Z |
| scaleup_group | 9609f208-6d50-4b8f-836e-b0222dc1e0b1 |
OS::Heat::AutoScalingGroup | CREATE_COMPLETE | 2022-10-
06T23:08:37Z |
+-----+-----+-----+-----+
+-----+-----+
+-----+-----+
```

The value of the **physical_resource_id** must match the **alarm_id** in the output of the **openstack alarm list** command.

5. Verify that metric resources exist for the stack. Set the value of the **server_group** query to the stack ID:

```
$ openstack metric resource search --sort-column launched_at -c id -c display_name -c
launched_at -c deleted_at --type instance server_group="$STACK_ID"
+-----+-----+-----+-----+
+-----+-----+
| id | display_name | launched_at |
| deleted_at |
+-----+-----+-----+-----+
+-----+-----+
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7 | vn-dvaxcqb-6bqh2qd2fpif-hicmkm5dzjug-vnf-
ywrydc5wqjjc | 2022-10-06T23:09:28.496566+00:00 | None |
+-----+-----+-----+-----+
+-----+-----+
```

6. Verify that measurements exist for the instance resources created through the stack:

```
$ openstack metric aggregates --resource-type instance --sort-column timestamp '(metric
cpu rate:mean)' server_group="$STACK_ID"
+-----+-----+-----+-----+
+
| name                                | timestamp          | granularity | value |
+-----+-----+-----+-----+
+
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:11:00+00:00 | 60.0 | 69470000000.0 |
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:12:00+00:00 | 60.0 | 81060000000.0 |
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:13:00+00:00 | 60.0 | 82840000000.0 |
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:14:00+00:00 | 60.0 | 66660000000.0 |
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:15:00+00:00 | 60.0 | 73600000000.0 |
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:16:00+00:00 | 60.0 | 31500000000.0 |
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:17:00+00:00 | 60.0 | 27600000000.0 |
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:18:00+00:00 | 60.0 | 34700000000.0 |
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:19:00+00:00 | 60.0 | 27700000000.0 |
| 62e1b27c-8d9d-44a5-a0f0-80e7e6d437c7/cpu/rate:mean | 2022-10-06T23:20:00+00:00 | 60.0 | 27000000000.0 |
+-----+-----+-----+-----+
+
```

CHAPTER 4. TESTING AND TROUBLESHOOTING AUTOSCALING

Use the Orchestration service (heat) to automatically scale instances up and down based on threshold definitions. To troubleshoot your environment, you can look for errors in the log files and history records.

4.1. TESTING AUTOMATIC SCALING UP OF INSTANCES

You can use the Orchestration service (heat) to scale instances automatically based on the **cpu_alarm_high** threshold definition. When the CPU use reaches a value defined in the **threshold** parameter, another instance starts up to balance the load. The **threshold** value in the **template.yaml** file is set to 80%.

Procedure

1. Log in to the host environment as the **stack** user.
2. For standalone environments set the **OS_CLOUD** environment variable:

```
[stack@standalone ~]$ export OS_CLOUD=standalone
```

3. For director environments source the **stackrc** file:

```
[stack@undercloud ~]$ source ~/stackrc
```

4. Log in to the instance:

```
$ ssh -i ~/mykey.pem cirros@192.168.122.8
```

5. Run multiple **dd** commands to generate the load:

```
[instance ~]$ sudo dd if=/dev/zero of=/dev/null &
[instance ~]$ sudo dd if=/dev/zero of=/dev/null &
[instance ~]$ sudo dd if=/dev/zero of=/dev/null &
```

6. Exit from the running instance and return to the host.
7. After you run the **dd** commands, you can expect to have 100% CPU use in the instance. Verify that the alarm has been triggered:

```
$ openstack alarm list
+-----+-----+-----+-----+-----+
| alarm_id | type | name | state |
| severity | enabled | | |
+-----+-----+-----+-----+
| 022f707d-46cc-4d39-a0b2-afd2fc7ab86a | gnocchi_aggregation_by_resources_threshold |
example-cpu_alarm_high-odj77qpbld7j | alarm | low | True |
| 46ed2c50-e05a-44d8-b6f6-f1ebd83af913 | gnocchi_aggregation_by_resources_threshold |
example-cpu_alarm_low-m37jvnm56x2t | ok | low | True |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

- After approximately 60 seconds, Orchestration starts another instance and adds it to the group. To verify that an instance has been created, enter the following command:

```
$ openstack server list
+-----+-----+-----+-----+-----+
| ID                               | Name                               | Status | Task State | Power
State | Networks                               |
+-----+-----+-----+-----+-----+
| 477ee1af-096c-477c-9a3f-b95b0e2d4ab5 | ex-3gax-4urpikl5koff-yrxk3zxzfmpr-server-
2hde4tp4trnk | ACTIVE | -      | Running | internal1=10.10.10.13, 192.168.122.17 |
| e1524f65-5be6-49e4-8501-e5e5d812c612 | ex-3gax-5f3a4og5cwn2-png47w3u2vjd-server-
vaajhuv4mj3j | ACTIVE | -      | Running | internal1=10.10.10.9, 192.168.122.8 |
+-----+-----+-----+-----+-----+
```

- After another short period of time, observe that the Orchestration service has autoscaled to three instances. The configuration is set to a maximum of three instances. Verify there are three instances:

```
$ openstack server list
+-----+-----+-----+-----+-----+
| ID                               | Name                               | Status | Task State | Power
State | Networks                               |
+-----+-----+-----+-----+-----+
| 477ee1af-096c-477c-9a3f-b95b0e2d4ab5 | ex-3gax-4urpikl5koff-yrxk3zxzfmpr-server-
2hde4tp4trnk | ACTIVE | -      | Running | internal1=10.10.10.13, 192.168.122.17 |
| e1524f65-5be6-49e4-8501-e5e5d812c612 | ex-3gax-5f3a4og5cwn2-png47w3u2vjd-server-
vaajhuv4mj3j | ACTIVE | -      | Running | internal1=10.10.10.9, 192.168.122.8 |
| 6c88179e-c368-453d-a01a-555eae8cd77a | ex-3gax-fvxz3tr63j4o-36fhftuja3bw-server-
rhl4sqkjuy5p | ACTIVE | -      | Running | internal1=10.10.10.5, 192.168.122.5 |
+-----+-----+-----+-----+-----+
```

4.2. TESTING AUTOMATIC SCALING DOWN OF INSTANCES

You can use the Orchestration service (heat) to automatically scale down instances based on the **cpu_alarm_low** threshold. In this example, the instances are scaled down when CPU use is below 5%.

Procedure

- From within the workload instance, terminate the running **dd** processes and observe Orchestration begin to scale the instances back down.

```
$ killall dd
```

- Log in to the host environment as the **stack** user.
- For standalone environments set the **OS_CLOUD** environment variable:

```
[stack@standalone ~]$ export OS_CLOUD=standalone
```

- For director environments source the **stackrc** file:

```
[stack@undercloud ~]$ source ~/stackrc
```

- When you stop the **dd** processes, this triggers the **cpu_alarm_low event** alarm. As a result, Orchestration begins to automatically scale down and remove the instances. Verify that the corresponding alarm has triggered:

```
$ openstack alarm list
+-----+-----+-----+-----+
| alarm_id           | type           | name           | state |
| severity | enabled |
+-----+-----+-----+-----+
| 022f707d-46cc-4d39-a0b2-afd2fc7ab86a | gnocchi_aggregation_by_resources_threshold |
example-cpu_alarm_high-odj77qpbl7j | ok | low | True |
| 46ed2c50-e05a-44d8-b6f6-f1ebd83af913 | gnocchi_aggregation_by_resources_threshold |
example-cpu_alarm_low-m37jvnm56x2t | alarm | low | True |
+-----+-----+-----+-----+
```

After a few minutes, Orchestration continually reduce the number of instances to the minimum value defined in the **min_size** parameter of the **scaleup_group** definition. In this scenario, the **min_size** parameter is set to **1**.

4.3. TROUBLESHOOTING FOR AUTOSCALING

If your environment is not working properly, you can look for errors in the log files and history records.

Procedure

- Log in to the host environment as the **stack** user.
- For standalone environments set the **OS_CLOUD** environment variable:

```
[stack@standalone ~]$ export OS_CLOUD=standalone
```

- For director environments source the **stackrc** file:

```
[stack@undercloud ~]$ source ~/stackrc
```

- To retrieve information on state transitions, list the stack event records:

```
$ openstack stack event list example
2017-03-06 11:12:43Z [example]: CREATE_IN_PROGRESS Stack CREATE started
2017-03-06 11:12:43Z [example.scaleup_group]: CREATE_IN_PROGRESS state changed
2017-03-06 11:13:04Z [example.scaleup_group]: CREATE_COMPLETE state changed
2017-03-06 11:13:04Z [example.scaledown_policy]: CREATE_IN_PROGRESS state
changed
2017-03-06 11:13:05Z [example.scaleup_policy]: CREATE_IN_PROGRESS state changed
```

```

2017-03-06 11:13:05Z [example.scaledown_policy]: CREATE_COMPLETE state changed
2017-03-06 11:13:05Z [example.scaleup_policy]: CREATE_COMPLETE state changed
2017-03-06 11:13:05Z [example.cpu_alarm_low]: CREATE_IN_PROGRESS state changed
2017-03-06 11:13:05Z [example.cpu_alarm_high]: CREATE_IN_PROGRESS state changed
2017-03-06 11:13:06Z [example.cpu_alarm_low]: CREATE_COMPLETE state changed
2017-03-06 11:13:07Z [example.cpu_alarm_high]: CREATE_COMPLETE state changed
2017-03-06 11:13:07Z [example]: CREATE_COMPLETE Stack CREATE completed
successfully
2017-03-06 11:19:34Z [example.scaleup_policy]: SIGNAL_COMPLETE alarm state
changed from alarm to alarm (Remaining as alarm due to 1 samples outside threshold, most
recent: 95.4080102993)
2017-03-06 11:25:43Z [example.scaleup_policy]: SIGNAL_COMPLETE alarm state
changed from alarm to alarm (Remaining as alarm due to 1 samples outside threshold, most
recent: 95.8869217299)
2017-03-06 11:33:25Z [example.scaledown_policy]: SIGNAL_COMPLETE alarm state
changed from ok to alarm (Transition to alarm due to 1 samples outside threshold, most
recent: 2.73931707966)
2017-03-06 11:39:15Z [example.scaledown_policy]: SIGNAL_COMPLETE alarm state
changed from alarm to alarm (Remaining as alarm due to 1 samples outside threshold, most
recent: 2.78110858552)

```

5. Read the alarm history log:

```

$ openstack alarm-history show 022f707d-46cc-4d39-a0b2-afd2fc7ab86a
+-----+-----+-----+
| timestamp          | type          | detail          |
| event_id           |               |                 |
+-----+-----+-----+
| 2017-03-06T11:32:35.510000 | state transition | {"transition_reason": "Transition to ok due to 1 samples inside threshold, most recent: 25e0e70b-3eda-466e-abac-42d9cf67e704 | 2.73931707966", "state": "ok"} |
| 2017-03-06T11:17:35.403000 | state transition | {"transition_reason": "Transition to alarm due to 1 samples outside threshold, most recent: 8322f62c-0d0a-4dc0-9279-435510f81039 | 95.0964497325", "state": "alarm"} |
| 2017-03-06T11:15:35.723000 | state transition | {"transition_reason": "Transition to ok due to 1 samples inside threshold, most recent: 1503bd81-7eba-474e-b74e-ded8a7b630a1 | 3.59330523447", "state": "ok"} |
| 2017-03-06T11:13:06.413000 | creation        | {"alarm_actions": ["trust+http://fca6e27e3d524ed68abdc0fd576aa848:delete@192.168.122.126:8004/v1/fd | 224f15c0-b6f1-4690-9a22-0c1d236e65f6 | 1c345135be4ee587fef424c241719d/stacks/example/d9ef59ed-b8f8-4e90-bd9b-ae87e73ef6e2/resources/scaleup_policy/signal"], "user_id": "a85f83b7f7784025b6acdc06ef0a8fd8", "name": "example-cpu_alarm_high-odj77qpbld7j", "state": "insufficient data", "timestamp": "2017-03-06T11:13:06.413455", "description": "Scale up if

```

```

CPU > 80%", "enabled": true,
|
| "state_timestamp": "2017-03-06T11:13:06.413455", "rule":
| {"evaluation_periods": 1, "metric":
| "cpu_util", "aggregation_method": "mean", "granularity": 300,
| "threshold": 80.0, "query": "{\n=\n":
| {"server_group\": \"d9ef59ed-b8f8-4e90-bd9b-
| ae87e73ef6e2\"}}", "comparison_operator": "gt",
| "resource_type": "instance", "alarm_id": "022f707d-46cc-
| 4d39-a0b2-afd2fc7ab86a",
| "time_constraints": [], "insufficient_data_actions": null,
| "repeat_actions": true, "ok_actions":
| null, "project_id": "fd1c345135be4ee587fef424c241719d",
| "type":
| "gnocchi_aggregation_by_resources_threshold", "severity":
| "low"}
+-----+-----+-----+
+-----+

```

6. To view the records of scale-out or scale-down operations that heat collects for the existing stack, you can use the **awk** command to parse the **heat-engine.log**:

```

$ awk '/Stack UPDATE started/,/Stack CREATE completed successfully/ {print $0}'
/var/log/containers/heat/heat-engine.log

```

7. To view aodh-related information, examine the **evaluator.log**:

```

$ grep -i alarm /var/log/containers/aodh/evaluator.log | grep -i transition

```

4.4. USING CPU TELEMETRY VALUES FOR AUTOSCALING THRESHOLD WHEN USING RATE:MEAN AGGREGATION

When using the **OS::Heat::Autoscaling** heat orchestration template (HOT) and setting a threshold value for CPU, the value is expressed in nanoseconds of CPU time which is a dynamic value based on the number of virtual CPUs allocated to the instance workload. In this reference guide we'll explore how to calculate and express the CPU nanosecond value as a percentage when using the Gnocchi **rate:mean** aggregation method.

4.4.1. Calculating CPU telemetry values as a percentage

CPU telemetry is stored in Gnocchi (OpenStack time-series data store) as CPU utilization in nanoseconds. When using CPU telemetry to define autoscaling thresholds it is useful to express the values as a percentage of CPU utilization since that is more natural when defining the threshold values. When defining the scaling policies used as part of an autoscaling group, we can take our desired threshold defined as a percentage and calculate the required threshold value in nanoseconds which is used in the policy definitions.

Value (ns)	Granularity (s)	Percentage
600000000000	60	100
540000000000	60	90

Value (ns)	Granularity (s)	Percentage
480000000000	60	80
420000000000	60	70
360000000000	60	60
300000000000	60	50
240000000000	60	40
180000000000	60	30
120000000000	60	20
60000000000	60	10

4.4.2. Displaying instance workload vCPU as a percentage

You can display the gnocchi-stored CPU telemetry data as a percentage rather than the nanosecond values for instances by using the **openstack metric aggregates** command.

Prerequisites

- Create a heat stack using the autoscaling group resource that results in an instance workload.

Procedure

1. Login to your OpenStack environment as the cloud administrator.
2. Retrieve the ID of the autoscaling group heat stack:

```
$ openstack stack show vnf -c id -c stack_status
+-----+-----+
| Field   | Value                                     |
+-----+-----+
| id      | e0a15cee-34d1-418a-ac79-74ad07585730 |
| stack_status | CREATE_COMPLETE                       |
+-----+-----+
```

3. Set the value of the stack ID to an environment variable:

```
$ export STACK_ID=$(openstack stack show vnf -c id -f value)
```

4. Return the metrics as an aggregate by resource type instance (server ID) with the value calculated as a percentage. The aggregate is returned as a value of nanoseconds of CPU time. We divide that number by 1000000000 to get the value in seconds. We then divide the value by our granularity, which in this example is 60 seconds. That value is then converted to a

percentage by multiplying by 100. Finally, we divide the total value by the number of vCPU provided by the flavor assigned to the instance, in this example a value of 2 vCPU, providing us a value expressed as a percentage of CPU time:

```
$ openstack metric aggregates --resource-type instance --sort-column timestamp --sort-
descending '/ ( * (/ (/ (metric cpu rate:mean) 1000000000) 60) 100) 2)'
server_group="$STACK_ID"
```

name	timestamp	granularity	value
61bfb555-9efb-46f1-8559-08dec90f94ed/cpu/rate:mean	2022-11-07T21:03:00+00:00	60.0	3.158333333333333
61bfb555-9efb-46f1-8559-08dec90f94ed/cpu/rate:mean	2022-11-07T21:02:00+00:00	60.0	2.633333333333333
199b0cb9-6ed6-4410-9073-0fb2e7842b65/cpu/rate:mean	2022-11-07T21:02:00+00:00	60.0	2.533333333333333
61bfb555-9efb-46f1-8559-08dec90f94ed/cpu/rate:mean	2022-11-07T21:01:00+00:00	60.0	2.833333333333333
199b0cb9-6ed6-4410-9073-0fb2e7842b65/cpu/rate:mean	2022-11-07T21:01:00+00:00	60.0	3.083333333333335
61bfb555-9efb-46f1-8559-08dec90f94ed/cpu/rate:mean	2022-11-07T21:00:00+00:00	60.0	13.450000000000001
a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean	2022-11-07T21:00:00+00:00	60.0	2.45
199b0cb9-6ed6-4410-9073-0fb2e7842b65/cpu/rate:mean	2022-11-07T21:00:00+00:00	60.0	2.616666666666667
61bfb555-9efb-46f1-8559-08dec90f94ed/cpu/rate:mean	2022-11-07T20:59:00+00:00	60.0	60.583333333333336
a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean	2022-11-07T20:59:00+00:00	60.0	2.35
199b0cb9-6ed6-4410-9073-0fb2e7842b65/cpu/rate:mean	2022-11-07T20:59:00+00:00	60.0	2.525
61bfb555-9efb-46f1-8559-08dec90f94ed/cpu/rate:mean	2022-11-07T20:58:00+00:00	60.0	71.35833333333333
a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean	2022-11-07T20:58:00+00:00	60.0	3.025
199b0cb9-6ed6-4410-9073-0fb2e7842b65/cpu/rate:mean	2022-11-07T20:58:00+00:00	60.0	9.3
61bfb555-9efb-46f1-8559-08dec90f94ed/cpu/rate:mean	2022-11-07T20:57:00+00:00	60.0	66.19166666666668
a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean	2022-11-07T20:57:00+00:00	60.0	2.275
199b0cb9-6ed6-4410-9073-0fb2e7842b65/cpu/rate:mean	2022-11-07T20:57:00+00:00	60.0	56.31666666666667
61bfb555-9efb-46f1-8559-08dec90f94ed/cpu/rate:mean	2022-11-07T20:56:00+00:00	60.0	59.50833333333333
a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean	2022-11-07T20:56:00+00:00	60.0	2.375
199b0cb9-6ed6-4410-9073-0fb2e7842b65/cpu/rate:mean	2022-11-07T20:56:00+00:00	60.0	63.949999999999996
a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean	2022-11-07T20:55:00+00:00	60.0	15.558333333333335
199b0cb9-6ed6-4410-9073-0fb2e7842b65/cpu/rate:mean	2022-11-07T20:55:00+00:00	60.0	93.85

```
| a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean | 2022-11-07T20:54:00+00:00 |
60.0 | 59.54999999999999 |
| 199b0cb9-6ed6-4410-9073-0fb2e7842b65/cpu/rate:mean | 2022-11-07T20:54:00+00:00 |
60.0 | 61.23333333333333 |
| a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean | 2022-11-07T20:53:00+00:00 |
60.0 | 74.73333333333333 |
| a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean | 2022-11-07T20:52:00+00:00 |
60.0 | 57.86666666666667 |
| a95ab818-fbe8-4acd-9f7b-58e24ade6393/cpu/rate:mean | 2022-11-07T20:51:00+00:00 |
60.0 | 60.41666666666666 |
+-----+-----+-----+-----+
-----+
```

4.4.3. Retrieving available telemetry for an instance workload

Retrieve the available telemetry for an instance workload and express the vCPU utilization as a percentage.

Prerequisites

- Create a heat stack using the autoscaling group resource that results in an instance workload.

Procedure

1. Login to your OpenStack environment as the cloud administrator.
2. Retrieve the ID of the autoscaling group heat stack:

```
$ openstack stack show vnf -c id -c stack_status
+-----+-----+
| Field      | Value                                     |
+-----+-----+
| id         | e0a15cee-34d1-418a-ac79-74ad07585730 |
| stack_status | CREATE_COMPLETE                       |
+-----+-----+
```

3. Set the value of the stack ID to an environment variable:

```
$ export STACK_ID=$(openstack stack show vnf -c id -f value)
```

4. Retrieve the ID of the workload instance you want to return data for. We are using the server list long form and filtering for instances that are part of our autoscaling group:

```
$ openstack server list --long --fit-width | grep "metering.server_group='$STACK_ID'"
| bc1811de-48ed-44c1-ae22-c01f36d6cb02 | vn-xlfb4jb-yhbq6fkk2kec-qsu2lr47zigs-vnf-
y27wuo25ce4e | ACTIVE | None | Running | private=192.168.100.139, 192.168.25.179
| fedora36 | d21f1aaa-0077-4313-8a46-266c39b705c1 | m1.small | 692533fe-0912-417e-
b706-5d085449db53 | nova | standalone.localdomain |
metering.server_group='e0a15cee-34d1-418a-ac79-74ad07585730' |
```

5. Set the instance ID for one of the returned instance workload names:

```
$ INSTANCE_NAME='vn-xlfb4jb-yhbq6fkk2kec-qsu2lr47zigs-vnf-y27wuo25ce4e' ; export
INSTANCE_ID=$(openstack server list --name $INSTANCE_NAME -c ID -f value)
```

6. Verify metrics have been stored for the instance resource ID. If no metrics are available it's possible not enough time has elapsed since the instance was created. If enough time has elapsed, you can check the logs for the data collection service in `/var/log/containers/ceilometer/` and logs for the time-series database service gnocchi in `/var/log/containers/gnocchi/`:

```
$ openstack metric resource show --column metrics $INSTANCE_ID
+-----+-----+-----+
| Field | Value |
+-----+-----+-----+
| metrics | compute.instance.booting.time: 57ca241d-764b-4c58-aa32-35760d720b08 |
|      | cpu: d7767d7f-b10c-4124-8893-679b2e5d2ccd |
|      | disk.ephemeral.size: 038b11db-0598-4cfd-9f8d-4ba6b725375b |
|      | disk.root.size: 843f8998-e644-41f6-8635-e7c99e28859e |
|      | memory.usage: 1e554370-05ac-4107-98d8-9330265db750 |
|      | memory: fbd50c0e-90fa-4ad9-b0df-f7361ceb4e38 |
|      | vcpus: 0629743e-6baa-4e22-ae93-512dc16bac85 |
+-----+-----+-----+
```

7. Verify there are available measures for the resource metric and note the granularity value as we'll use it when running the **openstack metric aggregates** command:

```
$ openstack metric measures show --resource-id $INSTANCE_ID --aggregation rate:mean
cpu
+-----+-----+-----+
| timestamp | granularity | value |
+-----+-----+-----+
| 2022-11-08T14:12:00+00:00 | 60.0 | 71920000000.0 |
| 2022-11-08T14:13:00+00:00 | 60.0 | 88920000000.0 |
| 2022-11-08T14:14:00+00:00 | 60.0 | 76130000000.0 |
| 2022-11-08T14:15:00+00:00 | 60.0 | 17640000000.0 |
| 2022-11-08T14:16:00+00:00 | 60.0 | 33300000000.0 |
| 2022-11-08T14:17:00+00:00 | 60.0 | 24500000000.0 |
| ... | ... | ... |
```

8. Retrieve the number of vCPU cores applied to the workload instance by reviewing the configured flavor for the instance workload:

```
$ openstack server show $INSTANCE_ID -c flavor -f value
m1.small (692533fe-0912-417e-b706-5d085449db53)

$ openstack flavor show 692533fe-0912-417e-b706-5d085449db53 -c vcpus -f value
2
```

9. Return the metrics as an aggregate by resource type instance (server ID) with the value calculated as a percentage. The aggregate is returned as a value of nanoseconds of CPU time. We divide that number by 1000000000 to get the value in seconds. We then divide the value by our granularity, which in this example is 60 seconds (as previously retrieved with **openstack metric measures show** command). That value is then converted to a percentage by multiplying by 100. Finally, we divide the total value by the number of vCPU provided by the flavor assigned to the instance, in this example a value of 2 vCPU, providing us a value expressed as a percentage of CPU time:

```
$ openstack metric aggregates --resource-type instance --sort-column timestamp --sort-
```

```

descending '(/ (* (/ (/ (metric cpu rate:mean) 1000000000) 60) 100) 2)' id=$INSTANCE_ID
+-----+-----+-----+-----+
----+
| name                                | timestamp          | granularity |      value |
+-----+-----+-----+-----+
----+
| bc1811de-48ed-44c1-ae22-c01f36d6cb02/cpu/rate:mean | 2022-11-08T14:26:00+00:00 | 60.0 | 2.45 |
| bc1811de-48ed-44c1-ae22-c01f36d6cb02/cpu/rate:mean | 2022-11-08T14:25:00+00:00 | 60.0 | 11.075 |
| bc1811de-48ed-44c1-ae22-c01f36d6cb02/cpu/rate:mean | 2022-11-08T14:24:00+00:00 | 60.0 | 61.3 |
| bc1811de-48ed-44c1-ae22-c01f36d6cb02/cpu/rate:mean | 2022-11-08T14:23:00+00:00 | 60.0 | 74.78333333333332 |
| bc1811de-48ed-44c1-ae22-c01f36d6cb02/cpu/rate:mean | 2022-11-08T14:22:00+00:00 | 60.0 | 55.383333333333326 |
...

```