Red Hat OpenStack Platform 16.1

Auto Scaling for Instances

Configure Auto Scaling in Red Hat OpenStack Platform

Last Updated: 2021-03-04

Red Hat OpenStack Platform 16.1 Auto Scaling for Instances

Configure Auto Scaling in Red Hat OpenStack Platform

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Automatically scale out your Compute instances in response to system usage.

Table of Contents

CHAPTER 1. ABOUT THIS GUIDE

WARNING

Red Hat is currently reviewing the information and procedures provided in this guide for this release.

This document is based on the Red Hat OpenStack Platform 12 document, available at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/?version=12.

If you require assistance for the current Red Hat OpenStack Platform release, please contact Red Hat support.

CHAPTER 2. CONFIGURING AUTO SCALING FOR COMPUTE INSTANCES

This guide describes how to automatically scale out your Compute instances in response to heavy system usage. By using pre-defined rules that consider factors such as CPU or memory usage, you can configure Orchestration (heat) to add and remove additional instances automatically, when they are needed.

2.1. ARCHITECTURAL OVERVIEW OF AUTO SCALING

2.1.1. Orchestration

The core component providing automatic scaling is Orchestration (heat). You can use Orchestration to define rules using human-readable YAML templates. These rules are applied to evaluate system load based on Telemetry data to find out whether there is need to add more instances into the stack. When the load drops, Orchestration can automatically remove the unused instances again.

2.1.2. Telemetry

Telemetry does performance monitoring of your OpenStack environment, collecting data on CPU, storage, and memory utilization for instances and physical hosts. Orchestration templates examine Telemetry data to assess whether any pre-defined action should start.

2.1.3. Key terms

- **Stack** - A stack stands for all the resources necessary to operate an application. It can be as simple as a single instance and its resources, or as complex as multiple instances with all the resource dependencies that comprise a multi-tier application.
- **Templates** - YAML scripts that define a series of tasks for Heat to execute. For example, it is preferable to use separate templates for certain functions:
  - **Template File** - This is where you define thresholds that Telemetry should respond to, and define the auto scaling group.
  - **Environment File** - Defines the build information for your environment: which flavor and image to use, how the virtual network should be configured, and what software should be installed.

2.2. EXAMPLE: AUTO SCALING BASED ON CPU USE

In this example, Orchestration examines Telemetry data, and automatically increases the number of instances in response to high CPU use. Create a stack template and environment template to define the rules and subsequent configuration. This example uses existing resources, such as networks, and uses names that might be different to those in your own environment.

> NOTE
>
> The **cpu_util** metric was deprecated and removed from Red Hat OpenStack Platform.

Procedure

1. Create the environment template, describing the instance flavor, networking configuration, and image type. Save the template in the **/home/<user>/stacks/example1/cirros.yaml** file. Replace the **<user>** variable with a real user name.

```
heat_template_version: 2016-10-14
description: Template to spawn an cirros instance.

parameters:
  metadata:
    type: json
  image:
    type: string
    description: image used to create instance
    default: cirros
  flavor:
    type: string
    description: instance flavor to be used
    default: m1.tiny
  key_name:
    type: string
    description: keypair to be used
    default: mykeypair
  network:
    type: string
    description: project network to attach instance to
    default: internal1
  external_network:
    type: string
    description: network used for floating IPs
    default: external_network

resources:
  server:
    type: OS::Nova::Server
    properties:
      block_device_mapping:
        - device_name: vda
          delete_on_termination: true
          volume_id: { get_resource: volume }
      flavor: {get_param: flavor}
      key_name: {get_param: key_name}
      metadata: {get_param: metadata}
      networks:
        - port: { get_resource: port }

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

  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 }

    volume:
      type: OS::Cinder::Volume
      properties:
        image: {get_param: image}
        size: 1
```

2. Register the Orchestration resource in ~/**stacks/example1/environment.yaml**:

```
resource_registry:

    "OS::Nova::Server::Cirros": ~/stacks/example1/cirros.yaml
```

3. Create the stack template. Describe the CPU thresholds to watch for and how many instances to add. An instance group is also created that defines the minimum and maximum number of instances that can participate in this template.

> **NOTE**
>
> The **cpu_util** metric was deprecated and removed from Red Hat OpenStack Platform. To obtain the equivalent functionality, use the cumulative **cpu** metric and an archive policy that includes the **rate:mean** aggregation method. For example, **ceilometer-high-rate** and **ceilometer-low-rate**. You must convert the threshold value from % to ns to use the **cpu** metric for the CPU utilisation alarm. The formula is: time_ns = 1,000,000,000 x {granularity} x {percentage_in_decimal}. For example, for a threshold of 80% with a granularity of 1s, the threshold is 1,000,000,000 x 1 x 0.8 = 800,000,000.0

1. Save the following values in ~/**stacks/example1/template.yaml**:

```
heat_template_version: 2016-10-14
  description: Example auto scale group, policy and alarm
  resources:
    scaleup_group:
      type: OS::Heat::AutoScalingGroup
      properties:
        cooldown: 300
        desired_capacity: 1
        max_size: 3
        min_size: 1
        resource:
          type: OS::Nova::Server::Cirros
          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: 300
      scaling_adjustment: 1

  scaledown_policy:
    type: OS::Heat::ScalingPolicy
    properties:
      adjustment_type: change_in_capacity
      auto_scaling_group_id: { get_resource: scaleup_group }
      cooldown: 300
      scaling_adjustment: -1

  cpu_alarm_high:
    type: OS::Aodh::GnocchiAggregationByResourcesAlarm
    properties:
      description: Scale up if CPU > 80%
      metric: cpu
      aggregation_method: rate:mean
      granularity: 1
      evaluation_periods: 3
      threshold: 800000000.0
      resource_type: instance
      comparison_operator: gt
      alarm_actions:
        - str_replace:
            template: trust+url
            params:
              url: {get_attr: [scaleup_policy, signal_url]}
      query:
        str_replace:
          template: '{"=": {"server_group": "stack_id"}}'
          params:
            stack_id: {get_param: "OS::stack_id"}

  cpu_alarm_low:
    type: OS::Aodh::GnocchiAggregationByResourcesAlarm
    properties:
      metric: cpu
      aggregation_method: rate:mean
      granularity: 1
      evaluation_periods: 3
      threshold: 200000000.0
      resource_type: instance
      comparison_operator: lt
      alarm_actions:
        - str_replace:
            template: trust+url
            params:
              url: {get_attr: [scaledown_policy, signal_url]}
      query:
        str_replace:
          template: '{"=": {"server_group": "stack_id"}}'
          params:
            stack_id: {get_param: "OS::stack_id"}

outputs:
  scaleup_policy_signal_url:
```

```
    value: {get_attr: [scaleup_policy, signal_url]}

  scaledown_policy_signal_url:
    value: {get_attr: [scaledown_policy, signal_url]}
```

2. To build the environment and deploy the instance, enter the following command:

```
$ openstack stack create  -t template.yaml -e environment.yaml example
+---------------------+--------------------------------------------+
| Field               | Value                                      |
+---------------------+--------------------------------------------+
| id                  | 248a98bb-f56e-4934-a281-fffde62d78d8       |
| stack_name          | example                                    |
| description         | Example auto scale group, policy and alarm |
| creation_time       | 2017-03-06T15:00:29Z                       |
| updated_time        | None                                       |
| stack_status        | CREATE_IN_PROGRESS                         |
| stack_status_reason | Stack CREATE started                       |
+---------------------+--------------------------------------------+
```

3. Orchestration creates the stack and launches a defined minimum number of cirros instances, as defined in the **min_size** parameter of the **scaleup_group** definition. Verify that the instances were created successfully:

```
$ openstack server list
+--------------------------------------+-------------------------------------------------------+--------+------------+-------------+-------------------------------------+
| ID                                   | Name                                                  | Status | Task State | Power State | Networks                            |
+--------------------------------------+-------------------------------------------------------+--------+------------+-------------+-------------------------------------+
| e1524f65-5be6-49e4-8501-e5e5d812c612 | ex-3gax-5f3a4og5cwn2-png47w3u2vjd-server-vaajhuv4mj3j | ACTIVE | -          | Running     | internal1=10.10.10.9, 192.168.122.8 |
+--------------------------------------+-------------------------------------------------------+--------+------------+-------------+-------------------------------------+
----
+
. Orchestration also creates two cpu alarms which are used to trigger scale-up or scale-down events, as defined in `cpu_alarm_high` and `cpu_alarm_low`. Verify that the triggers exist:
+
[options="nowrap"]
----
$ 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 | insufficient data | low      | True    |
| 46ed2c50-e05a-44d8-b6f6-f1ebd83af913 | gnocchi_aggregation_by_resources_threshold |
```

```
example-cpu_alarm_low-m37jvnm56x2t  | insufficient data | low      | True    |
  +--------------------------------------+--------------------------------------------+--------------------------
------------+-------------------+----------+---------+
```

2.2.1. Test automatic scaling up instances

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

1. Log on to the instance and run several **dd** commands to generate the load:

```
$ ssh -i ~/mykey.pem cirros@192.168.122.8
$ sudo dd if=/dev/zero of=/dev/null &
$ sudo dd if=/dev/zero of=/dev/null &
$ sudo dd if=/dev/zero of=/dev/null &
```

2. After you run the **dd** commands, you can expect to have 100% CPU use in the cirros 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     |
+--------------------------------------+--------------------------------------------+--------------------------
-----------+-------+----------+---------+
```

3. After some time (approximately 60 seconds), Orchestration starts another instance and adds it into the group. To verify this, enter the following command:

```
$ openstack server list
+--------------------------------------+-------------------------------------------------------+--------+------
------+-------------+--------------------------------------+
| ID                                   | Name                                                  | Status | Task State | Power
State | Networks                             |
+--------------------------------------+-------------------------------------------------------+--------+------
------+-------------+--------------------------------------+
| 477ee1af-096c-477c-9a3f-b95b0e2d4ab5 | ex-3gax-4urpikl5koff-yrxk3zxzfmpf-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   |
+--------------------------------------+-------------------------------------------------------+--------+------
------+-------------+--------------------------------------+
```

4. After another short period, observe that Orchestration has auto scaled again to three instances. The configuration is set to three instances maximally, so it will not scale any higher (the **scaleup_group** definition: **max_size**). Verify that with the following command:

```
$ openstack server list
+--------------------------------------+-------------------------------------------------------+--------+------------+-------------+---------------------------------------+
| ID                                   | Name                                                  | Status | Task State | Power State | Networks                              |
+--------------------------------------+-------------------------------------------------------+--------+------------+-------------+---------------------------------------+
| 477ee1af-096c-477c-9a3f-b95b0e2d4ab5 | ex-3gax-4urpikl5koff-yrxk3zxzfmpf-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   |
+--------------------------------------+-------------------------------------------------------+--------+------------+-------------+---------------------------------------+
```

2.2.2. Automatically scaling down instances

Orchestration can also 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%.

1. Terminate the running **dd** processes and observe Orchestration begin to scale the instances back down.

```
$ killall dd
```

2. Stopping the **dd** processes causes the **cpu_alarm_low event** to trigger. As a result, Orchestration begins to automatically scale down and remove the instances. Verify that the corresponding 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 | 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**.

2.2.3. Troubleshooting the setup

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

1. 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)
```

2. To read the alarm history log, enter the following command:

```
$ 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": "{\"=\":   |                                  |
|                         |                 | {\"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"}                                  |                                  |
+---------------------------+------------------+--------------------------------------------------------------
-------------------------------------+-------------------------------------
```


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

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

4. To see the **aodh** related information, examine the **evaluator.log**:

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