



Red Hat OpenStack Platform 10

Compute のオートスケール

Red Hat OpenStack Platform におけるオートスケールの設定

Red Hat OpenStack Platform 10 Compute のオートスケール

Red Hat OpenStack Platform におけるオートスケールの設定

OpenStack Team
rhos-docs@redhat.com

法律上の通知

Copyright © 2017 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, 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 Software Collections 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.

概要

システムの使用状況に応じて、Compute インスタンスを自動的にスケールアウトします。

目次

第1章 COMPUTE のオートスケールの設定	3
1.1. アーキテクチャーの概要	3
1.1.1. Orchestration	3
1.1.2. Telemetry	3
1.1.3. 主要な用語	3
1.2. 例: CPU 使用率に基づいたオートスケール	3
1.2.1. インスタンスの自動スケーリングアップのテスト	8
1.2.2. インスタンスの自動スケールダウン	9
1.2.3. セットアップのトラブルシューティング	10
1.3. 例: アプリケーションのオートスケール	12
1.3.1. アプリケーションのオートスケールのテスト	18
1.3.2. アプリケーションの自動スケールダウン	20

第1章 COMPUTE のオートスケールの設定

本ガイドは、システム使用率が過度に高くなった場合に Compute インスタンスを自動的にスケールアウトする方法について説明します。CPU やメモリーの使用率を考慮した事前定義済みのルールを使用することにより、Orchestration (heat) が必要に応じて自動的にインスタンスの追加/削除をするように設定することが可能です。

1.1. アーキテクチャーの概要

1.1.1. Orchestration

オートスケール機能を提供するコアコンポーネントは Orchestration (heat) です。Orchestration では、人間が判読できる YAML テンプレートを使用してルールを定義することができます。これらのルールは、Telemetry のデータに基づいてシステムの負荷を評価し、スタックにインスタンスを追加する必要があるかどうかを確認するのに適用されます。負荷が低減した後は、Orchestration は使用されていないインスタンスを自動的に削除することができます。

1.1.2. Telemetry

Telemetry は、インスタンスおよび物理ホストの CPU、ストレージ、メモリーの使用率に関するデータを収集して、OpenStack 環境のパフォーマンスを監視します。Orchestration テンプレートは、Telemetry データを検証して、事前定義されたアクションを開始するかどうかを評価します。

1.1.3. 主要な用語

- **スタック**: スタックとは、1つのアプリケーションを稼働させるのに必要な全リソースを意味します。1つのインスタンスとそのリソースから成る単純なスタックもあれば、複数階層のアプリケーションを構成するリソースの依存関係を伴う複数のインスタンスから成る複雑なスタックもあります。
- **テンプレート**: Heat が実行する一式のタスクを定義する YAML スクリプト。たとえば、機能別に異なるテンプレートを使用するのが望ましいです。
 - **テンプレートファイル**: このファイルには、Telemetry が対応する必要のある閾値やオートスケールグループを定義します。
 - **環境ファイル**: 使用するフレーバーやイメージ、仮想ネットワークの設定方法、インストールするソフトウェアなどの環境のビルド情報を定義します。

1.2. 例: CPU 使用率に基づいたオートスケール

以下の例では、Orchestration が Telemetry データを検証し、CPU 使用率の増加に対応して、インスタンスの数を自動的に増やします。必要なルールとその後の設定を定義するためにスタックテンプレートと環境テンプレートが作成されます。この例では、既存のリソース (ネットワークなど) を使用しており、実際にお使いの環境のリソース名とは異なる可能性があります。

1. インスタンスのフレーバー、ネットワーク設定、イメージ種別を記述した環境テンプレートを作成して、`/home/<user>/stacks/example1/cirros.yaml` のテンプレートファイルに保存します。<user> の変数は実際のユーザー名に置き換えてください。

```
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. `~/stacks/example1/environment.yaml` に Orchestration のリソースを登録します。

```

resource_registry:

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

```

3. 監視する CPU の閾値と追加すべきインスタンス数を記述するスタックテンプレートを作成します。また、インスタンスグループも作成して、このテンプレートに参加することが可能なインスタンス数の最小値および最大値を定義します。



注記

granularity パラメーターは、**gnocchi cpu_util** メトリックの粒度に応じて設定する必要があります。詳しくは、[ソリューションの記事](#) を参照してください。

`~/stacks/example1/template.yaml` に以下の値を保存します。

```

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_util
    aggregation_method: mean
    granularity: 300
    evaluation_periods: 1
    threshold: 80
    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_util
    aggregation_method: mean
    granularity: 300
    evaluation_periods: 1
    threshold: 5
    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]}

```

4. 以下の OpenStack コマンドを実行して環境を構築し、インスタンスをデプロイします。

```

$ 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
|
+-----+-----+
--+

```

5. Orchestration は、スタックを作成して、**scaleup_group** 定義の **min_size** パラメーターで指定されている最小数の cirros インスタンスを起動します。インスタンスが正常に作成されたことを確認します。

```

$ 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 |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+

```

6. Orchestration は、**cpu_alarm_high** および **cpu_alarm_low** の定義に従って、スケールアップまたはスケールダウンのイベントをトリガーするのに使用する 2 つの CPU アラームも作成します。トリガーが存在することを確認します。

```

$ 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 | insufficient data | low      | True      |
| 46ed2c50-e05a-44d8-b6f6-f1ebd83af913 |
gnocchi_aggregation_by_resources_threshold | example-cpu_alarm_low-
m37jvnm56x2t | insufficient data | low      | True      |
+-----+-----+-----+
-----+-----+-----+
-----+-----+-----+

```

1.2.1. インスタンスの自動スケーリングアップのテスト

Orchestration は、**cpu_alarm_high** 閾値の定義に基づいて、インスタンスを自動的にスケーリングすることができます。CPU の使用率が **threshold** パラメーターで定義されている値に達すると、負荷のバランスを取るために別のインスタンスが起動します。上記の **template.yaml** ファイルでは、**threshold** 値は 80% に設定されています。

1. インスタンスにログインして **dd** コマンドを数回実行し、負荷を生成します。

```

$ 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. **dd** コマンドを実行すると、cirros インスタンスの CPU 使用率が 100% となることが予想できます。アラームがトリガーされていることを確認します。

```

$ 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 | alarm | low      | True      |
| 46ed2c50-e05a-44d8-b6f6-f1ebd83af913 |
gnocchi_aggregation_by_resources_threshold | example-cpu_alarm_low-
m37jvnm56x2t | ok    | low      | True      |
+-----+-----+-----+
-----+-----+-----+
+-----+-----+

```

3. しばらく経つと (約 60 秒)、Orchestration は別のインスタンスを起動して、グループに追加します。これは、**nova list** コマンドで確認することができます。

```

$ 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. さらに短時間が経過すると、Orchestration がインスタンスを再度オートスケールして 3 つになったことを確認することができます。設定では最大 3 つに指定されているので、その値を上回る数にはスケーリングされません (**scaleup_group** 定義の **max_size** パラメーター)。この場合も、上記のコマンドで確認することができます。

```

$ 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   |
+-----+-----+-----+-----+
-+-----+

```

1.2.2. インスタンスの自動スケールダウン

Orchestration は、**cpu_alarm_low** の閾値に基づいて、インスタンスを自動的にスケールダウンすることも可能です。以下の例では、CPU の使用率が 5% を下回ると、インスタンスがスケールダウンされます。

1. 実行中の **dd** プロセスを終了すると、Orchestration がインスタンスのスケールダウンを開始するのを確認することができます。

```
$ killall dd
```

2. **dd** プロセスを停止すると、**cpu_alarm_low event** がトリガーされます。その結果、Orchestration が自動的にスケールダウンを開始して、インスタンスを削除します。対応するアラームがトリガーされていることを確認します。

```
$ 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    |
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+-----+

```

数分後には、Orchestration はインスタンス数が **scaleup_group** 定義の **min_size** パラメーターで指定されている最小値になるまでスケールダウンを続けます。**min_size** パラメーターは **1** に設定されています。

1.2.3. セットアップのトラブルシューティング

環境が適切に機能していない場合には、ログファイルと履歴の記録でエラーを確認することができます。

1. 状態の遷移を確認するには、スタックのイベント記録を一覧表示することができます。

```
$ 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. アラームの履歴ログを確認するには以下のコマンドを実行します。

```

$ 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", "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", "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", "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-"], "description": "CPU usage alarm", "name": "example.cpu_alarm_high", "notification_url": "http://192.168.122.126:8004/v1/fd", "repeat_interval": 300, "severity": "critical", "type": "cpu_alarm_high"} |

```


のような場合には、**neutron** で **Load Balancing-as-a-Service** を提供するように設定して、インスタンス間でトラフィックが均等に分散されるようにすることができます。

以下の例では、Orchestration が再び Telemetry データを検証して、高い CPU 使用率が検出されるとインスタンス数を増やし、指定した値よりも低い値の CPU 使用率が返されるとインスタンス数を減らします。

1. **load-balancer** 環境のプロパティを記述したテンプレートを作成します。~/stacks/example2/lb-env.yaml に以下の値を入力します。

```
heat_template_version: 2014-10-16
description: A load-balancer server
parameters:
  image:
    type: string
    description: Image used for servers
  key_name:
    type: string
    description: SSH key to connect to the servers
  flavor:
    type: string
    description: flavor used by the servers
  pool_id:
    type: string
    description: Pool to contact
  user_data:
    type: string
    description: Server user_data
  metadata:
    type: json
  network:
    type: string
    description: Network used by the server

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

  member:
    type: OS::Neutron::PoolMember
    properties:
      pool_id: {get_param: pool_id}
      address: {get_attr: [server, first_address]}
      protocol_port: 80

  port:
    type: OS::Neutron::Port
    properties:
```

```

        network: {get_param: network}
        security_groups:
            - base

outputs:
    server_ip:
        description: IP Address of the load-balanced server.
        value: { get_attr: [server, first_address] }
    lb_member:
        description: LB member details.
        value: { get_attr: [member, show] }

```

2. Web アプリケーションを実行するインスタンス用に別のテンプレートを作成します。以下のテンプレートは、ロードバランサーを作成して、既存のネットワークを使用します。パラメーターは環境に応じて必ず変更し、~/stacks/example2/lb-webserver-rhel7.yaml のようなファイルにテンプレートを保存してください。

```

heat_template_version: 2014-10-16
description: AutoScaling RHEL 7 Web Application
parameters:
    image:
        type: string
        description: Image used for servers
        default: RHEL 7
    key_name:
        type: string
        description: SSH key to connect to the servers
        default: admin
    flavor:
        type: string
        description: flavor used by the web servers
        default: m2.tiny
    network:
        type: string
        description: Network used by the server
        default: private
    subnet_id:
        type: string
        description: subnet on which the load balancer will be located
        default: 9daa6b7d-e647-482a-b387-dd5f855b88ef
    external_network_id:
        type: string
        description: UUID of a Neutron external network
        default: db17c885-77fa-45e8-8647-dbb132517960

resources:
    webserver:
        type: OS::Heat::AutoScalingGroup
        properties:
            min_size: 1
            max_size: 3
            cooldown: 60
            desired_capacity: 1
            resource:
                type: file:///etc/heat/templates/lb-env.yaml
            properties:

```

```

        flavor: {get_param: flavor}
        image: {get_param: image}
        key_name: {get_param: key_name}
        network: {get_param: network}
        pool_id: {get_resource: pool}
        metadata: {"metering.stack": {get_param: "OS::stack_id"}}
        user_data:
            str_replace:
                template: |
                    #!/bin/bash -v

                    yum -y install httpd php
                    systemctl enable httpd
                    systemctl start httpd
                    cat <<EOF > /var/www/html/hostname.php
                    <?php echo "Hello, My name is " . php_uname('n'); ?

>
            EOF
        params:
            hostip: 192.168.122.70
            fqdn: sat6.example.com
            shortname: sat6

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

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

monitor:
    type: OS::Neutron::HealthMonitor
    properties:
        type: TCP
        delay: 5
        max_retries: 5
        timeout: 5

pool:
    type: OS::Neutron::Pool
    properties:
        protocol: HTTP
        monitors: [{get_resource: monitor}]
        subnet_id: {get_param: subnet_id}
        lb_method: ROUND_ROBIN
        vip:
            protocol_port: 80

```

```

lb:
  type: OS::Neutron::LoadBalancer
  properties:
    protocol_port: 80
    pool_id: {get_resource: pool}

lb_floating:
  type: OS::Neutron::FloatingIP
  properties:
    floating_network_id: {get_param: external_network_id}
    port_id: {get_attr: [pool, vip, port_id]}

outputs:
  scale_up_url:
    description: >
      This URL is the webhook to scale up the autoscaling group.
  You
    can invoke the scale-up operation by doing an HTTP POST to
  this
    URL; no body nor extra headers are needed.
    value: {get_attr: [web_server_scaleup_policy, alarm_url]}
  scale_dn_url:
    description: >
      This URL is the webhook to scale down the autoscaling group.
      You can invoke the scale-down operation by doing an HTTP POST
  to
    this URL; no body nor extra headers are needed.
    value: {get_attr: [web_server_scaledown_policy, alarm_url]}
  pool_ip_address:
    value: {get_attr: [pool, vip, address]}
    description: The IP address of the load balancing pool
  website_url:
    value:
      str_replace:
        template: http://serviceip/hostname.php
        params:
          serviceip: { get_attr: [lb_floating, floating_ip_address]
}
    description: >
      This URL is the "external" URL that can be used to access the
      website.
  gnocchi_query:
    value:
      str_replace:
        template: >
          gnocchi measures aggregation --query
'server_group="stackval"'
          --granularity 300 --aggregation mean cpu_util
        params:
          stackval: { get_param: "OS::stack_id" }
    description: >
      This is a gnocchi query for statistics on the cpu_util meter
      Samples about OS::Nova::Server instances in this stack.
      The --query parameter selects Samples according to the
      subject's metadata.

```

When a VM's metadata includes an item of the form `metering.X=Y`, the corresponding gnocchi resource has a metadata item of the form `user_metadata.X=Y` and samples about resources so tagged can be queried with a gnocchi query term of the form `metadata.user_metadata.X=Y`. In this case the nested stacks give their VMs metadata that is passed as a nested stack parameter, and this stack passes a metadata of the form `metering.stack=Y`, where Y is this stack's ID.

3. Telemetry のデータ収集の間隔を更新します。デフォルトでは、Telemetry はインスタンスを 10 分ごとにポーリングして、CPU のデータを取得します。以下の例では、`/etc/ceilometer/pipeline.yaml` で、この間隔を 60 秒に変更します。

```
- name: cpu_source
interval: 60
meters:
- "cpu"
sinks:
- cpu_sink
```



注記

ポーリングの頻度を高くすると、コントロールプレーンに対する負荷が高くなるため、実稼働環境では、60 秒に設定することは、お勧めできません。

4. OpenStack ceilometer の全サービスを再起動して、更新された Telemetry の設定を適用します。

```
# systemctl restart openstack-ceilometer*
```

5. Orchestration のスクリプトを実行します。このスクリプトにより、環境が構築され、テンプレートを使用してインスタンスがデプロイされます。

```
# heat stack-create webfarm -f /root/lb-webserver-rhel7.yaml
```

`/root/lb-webserver-rhel7.yaml` の箇所は、実際のパスとファイル名に変更してください。

Dashboard の **オーケストレーション → スタック → Webfarm** で、スタックの作成をモニタリングすることができます。スタックの作成が完了すると、複数の役立つ情報が表示されます。特に以下の情報を確認してください。

- 手動のスケールアップまたはスケールダウンイベントをトリガーするのに使用することができる URL
- Floating IP アドレス。Web サイトの IP アドレスです。
- スタック全体の CPU 負荷を表示するための Telemetry コマンド。スケーリングが想定通りに機能しているかどうかを確認するのに使用することができます。

Dashboard のページは以下のように表示されます。

ネットワーク → ロードバランサー を開いて、ロードバランサーを確認します。

メンバー をクリックします。このページには、ロードバランシングプールのメンバーが表示されます。これらのインスタンスに対して、Web サイトのトラフィックを分散することができます。対応するインスタンスが作成されて、Apache がインストール/設定されるまで、メンバーのステータスは **Active** にはならない点に注意してください。

[illegible]

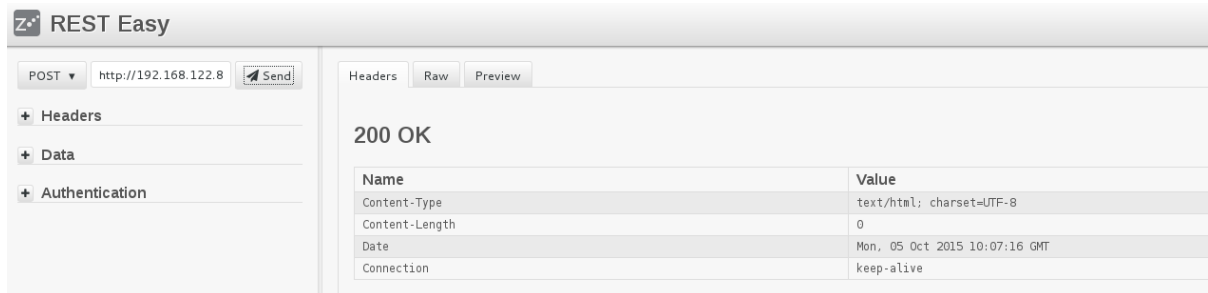
Hello, My name is we-zrwm-t4ezkpx34gxu-qbg5d7dqbc4j-server-mzdvigk2jugl

```
# ceilometer statistics -m cpu_util -q
metadata.user_metadata.stack=8f86c3d5-15cf-4a64-b9e8-70215498c046 -p 60 -a
avg
```

1.3.1. アプリケーションのオートスケールのテスト

アプリケーションのスケールリングを手動でトリガーするには、Dashboard のスタックの概要の画面に表示される REST **scale-up URL** を使用するか、始めにデプロイしたインスタンス上で resource-intensive コマンドを実行して負荷を生成します。

- REST API を使用するには、**REST Easy** という [Firefox アドオン](#) や **curl** など、HTTP POST 要求を実行できるツールが必要です。**REST Easy** を使用する場合には、**scale-up URL** をコピーしてフォームにペーストします。

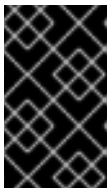


curl を使用する場合には、コマンドラインで URL をパラメーターとして渡します。

```
$ curl -X POST "scale-up URL"
```

- 負荷を人為的に生成するには、Floating IP をインスタンスに割り当てて、SSH でログインし、CPU の使用率を高くするコマンドを実行して、CPU をビジー状態にします。以下に例を示します。

```
$ dd if=/dev/zero of=/dev/null &
```



重要

top コマンドなどを使用して、CPU 使用率が 95% 以上かどうかを確認します。CPU 使用率が十分に高くない場合には、**dd** コマンドを複数回並行して実行するか、他の方法を使用して CPU をビジー状態にしてください。

次回に Telemetry が CPU データをスタックから収集すると、スケールアップイベントがトリガーされ、**オーケストレーション → スタック → Webfarm → イベント** に表示されます。新規 Web サーバーインスタンスが作成され、ロードバランサーに追加されます。この操作が終了すると、インスタンスがアクティブになり、Web サイトの URL はロードバランサーを介して、スタック内の両方のインスタンスにルーティングされます。



注記

インスタンスを初期化し、Apache をインストール/設定してからアプリケーションをデプロイする必要があるため、作成には数分かかる場合があります。この操作は、HAProxy によってモニタリングされ、ステータスが Active に切り替わる前に、Web サイトがインスタンス上で使用可能であることが確認されます。

新規インスタンスの作成中に、Dashboard では、ロードバランシングプールのメンバーが以下のように表示されます。

RED HAT® ENTERPRISE LINUX OPENSTACK PLATFORM						
Project Admin Identity						
Compute	Network	Object Store	Orchestration	Database	Data Processing	
Network Topology Networks Routers Load Balancers						
Pools Members Monitors						
<div> <div>+</div> Add Member <div>×</div> Delete Members </div>						
<input type="checkbox"/>	IP Address	Protocol Port	Weight	Pool	Status	Actions
<input type="checkbox"/>	10.10.1.176	80	1	webfarm-pool-hsmpjmgas2d7	Active	<div>Edit Member</div>
<input type="checkbox"/>	10.10.1.177	80	1	webfarm-pool-hsmpjmgas2d7	Inactive	<div>Edit Member</div>
Displaying 2 items						



重要

追加のインスタンスを作成するかどうかを決定する際には、**heat** スタック内のインスタンスの CPU 使用率の平均が考慮されます。2 番目のインスタンスは通常の CPU 使用率である可能性が最も高いので、1 番目のインスタンスの負荷が相殺されますが、2 番目のインスタンスもビジー状態となり、1 番目と 2 番目のインスタンスの CPU 使用率が 95% を超えると、もう 1 つ (3 番目) のインスタンスが作成されます。

1.3.2. アプリケーションの自動スケールダウン

この手順は、「[インスタンスの自動スケールダウン](#)」と同様に、スタック CPU 使用率の平均が、事前定義された値（「[アプリケーションのオートスケールのテスト](#)」に記載の例では、15%）を下回ると、スケールダウンポリシーがトリガーされます。また、インスタンスがこの方法でスタックから削除された場合には、ロードバランサーからも自動的に削除され、Web サイトのトラフィックは、残りのインスタンス間で分散されます。