Red Hat OpenStack Platform 13

インスタンスの自動スケーリング

Red Hat OpenStack Platform での自動スケーリングの設定

Last Updated: 2023-04-10

Red Hat OpenStack Platform 13 インスタンスの自動スケーリング

Red Hat OpenStack Platform での自動スケーリングの設定

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

システムの使用状況に応じて、コンピュート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 コンピュートインスタンスの自動スケーリングの設定

高いシステムの使用状況に応じてコンピュートインスタンスを自動的にスケールアウトできます。CPU やメモリー使用などの要素を考慮した事前定義のルールを使用することで、必要に応じて追加のインスタンスを自動的に追加および削除するようにオーケストレーション (Heat) を設定できます。

1.1. 自動スケーリングアーキテクチャーの概要

1.1.1. オーケストレーション

自動スケーリングを提供するコアコンポーネントは Orchestration (heat) です。Orchestration を使用すると、人間が判読可能な 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 データに基づいてシステムの負荷を評価するのに適用され、スタックにインスタンスをさらに追加する必要があるかどうかを判断します。負荷が減ると、オーケストレーションは未使用のインスタンスを再び自動的に削除できます。

1.1.2. Telemetry

Telemetry を使用して Red Hat OpenStack Platform 環境のパフォーマンスを監視し、インスタンスと物理ホストの CPU、ストレージ、およびメモリー使用に関するデータを収集できます。オーケストレーションテンプレートは Telemetry データを検査し、事前定義の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** - スタックは、アプリケーションを操作するために必要なすべてのリソースを表します。1 つのインスタンスとそのリソースではシンプルで、複数階層アプリケーションを設定する複数のインスタンスとすべてのリソースの依存関係の場合は複雑になります。
- **テンプレート** - Heat が実行する一連のタスクを定義する YAML スクリプト。たとえば、特定の機能に個別のテンプレートを使用することが推奨されます。
  - **テンプレートファイル** - ここで、Telemetry が応答する必要があるしきい値を定義し、自動スケーリンググループを定義します。
  - **環境ファイル** - 環境のビルド情報 (使用するフレーバーとイメージ、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) を定義します。

1.2. 例: CPU の使用状況に基づく自動スケーリング

この例では、オーケストレーションは Telemetry データを検査し、CPU の高い使用率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、ネットワークなどの既存のリソースを利用し、独自の環境では異なる可能性が高い名前を使用しています。

1. インスタンスのフレーバー、ネットワーク設定、イメージタイプを記述した環境テンプレートを作成し、テンプレートの **/home/<user>/stacks/example1/cirros.yaml** ファイルに保存します。**<user>** 変数を実際のユーザー名に置き換えます。

```
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. Orchestration リソースを ~/**stacks/example1/environment.yaml** に登録します。

```
resource_registry:

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

3. スタックテンプレートを作成し、監視する CPU しきい値と追加するインスタンス数を記述します。インスタンスグループも作成し、このテンプレートに参加できるインスタンスの最小数および最大数を定義します。

注記

granularity パラメーターは、**gnocchi cpu_util** メトリックの粒度に従って設定する必要があります。詳細については、この ソリューション記事 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. 次の 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、**scaleup_group** 定義の **min_size** パラメーターで定義されているように、定義された最小数の 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. オーケストレーションは、**cpu_alarm_high** および **cpu_alarm_low** で定義されているように、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ために使用される 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-odj77qpbld7j | 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ルアップのテスト

Orchestration は、**cpu_alarm_high** しきい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できます。CPU 使用率が **threshold** パラメーターで定義された値に達すると、負荷を分散するために別のインスタンスが開始されます。上記の **template.yaml** ファイルの **threshold** は 80% に設定されています。

1. インスタンスにログオンし、複数の **dd** コマンドを実行してロードを生成します。

```
$ 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 インスタンスで 100% の CPU 使用率が期待できます。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-odj77qpbld7j | alarm | low      | True    |
| 46ed2c50-e05a-44d8-b6f6-f1ebd83af913 | gnocchi_aggregation_by_resources_threshold |
example-cpu_alarm_low-m37jvnm56x2t  | ok    | low      | True    |
+--------------------------------------+----------------------------------------+-------------------------
----------+-------+----------+---------+
```

3. しばらくすると (約 60 秒)、Orchestration は別のインスタンスを開始し、それをグループに追加します。これは、**nova list** コマンドで確認できます。

```
$ 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. さらに少し経過したら、Orchestration が 3 つのインスタンスに再度自動的にスケーリングしていることを確認します。設定は最大で 3 つのインスタンスに設定されているため、これ以上スケールアップすることはありません (**scaleup_group** 定義: **max_size**)。繰り返しますが、上記のコマンドで確認できます。

```
$ 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

オーケストレーションは、**cpu_alarm_low** しきい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できます。この例では、CPU 使用率が 5% を下回るとインスタンスが縮小されます。

1. 実行中の **dd** プロセスを終了して、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します。

```
$ 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。これにより、Orchestration は自動的にスケールダウンを開始し、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たことを確認します。

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

5 分後、Orchestration はインスタンスの数を **scaleup_group** 定義の **min_size** パラメーターで定義された最小値まで継続的に減らします。このシナリオでは、**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は、ログファイルと履歴レコードでエラーを確認できます。

1. 状態遷移に関する情報を取得するには、スタックイベントレコードを一覧表示します。

```
$ 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. アラーム履歴ログを読むには:

```
$ 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. Heat が既存のスタックに対して収集するスケールアウトまたはスケールダウン操作の記録を確認するには、**awk** を使用して **heat-engine.log** を解析します。

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

4. **aodh** 関連の情報を確認するには、**evaluator.log** を調べます。

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