



Red Hat OpenStack Platform 16.2

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

Red Hat OpenStack Platform 16.2 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

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Auto_Scaling_for_Instances.ent file |.

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.

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

前書き	3
多様性を受け入れるオープンソースの強化	4
RED HAT ドキュメントへのフィードバック (英語のみ)	5
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	6
1.1. 自動スケーリングに関する主要な用語	6
1.2. 例：CPU の使用状況に基づく自動スケーリング	6
1.3. インスタンスの自動スケールアップのテスト	10
1.4. インスタンスを自動スケールダウン	12
1.5. 自動スケーリングのトラブルシューティング	12

前書き

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、[弊社](#) の CTO、Chris Wright の [メッセージ](#) を参照してください。

RED HAT ドキュメントへのフィードバック (英語のみ)

弊社ドキュメントに対するご意見をお聞かせください。ドキュメントの改善点があればお知らせください。

ドキュメントへのダイレクトフィードバック (DDF) 機能の使用 (英語版のみ)

特定の文章、段落、またはコードブロックに対して直接コメントを送付するには、DDF の **Add Feedback** 機能を使用してください。なお、この機能は英語版のドキュメントでのみご利用いただけます。

1. **Multi-page HTML** 形式でドキュメントを表示します。
2. ドキュメントの右上隅に **Feedback** ボタンが表示されていることを確認してください。
3. コメントするテキスト部分をハイライト表示します。
4. **Add Feedback** をクリックします。
5. **Add Feedback** フィールドにコメントを入力します。
6. (オプション) ドキュメントチームが連絡を取り問題についてお伺いできるように、ご自分のメールアドレスを追加します。
7. **送信** をクリックします。

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するには、CPU やメモリー使用量などの要素を考慮する事前定義済みのルールを使用することができます。お使いの環境のニーズに応じて追加のインスタンスを自動的に追加/削除するように、Orchestration (heat)を設定することもできます。

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。

Telemetry を使用して Red Hat OpenStack Platform (RHOSP)環境のパフォーマンスを監視し、インスタンスおよび物理ホストの CPU、ストレージ、およびメモリー使用率に関するデータを収集します。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. 自動スケーリングに関する主要な用語

- **スタック**: スタックは、アプリケーションの操作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構成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**: 実行する Orchestration(heat)の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**: Telemetry が応答す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**: 使用するフレーバーおよびイメージ、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、環境のビルド情報を定義します。

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、Orchestration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。スタックテンプレートおよび環境テンプレートを作成して、ルールとサブ設定を定義します。この例では、ネットワークなどの既存のリソースを使用し、実際の環境とは異なる可能性のある名前を使用します。



注記

cpu_util メトリックは非推奨となり、Red Hat OpenStack Platform から削除されました。

手順

1. インスタンスのフレーバー、ネットワーク設定、およびイメージ種別を記述する環境テンプレートを作成します。テンプレートを `/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

```

- 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

```

- スタックテンプレートを作成します。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

cpu_util メトリックは非推奨となり、Red Hat OpenStack Platform から削除されました。同等の機能を取得するには、累積の **cpu** メトリックおよび **rate:mean** 集約メソッドが含まれるアーカイブポリシーを使用します(例: **ceilometer-high-rate** および **ceilometer-low-rate**)。CPU 使用率のアラームに **cpu** メトリクスを使用するには、しきい値を%から ns に変換する必要があります。式は $ns = 1,000,000,000 \times \{granularity\} \times \{percentage_in_decimal\}$ です。たとえば、しきい値が 80% で粒度が1秒の場合、しきい値は $1,000,000,000 \times 1 \times 0.8 = 800,000,000.0$ です。

- `~/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
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]}
```

5. 環境をビルドし、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                    |
+-----+-----+
```

6. 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 |
+-----+-----+-----+-----+-----+-----+
```

7. また、Orchestration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 |
+-----+-----+-----+-----+-----+-----+
```

1.3. 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ルリングできます。CPU使用率が **threshold** パラメーターで定義された値に達すると、別のインスタンスが起動して負荷のバランスを取ります。**template.yaml** ファイルの **threshold** 値は、80% に設定されます。

手順

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 インスタンスでは CPU が 100% 使用されると予想されます。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. 約 60 秒後に、Orchestration は別のインスタンスを開始し、グループに追加します。これを確認するには、以下のコマンドを入力します。

```
$ 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 3 つのインスタンスに設定されているので、さらにスケーリングすることはできません (**scaleup_group** の定義: **max_size**)。インスタンスが 3 つあることを確認するには、以下のコマンドを入力します。

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

1.4. インスタンスを自動スケールダウン

Orchestration を使用して、**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できます。この例では、CPU の使用率が 5% 未満の場合に、インスタンスはスケールダウンされます。

手順

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** アラーム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 | 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、Orchestration は、**scaleup_group** 定義の **min_size** パラメーターで定義される最小値までインスタンスの数を継続的に削減します。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。

1.5. 自動スケーリング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。

手順

1. 状態遷移についての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確認するには、以下のコマンドを入力します。

```

$ 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 |
|                          | 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-odj77qpbl7j", "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. 既存のスタック用に heat が収集するスケールアウトまたはスケールダウン操作のレコードを表示するには、**awk** コマンドを使用して **heat-engine.log** を解析することができます。

```

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

```

4. **aodh** 関連の情報を表示するには、**evaluator.log** を調べます。

```

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

```