Auto-Scaling in OpenStack

Saurav Tewari

Student volunteer (May 2015- Aug 2015)

Index

Introduction

Heat is the component responsible for orchestration in OpenStack. It manages the lifecycle of openstack application using templates and defines the relationship among resources. It defines the lifecycle of applications. Heat allow advanced functionality such as nested stacks and autoscaling. AutoScaling is a feature of OpenStack Heat that allows resources of an application to autoscale when required. Autoscaling is possible for any number of resources but in this application we have considered only autoscaling of virtual machines.

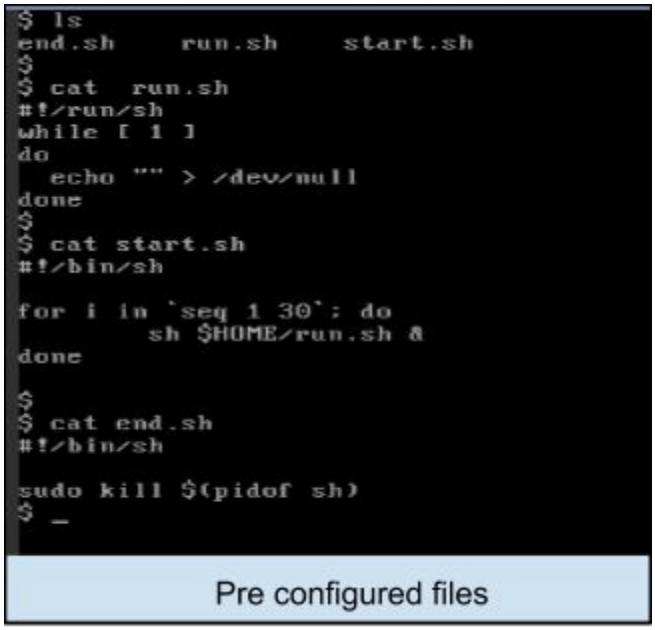
In auto-scaling Heat and Ceilometer plays vital role. Heat provides resources to be scaled and policy for scaling. Ceilometer provides alarm which notifies when certain threshold (of some meter) have been met.

Pre-Configuration

Creating image with following three files.

1. run.sh: this file contains code for running infinite echo and dumping to \dev\null file.
2. start.sh: it contains for loop that executes run.sh file in each loop.
3. end.sh: it contains command to kill all the running shell processes.

After all files have been created take a snapshot of this image. This screenshot will be used for booting all the virtual machine instances for auto-scaling.



Pre configured files

Heat Orchestration Template (HOT)

It it the default format which is used to create stack: a collection of resource. HOT is written in YAML (YAML Ain't Markup Language) format. It integrates well with software configuration management tools and other OpenStack components. There are three versions of HOT that are

available today. The version of templates matter because each version contains specific features and supports specific functions:

1. Icehouse (oldest) 2013-05-23
2. Juno 214-10-16
3. Kilo (latest) 2015-04-30

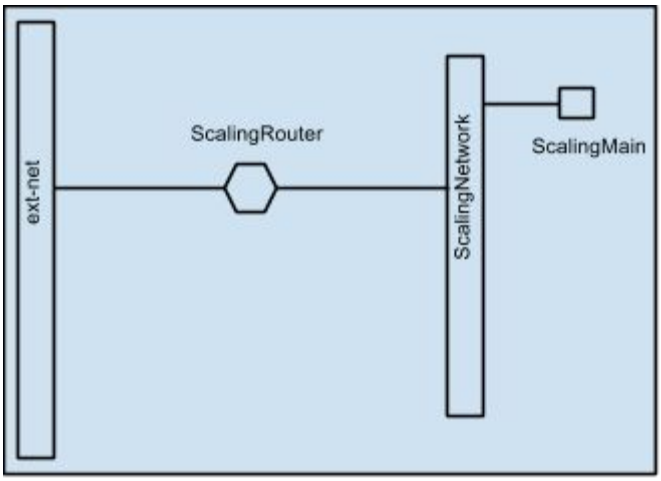
```
heat_template_version: 2013-05-23

description:
  # a description of the template

parameter_groups:
  # a declaration of input parameter groups and order

parameters:
  # declaration of input parameters

resources:
  # declaration of template resources

outputs:
  # declaration of output parameters
```

HOT structure

HOT created as part of autoscaling application

1. networkmain.yaml

This templates creates an infrastructure for creating auto-scaling resources. It creates a Router (ScalingRouter) which is connected to Network (ScalingNetwork) and an Instance (ScalingMain).



networkmain.yaml

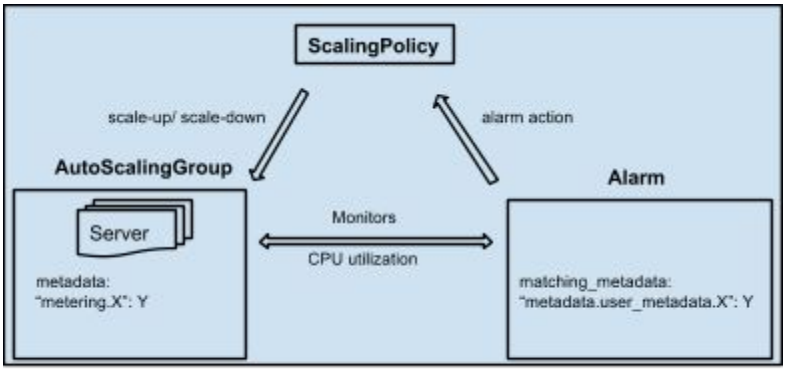
2.scalingmain.yaml

This is the main templates which is used for scaling. It has following main resource:

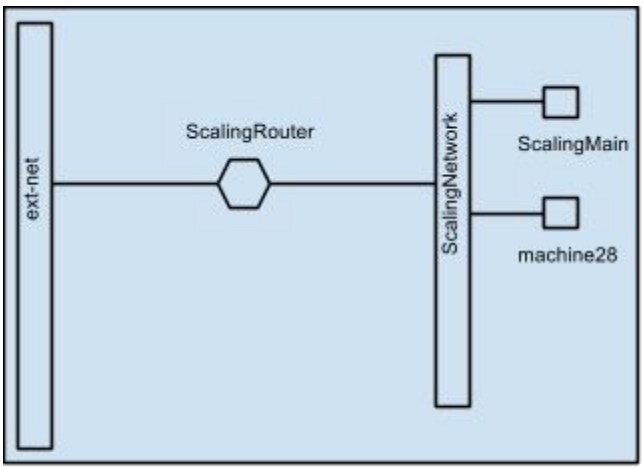
a. Heat::AutoScalingGroup: It is the group of resources that can be scaled. The template contains only Nova::Server as scaling-group member. Metadata has been set for server since alarm will check resources with this metadata (key, value pair) for triggering alarm.

b. Heat::ScalingPolicy: Defines the policy for change (add/remove) in scaling-group. The template includes two scaling policies. One for scaling-up and other for scaling down. Depending on the triggered alarm it will scale-up/ scale-down the AutoScalingGroup resources.

c. Ceilometer::Alarm: Defines the meter and condition to be monitored for triggering alarm. The template monitors cpu utilization (cpu_util meter) of resources having metadata as defined in AutoScalingGroup. This is important because by default alarm monitors cpu utilization for all the resource present in tenant. By setting matching-metadata we are restricting to monitor only those resource with specific metadata.



main resources for scaling

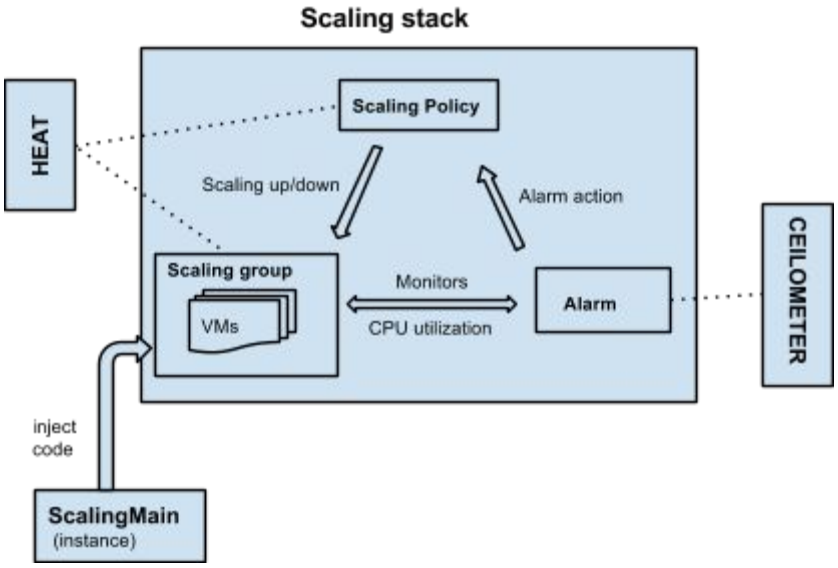


scalingmain.yaml

Workflow of autoscaling

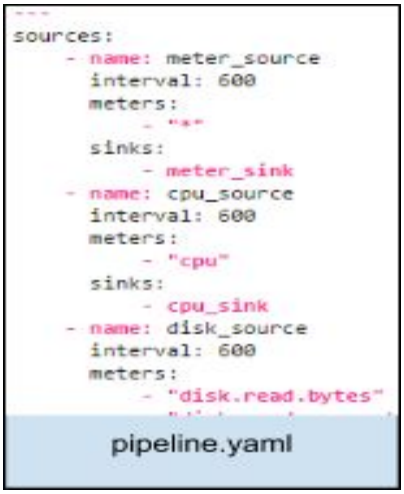
ScalingMain is used to inject code to any virtual machine of AutoScalingGroup. Initially AutoScalingGroup has one virtual machine. Overall two virtual machines on ScalingNetwork.

1. Using SSH inject code/kill process from ScalingMain to other virtual machine.
2. Ceilometer Alarms monitors the average cpu utilization of all the virtual machines of Scaling stack (part of scaling group in scalingmain.yaml).
3. If the average cpu utilization of stack is greater than or equal to 30% (mentioned in cpu_high alarm) or less than or equal to 10% (mentioned in cpu_low alarm) then alarm is triggered.
4. The alarm notifies the ScalingPolicy URL (as mentioned in alarm_action of ceilometer alarm definition in scalingmain.yaml). The time at which alarm is triggered depends on the interval of pipeline.yaml file and period of alarm.
5. The ScalingPolicy add or removes virtual machines depending which alarm has been triggered.
6. step 1-5 is repeated until specific condition is specified.

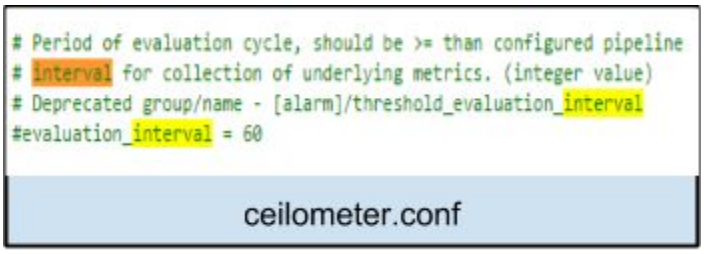


Interval at which alarm will be triggered

The time at which meter sample will be collected by Ceilometer Collector is defined in “pipeline.yaml”. By default the cpu_util meter sample are collected at every 600 seconds or 10 minutes (interval under meter ‘cpu’).



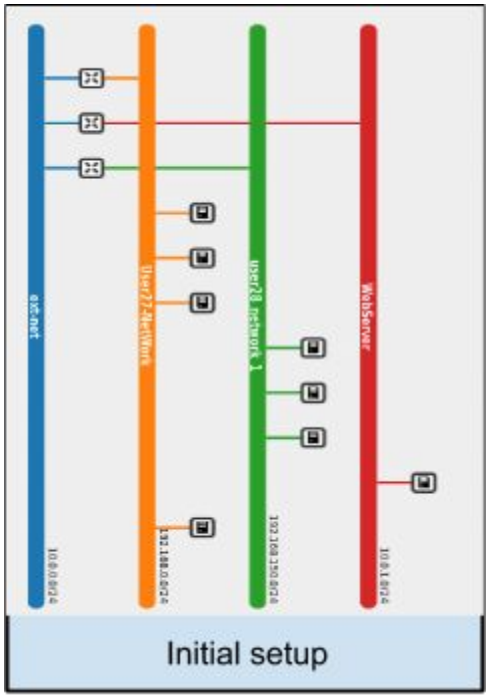
For auto-scaling application we have changed it to 60 so that samples are collected every 60 seconds. Also there is telemetry services configuration file “ceilometer.conf”. Here the interval for collection should be equal to or greater than interval in pipeline.yaml file.



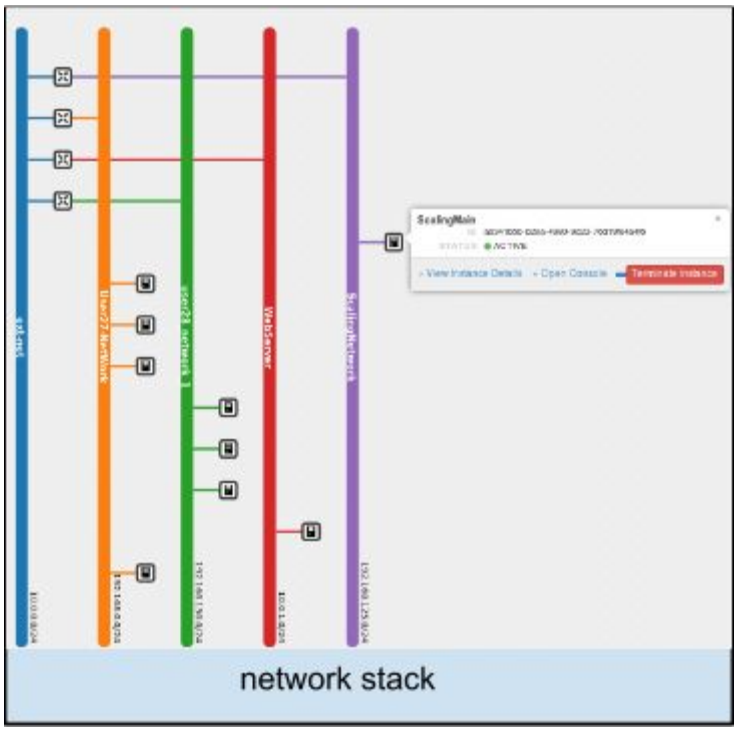
So whenever the interval is pipeline.yaml is changed, ceilometer.conf should also be updated if required.

Snapshots

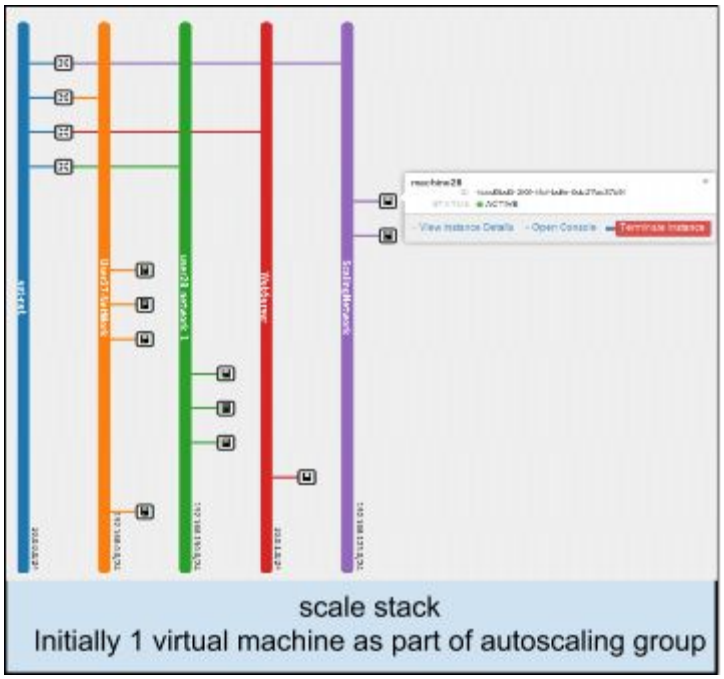
Initial state of network



Creating network stack from networkmain.yaml



Creating network scale from networkmain.yaml



scale stack
Initially 1 virtual machine as part of autoscaling group

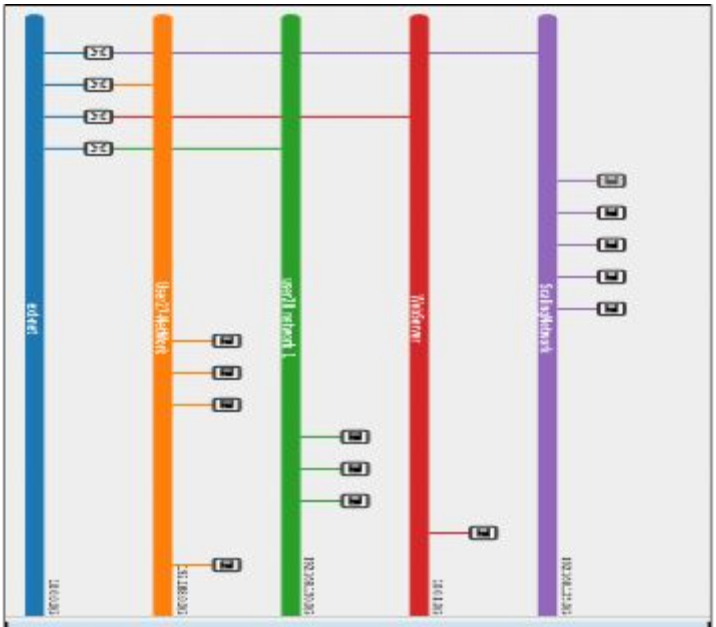
Injecting code (increasing cpu_util) from ScalingMain vm to machine28 vm

Connecting remotely using ssh
and injecting code by executing shell files remotely

```
Mem: 37276K used, 13612K free, 0K shrd, 752K buff, 2796K cached
CPU:   5% usr  40% sys   0% nic   0% idle   0% io   0% irq   0% sirq
Load average: 42.10 10.00 3.31 121/165 423
  PID  PPID USER     STAT   VSZ %VSZ %CPU COMMAND
  310     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  321     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  313     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  316     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  353     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  334     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  338     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  352     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  341     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  337     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  362     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  375     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  371     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  360     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  361     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  380     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  373     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  385     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  419     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  410     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
  393     1 cirros   R     3376   7%   1% sh /home/cirros/run.sh
```

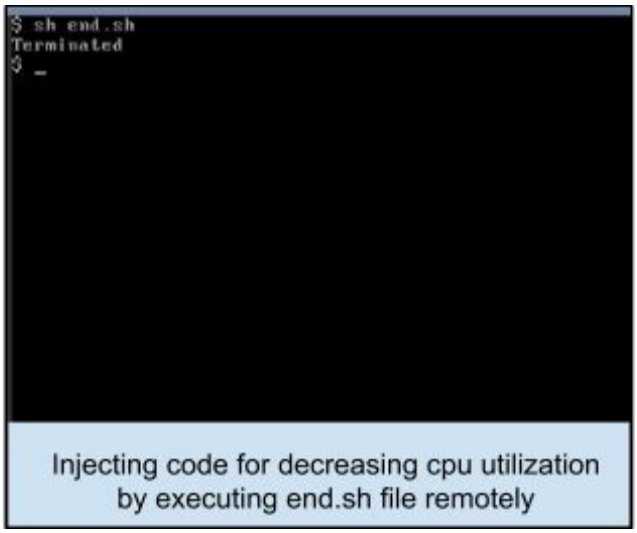
CPU utilization of machine28 using top command

Scaling up



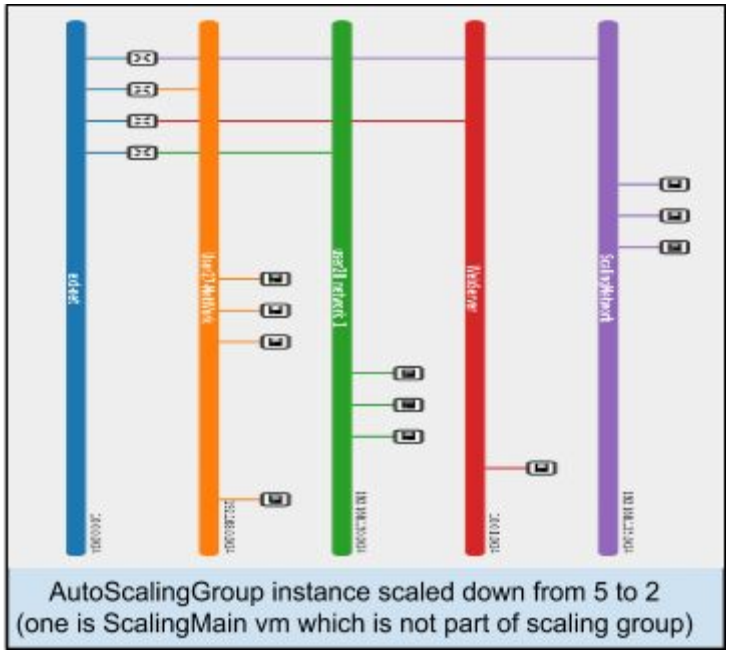
AutoScalingGroup instance scaled up from 1 to 4

Injecting code (decreasing cpu_util) from ScalingMain vm to machine28 vm



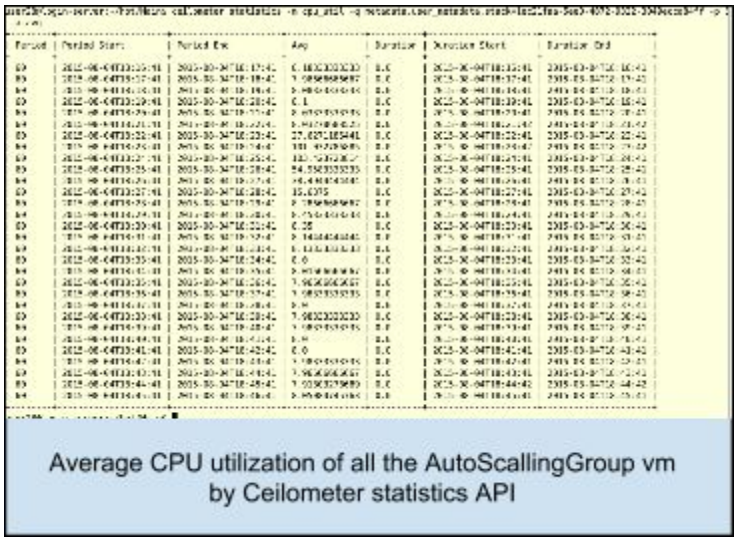
Injecting code for decreasing cpu utilization
by executing end.sh file remotely

Scaling down



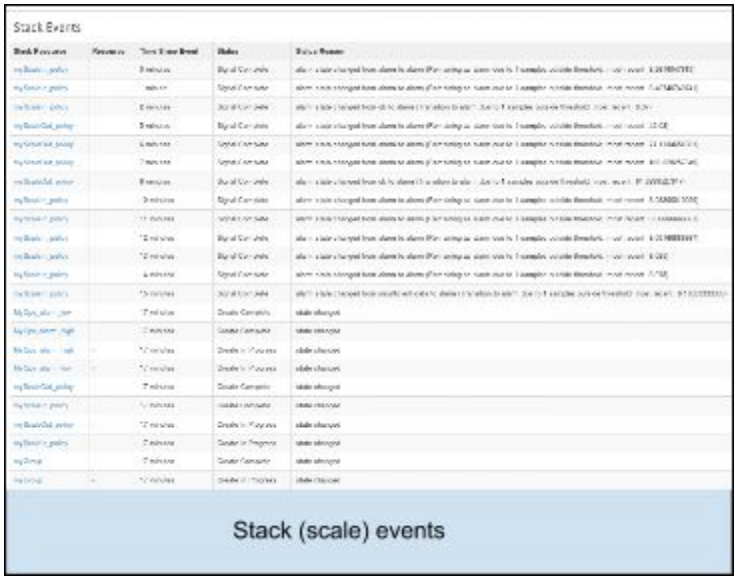
AutoScalingGroup instance scaled down from 5 to 2
(one is ScalingMain vm which is not part of scaling group)

Ceilometer statistics API



Average CPU utilization of all the AutoScallingGroup vm by Ceilometer statistics API

Stack event for checking alarm triggering time



Stack (scale) events

Ceilometer query for measuring specific resource cpu-utilization

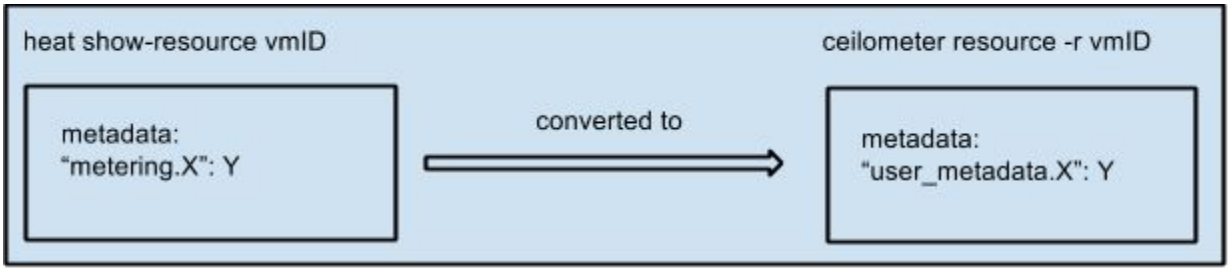
While creating Nova::Server (as part of AutoScalingGroup) we set following metadata:

metadata: {"metering.stack": {get_param: "OS::stack_id"}}

While creating Ceilometer::Alarm we set following matching_metadata:

matching_metadata: "metadata.user_metadata.stack": {get_param: "OS::stack_id"}

This metadata query is not same because when resource contains metadata in the form of *“metering.X: Y”* then ceilometer converts the metadata of resource to *“user_metadata.x: Y”.* And since we are checking the metadata the ceilometer query becomes *“metadata.user_metadata.x: Y”*



Conversion

OpenStack CLI

To check information of stack or any telemetry services OpenStack had provided API. For this application some of Heat and Ceilometer API are used. Following are the API used are

Heat API

To check stack present in tenant

heat stack-list

To create new stack

heat stack-create scale -f scalingmain.yaml //scale: name of stack, scalingmain.yaml: filename

To delete a stack

heat stack-delete scale //scale: name of stack to be deleted

To display stack information

heat stack-show scale //scale: stack name

Ceilometer API

To check alarm present in tenant

ceilometer alarm-list

To display alarm information

ceilometer alarm-show -a alarm_id //alarm_id: ID of alarm

To check sample-list of cpu_util meter

ceilometer sample-list -m cpu_util -q metadata.user_metadata.stack=stack_id

//-m cpu_util: meter for outputting sample, stack_id: id of stack

To check statistics

ceilometer statistics -m cpu_util -q metadata.user_metadata.stack=stack_id -p 120

//p: period for statistics should be displayed

To delete alarm

ceilometer alarm-delete -a alarm_id //alarm_id: ID of alarm

Code injection commands

To login to virtual machine remotely using ssh

ssh cirros@x.x.x.x //cirros: hostname, x.x.x.x floating IP

To start injecting code

sh start.sh //start.sh: shell file that runs multiple infinite loops in background

To kill all shell process

sh end.sh //end.sh: shell file that kill all the running shell process

References

- http://docs.openstack.org/developer/heat/template_guide/hot_spec.html
- https://ask.openstack.org/en/question/50124/heat-template-alarm/
- https://bugs.launchpad.net/heat/+bug/1356544
- https://ask.openstack.org/en/question/58566/heat-orchestration-scale-down-a-specific-instance/
- http://docs.openstack.org/admin-guide-cloud/content/section_telemetry-data-collection-processing.html
- http://docs.openstack.org/developer/ceilometer/configuration.html#pipeline-configuration
- http://docs.openstack.org/kilo/config-reference/content/section_ceilometer.conf.html
- http://docs.openstack.org/developer/ceilometer/architecture.html
- http://docs.openstack.org/developer/ceilometer/configuration.html#pipeline-configuration
- https://github.com/rbowen/presentations/blob/master/ceilometer/slides.md
- https://github.com/openstack/ceilometer/blob/master/ceilometer/compute/util.py#L34