Cloud orchestration in OpenStack: HEAT

HEAT

- Questo componente si occupa dell'orchestrazione dell'infrastruttura. Heat permette la creazione di applicazioni Cloud multiple e composite sulla base di template in formato testo che vengono trattati come codice. Nella terminologia di Heat un insieme di risorse che comunicano fra loro (VM, reti, volumi etc.) è denominato **stack:**
  - Sono integrabili in tool di configurazione automatizzata come Puppet
  - Possibilità di gestire alta affidabilità e autoscaling in maniera automatica
- Supportato dalla release Grizzly
- Il formato nativo per i template Heat è in evoluzione, ma Heat prevede anche di fornire compatibilità con il formato per i template AWS CloudFormation, in modo che sia possibile lanciare su OpenStack molti template esistenti CloudFormation.

HEAT: architettura

Heat è formato dal seguente insieme di applicazioni python:

- **heat**: il tool heat è una CLI che comunica con heat-api per eseguire le API AWS CloudFormation.
- **heat-api**: questo componente fornisce una ReST API nativa di OpenStack che processa le richieste API inviandole a heat-engine su RPC.
- **heat-api-cfn**: questo componente fornisce una Query API in stile AWS compatibile con AWS CloudFormation e processa le richieste API inviandole a heat-engine su RPC.
- **heat-engine**: questo componente svolge il lavoro principale di orchestrazione del lancio di template e restituisce gli eventi al consumer delle API.

HEAT: installazione

- Installazione tramite packstack:
  CONFIG_HEAT_INSTALL=y

- Troubleshooting: se QPID è configurato per implementare l'encryption dei dati, configurare heat in modo conseguente:
  qpid_port=5671
  qpid_protocol=ssl

HEAT: esempio di template VM su rete privata

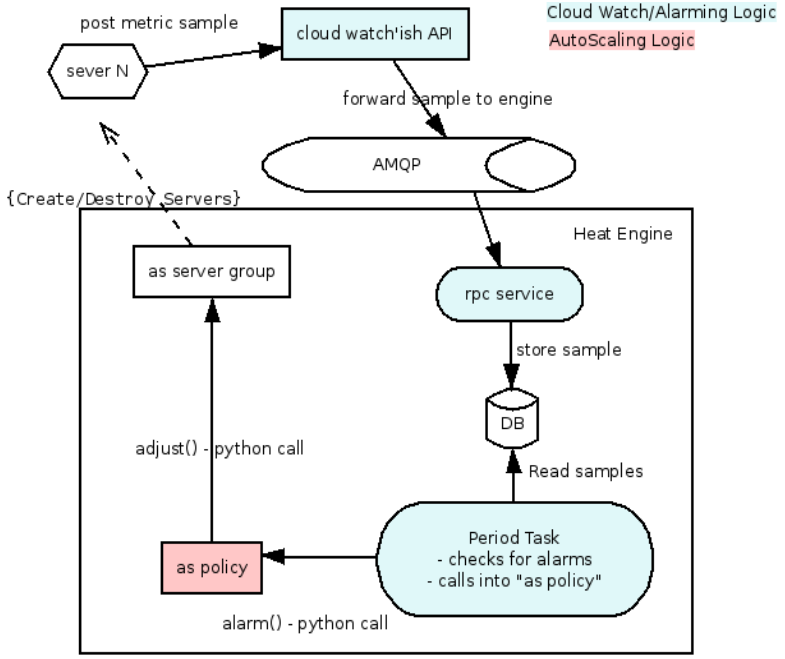
```
heat_template_version: 2014-05-14
description: Test Template

parameters:
  ImageID:
    type: string
    description: Image use to boot a server
  NetID:
    type: string
    description: Network ID for the server

resources:
  server1:
    type: OS::Nova::Server
    properties:
      name: "Test server Net"
      image: { get_param: ImageID }
      flavor: "m1.small"
      networks:
      - uuid: { get_param: NetID }

outputs:
  server1_private_ip:
    description: IP address of the server in the private network
    value: { get_attr: [ server1, first_address ] }
```

Autoscaling



In Havana:

- AWS::AutoScaling::AutoScalingGroup
- AWS::AutoScaling::ScalingPolicy

In IceHouse anche:

- OS::Heat::InstanceGroup
- OS::Heat::AutoScalingGroup
- OS::Heat::ScalingPolicy

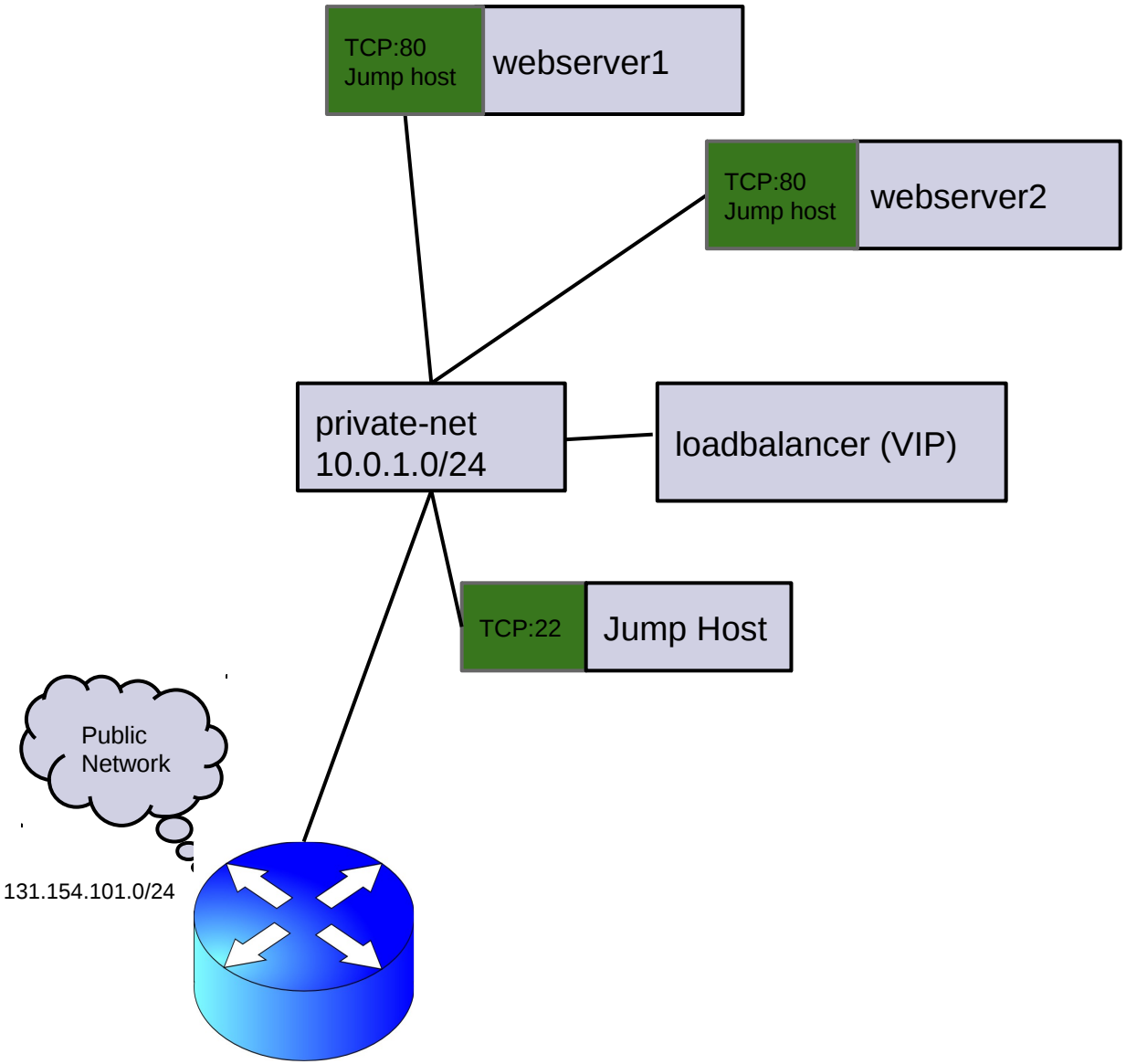
OS::AutoScale::**ScalingGroup** - a group that can scale an arbitrary set of heat resources.

OS::AutoScale::**ScalingPolicy** - affects the number of scaling units in a group (+1, -10%, etc)

In corso una discussione per una nuova **Autoscaling API**:

https://wiki.openstack.org/wiki/Heat/AutoScaling

Multi Tier app Cloud@CNAF hands-on



More complex templates

- All-in-one template formation per laboratorio hands-on Cloud@CNAF:
  https://gist.github.com/fabiok/e69426222972d1f58d03

- Autoscaling example – Web server with LbaaS:
  https://github.com/openstack/heat-templates/blob/master/hot/autoscaling.yaml

  Basato su AutoScalingGroup e ScalingPolicy

Accesso alle istanze

In Havana c'è solo la property:

```
'key_name': {
            'Type': 'String',
            'Description': _('Name of keypair to inject into
  the server')},
```

Definita in **OS::Nova::Server**

In Icehouse anche:

```
ADMIN_PASS: properties.Schema(
            properties.Schema.STRING,
            _('The administrator password for the server.'),
            required=False,
            update_allowed=True)
```

E inoltre è implementata **OS::Nova::KeyPair**

Metering in OpenStack: CEILOMETER

CEILOMETER

- **Misurazione del consumo delle risorse** per scopi di monitoring e accounting
- Questo aspetto è stato inizialmente lasciato fuori da OpenStack – la preferenza è stata di concentrarsi prima sulle caratteristiche IaaS di base
- Tuttavia il controllo dell'uso delle risorse è fondamentale **Ceilometer**
- Progetto iniziato a maggio 2012

CEILOMETER workflow

- Raccolta delle metriche di utilizzo dai componenti Openstack
- Eventuale aggregazione delle metriche raccolte
- Pubblicazione delle metriche verso diverse destinazioni incluso il collector di Ceilometer stesso
- Conservazione delle metriche di default in un database MongoDB
- Lettura delle metriche attraverso API Rest