xlcloud

Powering Computation in the Cloud

Workshop #1: Presentation of Heat

Patrick Petit

July 2012

xlcloud
Powering Computation in the Cloud

Agenda of presentation

- Heat Overview
- Heat Roadmap
- Heat basic architecture
- Heat CLIs
- Bootstrap methods
- CloudFormation helper scripts
- CloudFormation template
- Synchronization & rollback
- IAM resources
- Advanced Services

xlcloud
Powering Computation in the Cloud

Heat Overview

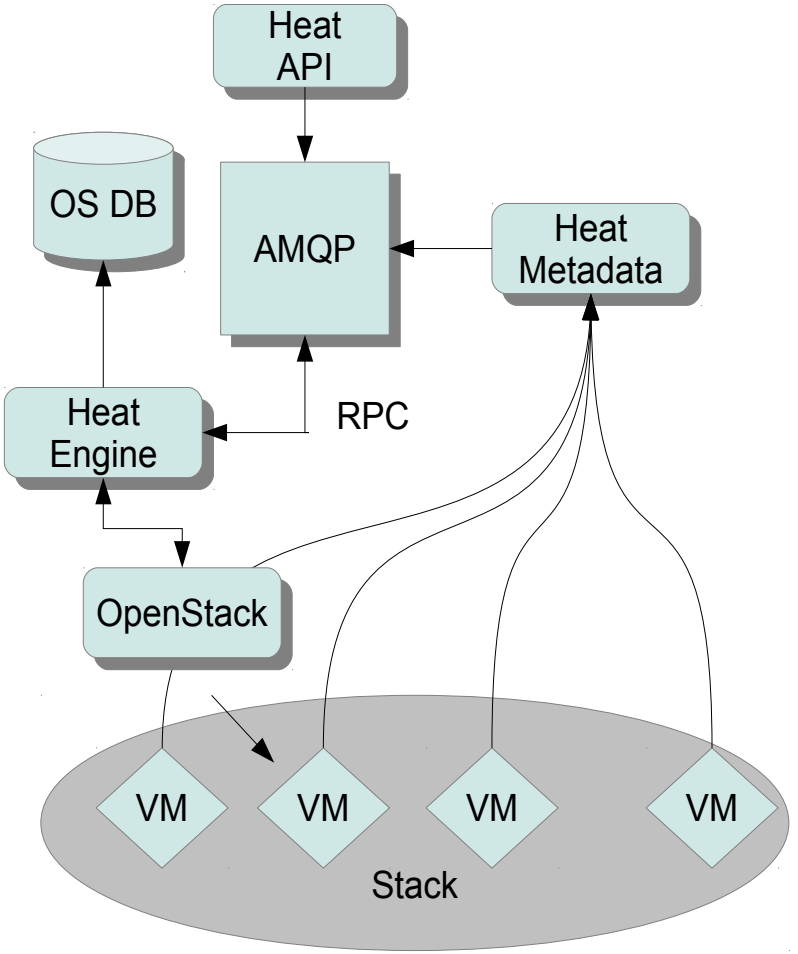
- Heat provides an AWS CloudFormation implementation for OpenStack (API and template) that orchestrates multiple composite cloud applications, called a ***stack,*** by executing a CloudFormation template
- All of the resources, installation, configuration, and startup commands are included in the CloudFormation template
- Allows creation of most resource types (such as instances, floating IPs, volumes, security groups, users, etc.) as well as some advanced services (such as high availability, auto-scaling and nested stacks)
- Heat orchestration is a whole or nothing asynchronous service that supports updating running stacks (not implemented yet)
- Compatible with AWS CloudFormation legacy (heat CLI based on boto and compatible API)
- Integrates well with Puppet and Chef
- OpenStack style project
  - Tight integration with other OpenStack projects (Eg. Glance, Keystone, Nova)
  - Python 2, matching OpenStack design principals
  - Open-source (ASL V2) since inception in March 2012 hosted on Github
  - Integrated with Stackforce (OpenStack workflow gerrit/jenkins)
- Very active community of 9 developers from Red Hat

xlcloud
Powering Computation in the Cloud

Heat Roadmap

- Targeted at Folsom:
  - Complete integration with Common, Glance, Keystone, Swift, and Nova
  - Complete implementation of the AWS CloudFormation API
  - Usable implementation of AWS CloudWatch API
  - Complete implementation for all non-VPC related resource types in CloudFormation
  - Instance and application high availability
  - Autoscaling
  - Nested Stacks
- Targeted at G release:
  - Project Incubation
  - Optimizing project governance to match OpenStack standards
  - Complete implementation of AWS CloudWatch API, contributing appropriate technology into Ceilometer
  - Complete integration with Quantum, providing complete VPC feature coverage
- Targeted at H release:
  - Hardening of source tree
  - Improving source tree to meet OpenStack design principles
  - Promotion to OpenStack core

Heat basic architecture



- Heat-api: is a service that exposes a CloudFormation REST API to the heat-engine. Communications between the heat-api and heat-engine uses RPC (requests and events)
- Heat-metadata: REST API server to access and manipulate metadata of the stacks. Also allows instances to send usage statistics (similar to the CloudWatch functionality)
- Heat-engine: is the heat project server. Heat engine does all the orchestration work and is the layer in which the resource integration is implemented

xlcloud
Powering Computation in the Cloud

Heat command line

heat <command> [options] [args]

Commands:

```
    create          Create the stack
    delete          Delete the stack
    describe        Describe the stack
    update          Update the stack
    list            List the user's stacks
    gettemplate     Get the template
    estimate-template-cost     Returns the estimated monthly cost of a template
    validate        Validate a template
    event-list      List events for a stack
    resource        Describe the resource
    resource-list   Show list of resources belonging to a stack
    resource-list-details    Detailed view of resources belonging to a stack
```

Example:

```
# heat -d create wordpress --template-file=WordPress_Single_Instance_With_HA.template \
--parameters="InstanceType=m1.xlarge;DBUsername=foo;DBPassword=bar;KeyName=mykey"
```

xlcloud
Powering Computation in the Cloud

Heat-jeos command line

heat-jeos <command> [options] [args]

Commands:

create	Create a JEOS image and register it with OpenStack
tdl	Prepare a template ready for Oz
image	Build an image from the specified template
register	Register the built image with OpenStack Glance

Here are some examples:

heat-jeos create F16-x86_64-cfntools-jeos

Create a Fedora 16 image from the bundled template.

heat-jeos create --template-file ~/templates/my.tdl

Create an image from a custom template.

heat-jeos create --gold --template-file ~/templates/my.tdl --iso /var/isos/my.iso

Create a golden image from a custom template and iso file

xlcloud
Powering Computation in the Cloud

Oz disk image template

```
<template>
  <name>F16-x86_64-cfntools-jeos</name>
  <os>
    <name>Fedora</name>
    <version>16</version>
    <arch>x86_64</arch>
    <install type='iso'>
      <iso>file:/var/lib/libvirt/images/Fedora-16-x86_64-DVD.iso</iso>
    </install>
  </os>
  <description>Fedora 16</description>
  <commands>
    <command name='commands'>
yum -y update --skip-broken;yum -y install yum-plugin-fastestmirror;yum -y update;/usr/sbin/useradd ec2-user;echo -e 'ec2-
user\tALL=(ALL)\tNOPASSWD: ALL' >> /etc/sudoers;yum -y install cloud-init;cat >> /etc/rc.d/rc.local &lt;&lt; EOF;chmod +x
/etc/rc.d/rc.local;chmod +x /opt/aws/bin/cfn-*
#!/bin/bash
setenforce 0
EOF
    </command>
  </commands>
  <files>
    <file name='/opt/aws/bin/cfn-init'  type='base64'></file>
    <file name='/opt/aws/bin/cfn-hup'  type='base64'></file>
    <file name='/opt/aws/bin/cfn-signal'  type='base64'></file>
    <file name='/opt/aws/bin/cfn_helper.py'  type='base64'></file>
    <file name='/opt/aws/bin/cfn-get-metadata'  type='base64'></file>
  </files>
</template>
```

xlcloud
Powering Computation in the Cloud

Several bootstrap methods are possible

- Create image with application ready to go (golden image)
- Use CloudFormation and cloud-init to run a startup script passed as user-data to the nova boot command
- Use CloudFormation metadata and helper scripts (based on cloud-init) that allow to update your metadata after your instance and applications are up and running
- All the above complemented with Chef or Puppet clients

CloudFormation helper scripts

- These scripts are installed by default in images created with heat-jeos. These scripts are not executed by default...
  - ***cfn-init***: used to execute the resource's metadata (install packages, create files and start services
  - ***cfn-signal***: a wrapper script to signal a WaitCondition resource allowing to wait for an application to be ready before continuing with the stack creation
  - ***cfn-hup***: a daemon to handle updates to metadata and execute custom hooks when changes are detected
  - ***cfn-get-metadata***: a wrapper script making it easy to retrieve either all metadata defined for a resource or a specific key or subtree of the resource's metadata
  - ***cfn-update-stack***: a wrapper script to update metadata content

Anatomy of a CloudFormation Template

- The template consists of five top-level JSON objects:
  - **Description**: Text description of the template
  - **Parameters**: Input parameters to the template to specify runtime parameters like key-pair, instance type, database name, etc.
  - **Mappings**: Like a hash table. Used for example to map the proper architecture to the instance type so that the template user need specify only the instance type (Eg. "c1.xlarge" : { "Arch" : "64" })
  - **Resources**: The only required JSON objects in the template. Used to describe resources such as instance, volume, security group, floating IP, ... Resources have metadata, properties and user-data sections
  - **Outputs**: Used to return application's runtime information like the public URL for a newly created website

xlcloud
Powering Computation in the Cloud

Resource Metadata

- The cfn-init script uses the resource's metadata block rooted by the ***AWS:CloudFormation::Init*** metadata key

```
"Resources": {
     "MyInstance" : {
          "Type": "AWS::EC2::Instance",
          "Metadata" : {
               "Param" : {"Ref":"ParamName"},
               "AWS::CloudFormation::Init": {
                    "config" : {
                         "sources" : {
                              :
                         },
                         "files" : {
                              :
                         },
                         "packages" : {
                              :
                         },
                         "services" : {
                              :
                         }
                    }
               }
          }
     }
```

Sources allows to download an archive file and unpack it in a target directory (tar, tar+gzip, tar+bz2 and zip)

Files allows to create arbitrary files. The content can be either inline or pulled from a URI.

Packages allows to download and install packages (apt, yum, rubygems, rpm and python)

Services allows to define which services should be enabled or disabled when the instance is launched. It also allows to specify dependencies on sources, packages and files so that if a restart is needed due to files being installed, cfn-init will take care of restarting the service.

xlcloud
Powering Computation in the Cloud

Interpreting the resource metadata

- The ***cfn-init*** bootstrap script interprets the resource's metadata block containing the sources, packages, files and services keys when the instance is launched
- Shall use the access key of an account with permission to call DescribeStackResource (not currently supported)
- ***cfn-init*** has the following syntax:

```
cfn-init   --access-key access.key \
           --secret-key secret.key \
           --credential-file | -f credential.file \
           --resource | -r logical.resource.id \
           --region region
#!/bin/bash
/opt/aws/bin/cfn-init -s <stackname> -r <resourcename>
--region <region> --access-key <accesskey> --secret-key
<secretkey>
```

xlcloud
Powering Computation in the Cloud

Template example #1

- Check WordPress_Single_Instance_With_EBS_EIP.template

xlcloud
Powering Computation in the Cloud

Stack synchronization and rollback

- Heat supports a ***WaitCondition*** resource that is used to synchronize resource creation and to make sure that either all of your stack's resources are created or none of them are
- The WaitCondition resource pauses execution of the template until a specified condition is met or a timeout period is exceeded
- To wait for the application to be ready, you can use the ***cfn-signal*** script to signal the application installed successfully or failed
- ***cfn-signal*** has the following syntax

```
cfn-signal --success | -s signal.to.send \
           --reason | -r resource.status.reason \
           --data | -d data \
           --id | -i unique.id \
           --exit-code | -e exitcode \
           waitcontionhandle.url
```

xlcloud
Powering Computation in the Cloud

Template example #2

- For a simple WaitCondition example check MySQL_Single_Instance.template

xlcloud
Powering Computation in the Cloud

IAM resources

- Heat integrates with Keystone to create users and keys from within the template
- But currently policies are not supported because there is no nice correlation between keystone's roles and policy statements
- AWS::IAM::User resource type
  - Used to create a keystone user
  - Useful to create an ephemeral user for the lifetime of the stack
- AWS::IAM::AccessKey resource type
  - Creates an ***AccessKey*** an assign it to the IAM user
  - You can get the ***SecretKey*** from the ***AccessKey*** using the ***Fn::GetAtt function*** and display the result in the output declarations of the template like in:

```
"AccessKeyformyaccesskey" : { "Value" : {"Ref" : "myaccesskey"} },
"SecretKeyformyaccesskey" : { "Value" : {"Fn::GetAtt" : ["myaccesskey",
"SecretAccessKey"]} }
```

Example: use of user credentials in an instance

```
"myuser" : {
    "Type" : "AWS::IAM::User",
    "Properties" : {
        "LoginProfile" : {
            "Password" : "verybadpasswd",
    },
},

"myaccesskey" : {
      "Type" : "AWS::IAM::AccessKey",
      "Properties" : {
        "UserName" : {"Ref": "foo"}
      }
},

"myinstance" : {
    "Type" : "AWS::EC2::Instance",
    "Properties" : {
        "UserData" : {"Fn::Base64" : {"Fn::Join" : [ "", [
            "ACCESS_KEY=", { "Ref" : "myaccesskey" }, "&",
            "SECRET_KEY=", {"Fn::GetAtt" : ["myaccesskey", "SecretAccessKey"]} ]
```

xlcloud
Powering Computation in the Cloud

Intrinsic functions

- Heat supports most AWS intrinsic function that help manage your stack
  - ***Fn::Base64***: returns the Base64 representation of the input a string
  - ***Fn::FindInMap***: returns the value of a key from a mapping declared in the Mappings section
  - ***Fn::Join***: appends a set of values into a single value, separated by the specified delimiter.
  - ***Ref***: returns the value of the specified parameter or resource logical name. The value is generally the name of the resource but can be a more meaningful identifier (Eg. instance ID, or parameter value like in “Ref” : “DBName”)
  - ***Fn::GetAtt***: returns the value of an attribute from a resource in the template. Eg. "Fn::GetAtt" : ["MyInstance" , "PublicIP"]. Resources may have no or multiple attributes.

But Heat is much more than a resource brokering service

Heat targets also advanced cloud services integration in the stack template

xlcloud
Powering Computation in the Cloud

Ongoing Support of AWS AutoScaling

- AWS::AutoScaling::AutoScalingGroup resource type
  - Creates an auto-scaling group.
  - Useful to launch a bunch of compute nodes at once, so you can do this:

```
heat create hpc-cluster -f ./templates/hpc-cluster.template
--parameters="NumInstances=50"
```

- AWS::AutoScaling::LaunchConfiguration resource type
  - Example of tight integration with OpenStack's specific capabilities like ***NovaSchedulerHints*** resource property to send arbitrary key/value pairs to the scheduler.
- AWS::AutoScaling::ScalingPolicy resource type
  - adds a scaling policy to an auto scaling group

xlcloud
Powering Computation in the Cloud

Template examples #3

- Check this simple template that uses AutoScalingGroup to create a cluster of compute nodes *BasicAutoScaling.template*

xlcloud
Powering Computation in the Cloud

Ongoing Implementation of CloudWatch

- AWS::CloudWatch::Alarm resource type
  - Creates a CloudWatch alarm
  - Useful to support instance and application high availability
  - Currently the only CloudWatch action that is supported is HEAT::HA::Restarter
  - Uses metadata server to communicate application stats
- HEAT::HA::Restarter resource type
  - Restart an instance

xlcloud
Powering Computation in the Cloud

Template examples #4

- Check the *WordPress_Single_Instance_With_HA.template*

xlcloud
Powering Computation in the Cloud

References

- http://wiki.openstack.org/Heat
- https://github.com/heat-api
- https://github.com/heat-api/heat/tree/master/templates
- http://docs.amazonwebservices.com/AWSCloudFormation/latest/APIReference/API_CreateStack.html
- http://docs.amazonwebservices.com/AWSCloudFormation/latest/UserGuide/create-stack.html
- http://docs.amazonwebservices.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html
- http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

xlcloud
Powering Computation in the Cloud

Table sample

Col 1	Col 2	Col 3	Col 4	Col 5
Table text				

Thank you for your attention