# Implementation and testing of OpenStack Heat

**September 2013**

Author:
Davide Michelino

Supervisor(s):
Jose Castro Leon
Luis Fernandez Alvarez

**CERN openlab Summer Student Report 2013**

## Project Specification

CERN is establishing a large scale private cloud based on OpenStack as part of the expansion of the computing infrastructure for the LHC.

Many cloud based services use auto-scaling and orchestration to expand and contract their resources according to user load. The OpenStack Heat project provides an open source framework to organize the configuration and deployment of cloud applications.

After the implementation of a working environment, was tested a sample use case, developing a template that deploys webservers within an auto scaling group behind a Load Balancer and serving webpages hosted on a NFS server. Webservers scale up and scale down according to current load.

## Abstract

The aim of this document is to describe the project that was implemented during the openlab Summer Programme ‘Implementation and testing of OpenStack Heat’.

This document gives a quick brief of what “Cloud Computing” is and which are the technologies and models used to build a Cloud Computing infrastructure; then it gives an overview on OpenSatck project.

The main part of this document gives details on how Heat works and how it has been integrated in the OpenStack project, and gives a reference on how to install and use it.

The last part describes a use case that has been deployed to test Heat features, giving some details about the template that implement it.

# Table of Contents

# 1 Introduction

OpenStack Heat is an incubated OpenStack project that provides cloud orchestration features, including high availability and auto scaling ones.

Before start to describe OpenStack Heat, will be given an overview about Cloud Computing technologies with some details on the service model that OpenStack implements (Infrastructure as a Service).

# 2 Cloud Computing

Cloud Computing is an expression used to describe a set of computer science technique and technologies that allow to access to computational and storage resources through a network. Usually these resources belong to heterogeneous and distributed sets, and they are served using virtualization technologies.

The aim of Cloud Computing is to abstract the whole infrastructure, composed by network resources as well as hardware resources, exposing some API (Application Programming Interface) to users, that communicate with the services that will manage the "cloud".

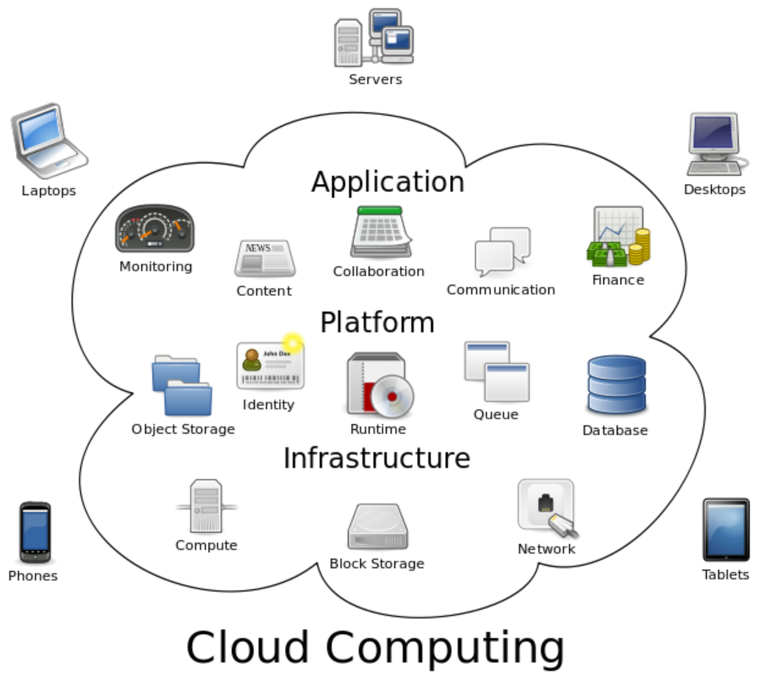


*Figure 1. Cloud Computing logic graph.*

## 2.1 The IaaS service model

Infrastructure as a Service is the most basic cloud service model. In this model, a provider offers resources to clients in order to allow them to build their own “virtual” infrastructure; moreover, it offers abstractions to build higher level service model, like Platform as a Service and Software as a Service.

Usually the resources belong to a pool of hardware managed by cloud software and clients can access them using an interface.

# 3 OpenStack

OpenStack is a project to provide an IaaS cloud computing service model. The project is released under the terms of the Apache License and it is managed by the OpenStack Foundation. Since its first release in July 2010, by NASA and Rackspace effort, more then 200 companies joined the project.

OpenStack consists of a series of interrelated projects written in python that offer all the main services an IaaS software should provides. The main core projects belonging to OpenStack are:

- Compute service (Nova)
- Object storage service (Swift)
- Block storage service (Cinder)
- Networking service (Neutron)
- Dashboard (Horizon)
- Identity service (Keystone)
- Image service (Glance)
- Metrics service (Ceilometer)

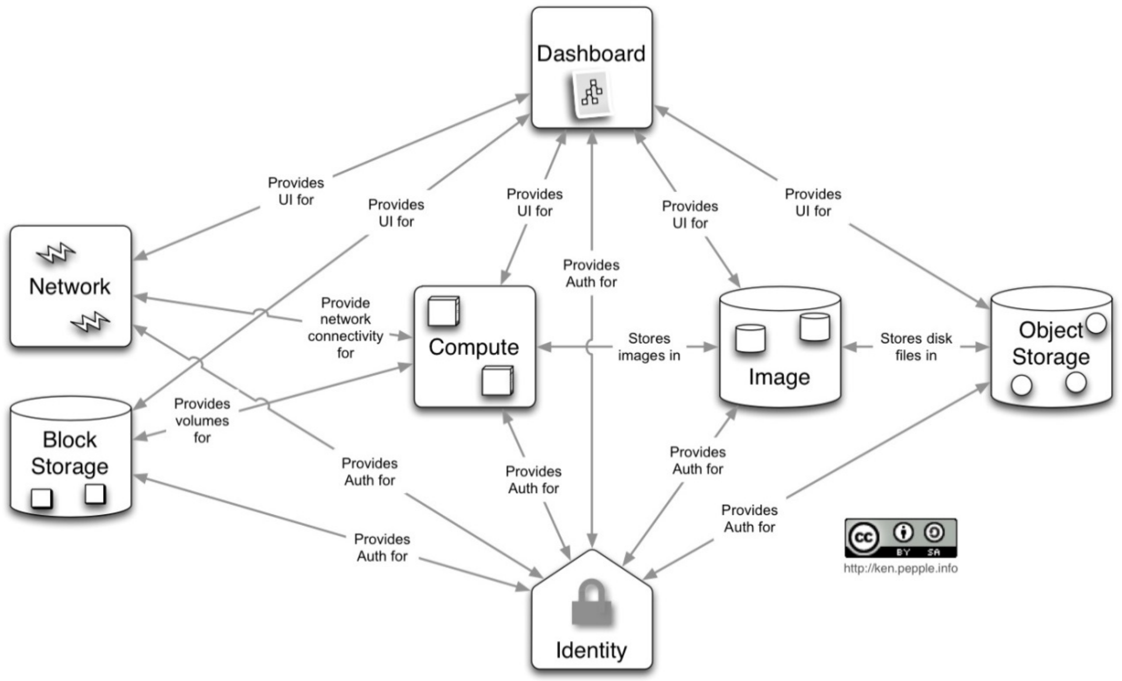


*Figure 2.* *OpenStack logical architecture*

## 4 Heat

Heat is an OpenStack incubated project that provides Orchestration for OpenStack. It allows describing cloud infrastructure or composed application (see fig. 3), called “stack”, in a text file called “template”.

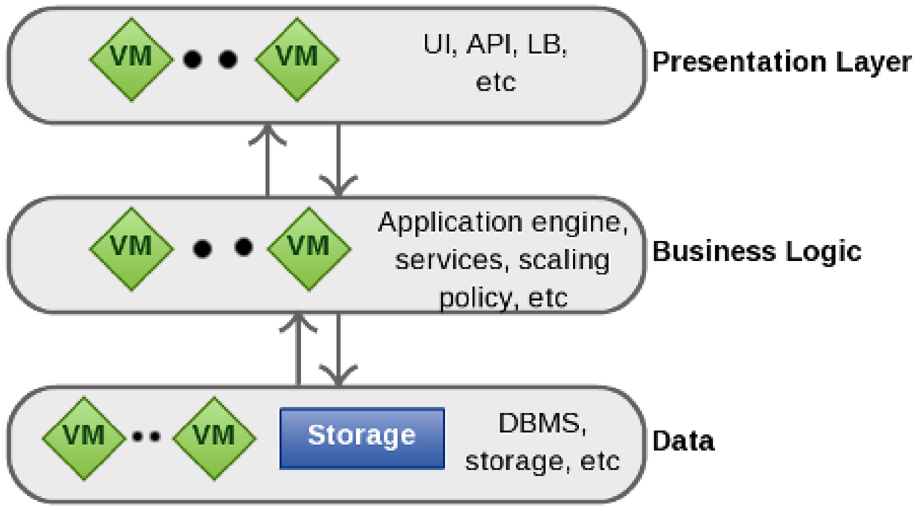


*Figure 3. An example of 3-tier model composed application*

Heat exposes some API that clients can use to send templates to the Heat engine, which parse them and it will communicate with all OpenStack services to deploy the resources and create the stack.

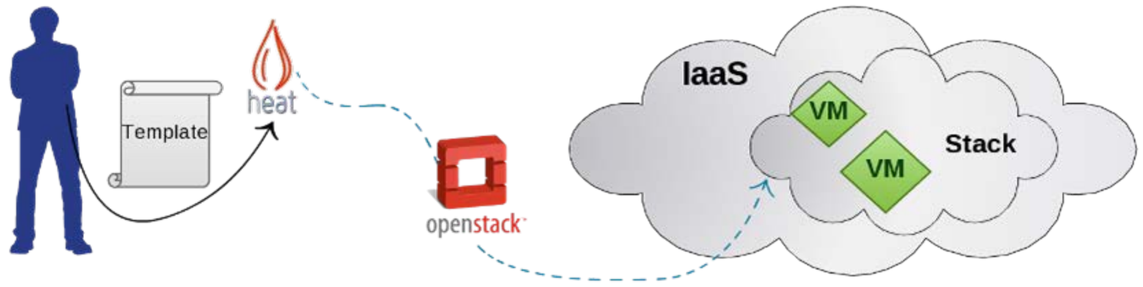


*Figure 4. How heat deploys a stack*

Moreover Heat provides auto scaling and high availability features within the stacks, so it can automatically add and destroy virtual machines according to the workload and it can restart services when something stop working correctly.

## 4.1 How Heat works from infrastructure side

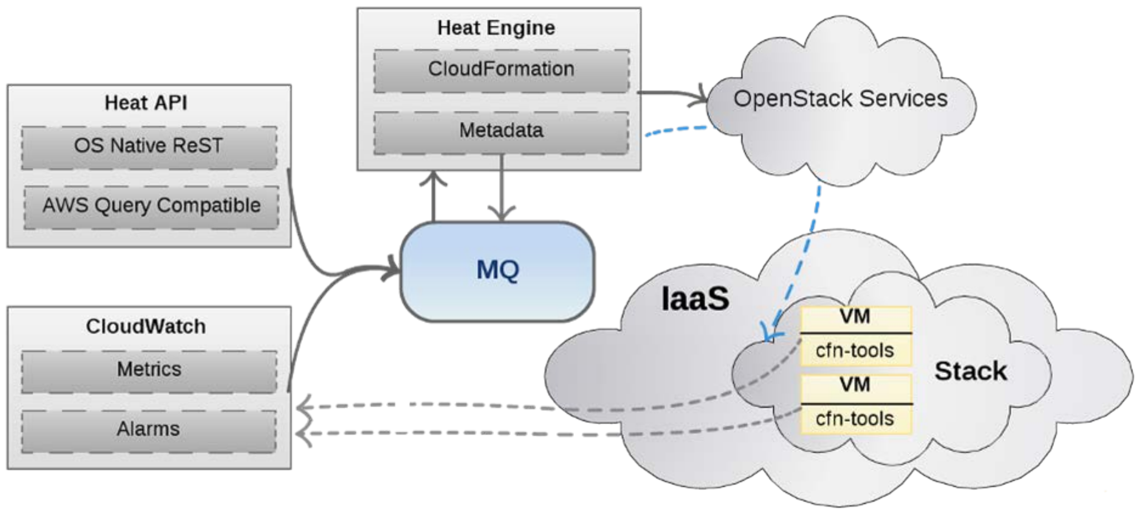


*Figure 5. Heat – Infrastructure Side*

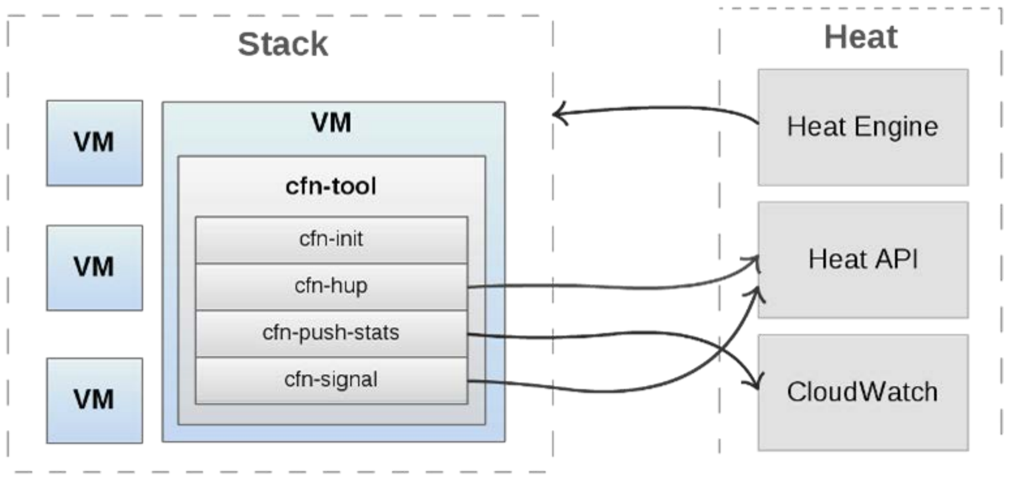
From infrastructure side, Heat is composed by three main services: Heat engine, Heat API and CloudWatch. Heat engine is the core service, it parses templates and provides resources, which are the basic elements of a stack; after it parsed a template, it will deploy and configure the resources in the cloud.

Heat exposes an openstack native restful API and an Amazon AWS Query compatible API through the Heat API service.

CloudWatch is the service that collects the metrics which virtual machines send to it, in order to monitor the status of the running stacks. It can manage alarms to trigger some action when metrics go above or below user defined values, so it provides auto scaling and high availability features.

All Heat services communicate with a message broker to send message among them, heat engine is the only service that communicate directly with external openstack services.

## 4.2 How Heat works from instance side



Within the instances Heat needs some helper scripts and daemons in order to provide CloudFormation features.

“cfn-tool” package provides all the helper scripts and daemons that heat needs in order to work, and it includes:

- cfn-init: the helper script that initialize the instance at boot time, executing userdata.
- cfn-hup: the daemon that periodically checks services status for High Availability, sends metrics to CloudWatch and execute hooks when events occur.
- cfn-push-stats: the helper script that send metrics to CloudWatch service.
- cfn-signal: the helper script that sends signals to Heat API.

## 4.3 Heat Template

Stack templates are json or yaml formatted text file that describe stacks in a declarative form. Currently Heat templates are aligned with template model provided by AWS CloudFormation templates.

The template structure is the following:

```
{
    "AWSTemplateFormatVersion" : "version date",
    "Description" : "Valid JSON strings up to 4K",

    "Parameters" : {
        keys and values #usually input parameters
    },
    "Mappings" : {
        keys and values #usually map input parameters with hard-coded strings
    },

    "Resources" : {
        keys and values #heat support the following Resource Types:
AWS::AutoScaling::LaunchConfiguration, AWS::AutoScaling::AutoScalingGroup,
AWS::AutoScaling::ScalingPolicy, OS::Heat::InstanceGroup,
AWS::CloudWatch::Alarm,
AWS::RDS::DBInstance,
AWS::EC2::EIP': ElasticIp, AWS::EC2::EIPAssociation': ElasticIpAssociation,
AWS::EC2::Instance, OS::Heat::HARestarter,
AWS::EC2::InternetGateway, AWS::EC2::VPCGatewayAttachment,
AWS::ElasticLoadBalancing::LoadBalancer,
AWS::EC2::NetworkInterface, AWS::EC2::RouteTable,
AWS::EC2::SubnetRouteTableAssocation,
AWS::S3::Bucket,
AWS::EC2::SecurityGroup,
AWS::CloudFormation::Stack,
AWS::EC2::Subnet,
OS::Swift::Container,
AWS::IAM::User, AWS::IAM::AccessKey, OS::Heat::AccessPolicy,
AWS::EC2::Volume, AWS::EC2::VolumeAttachment,
AWS::EC2::VPC,
AWS::CloudFormation::WaitCondition, AWS::CloudFormation::WaitConditionHandle
    },

    "Outputs" : {
       keys and values #are the output values of the stack (eg. webserver ip
address)
    }
}
```

### 4.3.1 Functions

Heat provides some functions to allow data manipulation within the templates in order to write “dynamic” templates. The functions provided are:

- Fn::FindInMap: returns the value corresponding to keys into a map declared in the Mappings template section.
- Fn::Base64: returns the Base64 representation of the string, it is useful to pass encoded UserData to instances.
- Fn::Join: appends a set of values into a single value, separated by the specified delimiter. You can concatenate values using an empty string delimiter. You can use it to concatenate strings with function outputs.

## 4.4 Heat Resources

Resources are the most basic element within a stack. They are provided by heat engine and are described by user in the template. In the following paragraphs some of the main resources are described.

### 4.4.1 Single Resource

#### 4.4.1.1 Instance

Instance resource describes a single instance to be deployed in the stack.

How to describe an Instance Resource:

```
"Resources" : {
    "ResourceName": {
      "Type": "AWS::EC2::Instance",
      "Metadata" : {  # Metadata to be injected into the instance
        "AWS::CloudFormation::Init"  :  {  #  CloudFormation  Configuration,  to
install packages and enable service
```

```
          "config" : {
            "packages" : {
              "yum" : {
                "package1"        : [],
                "package2"        : []
              }
            },
            "services" : {
              "systemd" : {
                "service1"    : { "enabled" : "true", "ensureRunning" : "true"
},
                "service2"    : { "enabled" : "true", "ensureRunning" : "true"
},
                "service3"   : { "enabled" : "false", "ensureRunning" : "false"
}
              }
            }
          }
        }
      },
     "Properties": { # Instance proprieties
        "ImageId" : "image-name-to-launch", # Glance Image to start
        "InstanceType"   : "Instance-flavor", # Flavor name to use
        "KeyName"        : { "Ref" : "KeyName" }, # the Resource Key to be used
for access CloudFormation services
        "UserData"              :  {  "Fn::Base64"  :  "#!/bin/bash  -v\n  echo
\"Userdata!\"" } # UserData to inject
      }
    }
```

### 4.4.1.2 Security Groups

SecurityGroup resource describes Security Group to be created in the stack

How to define a Security Group Resource:

```
    "ResourceName" : {
      "Type" : "AWS::EC2::SecurityGroup",
      "Properties" : {
        "GroupDescription" : "Description", # Resource description
        "SecurityGroupIngress" : [
          {"IpProtocol"  :  "protocol",  "FromPort"  :  "1",  "ToPort"  :  "65535",
"CidrIp"  :  "0.0.0.0/0"},  #  Security  Group  rule,  protocol  can  be  "icmp",  "tcp"
or "udp"
        ]
      }
    }
```

### 4.4.1.3 Elastic Load Balancer

ElasticLoadBalancer is the heat resource that provides a ready-to-use fedora instance with haproxy service installed. Heat will configure haproxy according to the user needs. Heat can deploy the LoadBalancer also on top of an Auto Scaling Group, updating the server list to balance automatically. Users just need to define a LoadBalancer resource and link it with instance(s) or ASG to be balanced.

How to define LoadBalancer Resource:

```
    "LoadBalancerName" : {
      "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
      "Properties" : {
        "AvailabilityZones" : { "Fn::GetAZs" : "" },
        "Instances"  :  [{"Ref":  "ServerOne"},  {"Ref":  "ServerTwo"},  {"Ref":
"ServerThree"}],  # List of the instances to balance
        "Listeners" : [ {
          "LoadBalancerPort" : "80",    # Specifies the port that Load Balancer
will listen on
          "InstancePort" : "80",    # Specifies the port to balance
          "Protocol" : "HTTP"       # Specifies the protocol to balance
        }],
        "HealthCheck" : {           # Specifies some parameters to monitor the
servers' health
          "Target" : "HTTP:80/",
          "HealthyThreshold" : "3",
          "UnhealthyThreshold" : "5",
          "Interval" : "30",
          "Timeout" : "5"

      }
    }
```

### 4.4.2 High Availability

High Availability is the feature that allows Heat to monitor services within instances and trigger some action on failures. In order to use High Availability features we need to define a HARestarter resource that create the restarter service policy whitin the instance, an Alarm resource that will trigger the HARestarter and configure the Instance in a specific way. In order to allow services within the instance to communicate with the external Heat services we need to create an Access Key using the AccessKey resource.

#### 4.4.2.1 Access Key Resource

It's the resource that defines an Access Key to authenticate services within the instance against Heat CloudFormation services.

How to define an Access Key Resource:

```
   "UserName" : { # The Username used for the Access key
      "Type" : "AWS::IAM::User"
    },
    "KeyName" : { # The name of the key
      "Type" : "AWS::IAM::AccessKey",
      "Properties" : {
        "UserName"  :  {"Ref":  "UserName"}  #  The  username  choosed  in  the
AWS::IAM::User resource
      }
```

#### 4.4.2.2 HARestarter Resource

It's the resource that creates a restart policy for the High Availably services.

How to define an HARestarter Resource:

```
    "RestartPolicyName" : {
      "Type" : "OS::Heat::HARestarter",
      "Properties" : {
        "InstanceId" : { "Ref" : "InstanceResouceName" } # The instance logic
name
      }
    }
```

### 4.4.2.3 Alarm Resource

It defines an event that trigger the restart policy.

How to define an Alarm Resource:

```
   "FailureAlarmName": {
     "Type": "AWS::CloudWatch::Alarm",
     "Properties": {
        "AlarmDescription": "Alarm Description",
        "MetricName": "ServiceFailure", # Collect signals from services within
the instance in order to check the services health
        "Namespace": "system/linux",
        "Statistic": "SampleCount", # Count signals
        "Period": "300",   # Period time in seconds
        "EvaluationPeriods": "1", # How many failing periods will trigger the
action
        "Threshold": "2",    # How many signals are needed in a Period to fail
it.
        "AlarmActions": [ { "Ref": "WebServerRestartPolicy" } ], # Policy to
trigger on Alarm failure
        "ComparisonOperator": "GreaterThanThreshold"
      }
    }
```

### 4.4.2.4 Instance Configuration for High Availability

You will need to configure CloudFormation services within the instance in order to provides AWS credentials and set up the scripts that will check the health of the service. You can use the AWS::CloudFormation::Init metadata to set up scripts and configuration file.

How to configure the instance for High Availability:

```
    "InstanceWithHA": {
      "Type": "AWS::EC2::Instance",
      "Metadata" : {
        "AWS::CloudFormation::Init" : {
          "config" : {
            "files" : {
              "/etc/cfn/cfn-credentials" : {  # The file where aws credentials
are stored
                "content" : { "Fn::Join" : ["", [
                  "AWSAccessKeyId=", { "Ref" : "KeyName" }, "\n", # The logic
name of AWS::IAM::AccessKey resource defined before
                  "AWSSecretKey=", {"Fn::GetAtt": ["KeyName",      # The Secret
of the Access Key
                                    "SecretAccessKey"]}, "\n"
                ]]},
                "mode"    : "000400",
                "owner"   : "root",
                "group"   : "root"
              },



              "/etc/cfn/cfn-hup.conf" : {         # Configuration of the daemon
that will do periodically checks
                "content" : { "Fn::Join" : ["", [
                  "[main]\n",
                  "stack=", { "Ref" : "AWS::StackName" }, "\n",
                  "credential-file=/etc/cfn/cfn-credentials\n",
                  "region=", { "Ref" : "AWS::Region" }, "\n",
                  "interval=",  {  "Ref"  :  "HupPollInterval"  },  "\n"    #  Poll
Interval in seconds
                ]]},
                "mode"    : "000400",
                "owner"   : "root",
                "group"   : "root"
              },

              "/etc/cfn/notify-on-service-restarted"  :  {          #  Action  on
service Failure
                "content" : { "Fn::Join" : ["", [
                "#!/bin/sh\n",
                "/opt/aws/bin/cfn-push-stats  --watch  ",              #  Send  a
failure signal to the related alarm defined before
                { "Ref" : "FailureAlarmName" },
                " --service-failure\n"
                ]]},
                "mode"    : "000700",
                "owner"   : "root",
                "group"   : "root"
              },



              "/tmp/cfn-hup-crontab.txt" : {         # Crontab file that launch
the cfn-hup script
                "content" : { "Fn::Join" : ["", [
                "MAIL=\"\"\n",
                "\n",
                "* * * * * /opt/aws/bin/cfn-hup -f\n"
                ]]},
                "mode"    : "000600",
                "owner"   : "root",
                "group"   : "root"
              },


              "/etc/cfn/hooks.conf"  :  {                 #  Set  hook  on  service
failure
                "content": { "Fn::Join" : ["", [
                  "[cfn-service-restarted]\n",
                  "triggers=service.restarted\n",      # When trigger the hook
                  "path=Resources.InstanceName.Metadata\n",      #  Resouce  to
monitor
                  "action=/etc/cfn/notify-on-service-restarted\n",     #  Script
to trigger
                  "runas=root\n"
                ]]},
                "mode"    : "000400",
                "owner"   : "root",
                "group"   : "root"
              }
```

Then you will need some custom UserData to install the crontab file:

```
       "UserData"       : { "Fn::Base64" : { "Fn::Join" : ["", [
          "#!/bin/bash -v\n",
          "# install cfn-hup crontab\n",
          "crontab /tmp/cfn-hup-crontab.txt\n",
        ]]}}
      }
```

### 4.4.3 Wait Condition (CloudFormation)

Wait Condition resources are used to triage resources deployment, sending signals when instance are initialized successfully. In order to use the wait condition we need the WaitConditionHandle, the WaitCondition resource, the "Depends on" directive whithin resources and some specific configuration within the instances.

#### 4.4.3.1 WaitConditionHandle Resource

WaitConditionHandle manages signals that instances can send when sucessfully execute some tasks.

How to define WaitConditionHandle Resource:

```
    "WaitHandleName" : {
      "Type" : "AWS::CloudFormation::WaitConditionHandle"
    }
```

#### 4.4.3.2 WaitCondition Resource

WaitCondition manages the timeout property that the resource will wait before to ratify the stack failure.

How to define WaitCondition Resource:

```
    "WaitConditionName" : {
      "Type" : "AWS::CloudFormation::WaitCondition",
      "Properties" : {
        "Handle"  :  {"Ref"  :  "WaitHandleName"},    #  The  Reference  to  the
WaitHandle.
        "Timeout" : "600"         # Maxiumum waiting time in seconds
      }
    },
```

#### 4.4.3.3 "Depends On" directive

It is the directive within the resource definition that allow its deployment only after the specified WaitCondition finish successfully

How to define the "Depends On" directive:

```
     "DependsOn" : "WaitConditionName"
```

Then we need some custom UserData in order to send signal when the instance starts its initialization and when it finish its initialization successfully.

```
        "UserData"       : { "Fn::Base64" : { "Fn::Join" : ["", [
          "#!/bin/bash -v\n",

          "# Helper function\n",
          "function error_exit\n",
          "{\n",
          "     /opt/aws/bin/cfn-signal   -e   1   -r   \"$1\"   '",   {   "Ref"   :
"NFSWaitHandle" }, "'\n",  # Signal to send in case of failure
          "  exit 1\n",
          "}\n",

          "/opt/aws/bin/cfn-init -s ", { "Ref" : "AWS::StackName" },          #
Command to start the instance initialization
          " -r InstanceName ",
          " --region ", { "Ref" : "AWS::Region" },
          " || error_exit 'Failed to run cfn-init'\n",

          "# Success Signal\n",
          "/opt/aws/bin/cfn-signal  -e  0  -r  \"Instance  setup  complete\"  '",
# Send a success signal to the WaitHandle
          { "Ref" : "WaitHandleName" }, "'\n"
        ]]}}
      }
```

### 4.4.4 Auto Scaling Group

Auto Scaling Group is the heat feature that allows multiple deployments of the same instance whitin a "Scaling Group". User can defines policy to scale up and scale down instances according to the workload. In order to create an Auto Scaling Group user need to define AutoScalingGroup, ScalingPolicy, Alarm and LaunchConfiguration resources.

#### 4.4.4.1 AutoScalingGroup Resource

It's the resource that describe a scaling group whitin one or more Availability Zones, it defines the starting number of instances and a maximum number of instances allowed. Each instance will be deployed using the configuration described by LaunchConfiguration resource. We need some custom userdata whitin the instances for sending metrics to heat services.

How to define an AutoScalingGroup Resource:

```
   "GroupName" : {      # It's the name Scaling Group name
      "Type" : "AWS::AutoScaling::AutoScalingGroup",
      "Properties" : {
        "AvailabilityZones" : { "Fn::GetAZs" : ""},    # This example allows it
in all AZ
        "LaunchConfigurationName" : { "Ref" : "LaunchConfig" },    # It's the
instance configuration described by LaunchConfiguration resource
        "MinSize" : "1",      # Starting number of instances within the ASG
        "MaxSize" : "2"         # Maximum number of instances allowed.
      }
    }
```

#### 4.4.4.2 ScalingPolicy Resource

It's the resource that define how the Scaling Group will scale up or down.

How to define a ScailngPolicy Resource:

```
    "ScalePolicyName" : {         # It's the Scaling Policy Name
      "Type" : "AWS::AutoScaling::ScalingPolicy",
      "Properties" : {
        "AdjustmentType"  :  "ChangeInCapacity",     #  Defines  a  change  of  the
number of instances within the scaling group
        "AutoScalingGroupName" : { "Ref" : "GroupName" },  # Define which group
this policy refer to.
        "Cooldown" : "60",              # Define how many seconds heat have to
wait until triggers this policy again
        "ScalingAdjustment" : "1"          # Define how many instances will be
added  (using  a  positive  number  for  scaling  up)  or  will  be  deleted  (using  a
negative number for scaling down)
      }
    },
```

#### 4.4.4.3 Alarm Resource

Defines an event that trigger the scaling policy.

How to define an Alarm Resource:

```
    "AlarmName": {      # It's the alarm name
     "Type": "AWS::CloudWatch::Alarm",
     "Properties": {
        "AlarmDescription": "Alarm Description",
        "MetricName":  "MemoryUtilization",  #  Define  which  metric  check  (it
could be MemoryUtilization, CPUUtilization, etc..)
        "Namespace": "system/linux",
        "Statistic":  "Average",  #  It  will  monitor  the  average  of  collected
metrics
        "Period": "60",    # Period time in seconds, the alarm will trigger if
the threshold  is reached for at least PERIOD seconds
        "EvaluationPeriods": "1", # How many periods will trigger the action
        "Threshold": "80",    # Threshold value of the monitored metric
        "AlarmActions": [ { "Ref": "ScalingPolicy" } ], # Scaling Policy to be
triggered
        "ComparisonOperator":     "GreaterThanThreshold"               #     Use
GreaterThanThreshold if you want to trigger the policy when the metric is above
the thrashold or LessThenThreshold if you want to trigger the policy when the
metric is below the threshold
      }
    }
```

#### 4.4.4.4 LaunchConfiguration Resource and UserData

Defines the configuration of the instances deployed within the Scaling Group, it has exact the same syntax, options, and section of Instance Resource, but it needs some particular UserData in order to periodically send collected metrics to heat external services.

How to define LaunchConfiguration resource:

```
   "LaunchConfigName" : {
      "Type" : "AWS::AutoScaling::LaunchConfiguration",
      "Metadata" : {
        "AWS::CloudFormation::Init" : {
          "config" : {
            "files" : {
              "/etc/cfn/cfn-credentials"  :  {            #  The  aws  credentials
defined in the tempalte
                "content" : { "Fn::Join" : ["", [
                  "AWSAccessKeyId=", { "Ref" : "ServerKeys" }, "\n",
                  "AWSSecretKey=", {"Fn::GetAtt": ["ServerKeys",
                                    "SecretAccessKey"]}, "\n"
                ]]},
                "mode"    : "000400",
                "owner"   : "root",
                "group"   : "root"
              },
             "/tmp/stats-crontab.txt" : {          # The crontab that will send
metrics to heat external services
                "content" : { "Fn::Join" : ["", [
                "MAIL=\"\"\n",
                "\n",
                "*  *  *  *  * /opt/aws/bin/cfn-push-stats --watch ",   # Send the
metric specified by the parameter (in this case --mem-util) to heat external
service
                { "Ref" : "UserAlarm1" }, " --mem-util\n",
                "* * * * * /opt/aws/bin/cfn-push-stats --watch ",
                { "Ref" : "UserAlarm2" }, " --mem-util\n"
                ]]},
                "mode"    : "000600",
                "owner"   : "root",
                "group"   : "root"
              }
```

How to define UserData:

```
      "UserData"       : { "Fn::Base64" : { "Fn::Join" : ["", [
          "#!/bin/bash -v\n",
          "/opt/aws/bin/cfn-init  -s  ",  {  "Ref"  :  "AWS::StackName"  },    #
Initialize the instance
          " -r LaunchConfig ",
          " --region ", { "Ref" : "AWS::Region" }, "\n",
          "# install crontab\n",
          "crontab /tmp/stats-crontab.txt\n"      # install the crontab file
}
```

### 4.4.4.5 LoadBalancer within an Auto Scaling Group

The resource syntax is exactly the same as the normal LoadBalancer, but without the "Instances" property, instant of it you will need to add "LoadBalancerNames" property in the related AutoScalingGroup resource definition.
So the LoadBalancer definition will be like this:

```
    "LoadBalancerName" : {
      "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
      "Properties" : {
        "AvailabilityZones" : { "Fn::GetAZs" : "" },
        "Listeners" : [ {
          "LoadBalancerPort" : "80",    # Specifies the port that Load Balancer
will listen on
          "InstancePort" : "80",   # Specifies the port to balance
          "Protocol" : "HTTP"      # Specifies the protocol to balance
        }],
        "HealthCheck" : {          # Specifies some parameters to monitor the
servers' health
          "Target" : "HTTP:80/",
          "HealthyThreshold" : "3",
          "UnhealthyThreshold" : "5",
          "Interval" : "30",
          "Timeout" : "5"

      }
    }
```

and the AutoScalingGroup resource definition will be like this:

```
    "ServerGroup" : {
      "Type" : "AWS::AutoScaling::AutoScalingGroup",
      "Properties" : {
        "AvailabilityZones" : { "Fn::GetAZs" : ""},
        "LaunchConfigurationName" : { "Ref" : "ServerLaunchConfig" },
        "MinSize" : "2",
        "MaxSize" : "8",
        "LoadBalancerNames" : [ { "Ref" : "LoadBalancerName" } ]   # The
LoadBalancer logic name to automatically configure according to the
AutoScalingGroup
      }
    },
```

# 5 Heat Installation

OpenStack Heat has been installed using RDO repository on Scientific Linux Cern 6.4, while the OpenStack testing environment has been installed using PackStack.

## 5.1 RDO repository

RDO is a Red Hat community service which provides a repository for users of OpenStack on Red Hat based platforms.

### 5.1.1 PackStack

PackStack is a tool for set up a quick and full installation of OpenStack usung puppet modules. Using PackStack you can install the infrastructure on several servers or deploy an all-in-one installation.

## 5.2 Setting up the environment

The OpenStack testing environment has been set up with a PackStack all-in-one installation, while Heat has been installed on a dedicated machine using RDO repository.

### 5.2.1 OpenStack installation using PackStack

Add the rdo grizzly repository

```
# yum install http://rdo.fedorapeople.org/openstack/openstack-grizzly/rdo-release-grizzly.rpm
```

Install packstack installer

```
# yum install openstack-packstack
```

If you don't have a password-enabled root account, add the local public key to the authorized_key file

```
# cd /root/.ssh; cat id_rsa.pub >> authorized_keys
```

Install OpenStack without Neutron

```
# packstack --allinone --os-quantum-install=n
```

If you want to test you OpenStack installation, download this test image and add it to glance

```
# wget -c https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.img -O cirros.img
# glance image-create --name=cirros-0.3.0-x86_64 --disk-format=qcow2 --container-format=bare < cirros.img
```

Now connect to your dashboard at http://YOUR_IP/dashboard using the credentials stored in /root/keystonerc_admin and launch your VM!

### 5.2.2 Heat Installation

Add rdo grizzly repository if you don't have it yet

```
# yum install http://rdo.fedorapeople.org/openstack/openstack-grizzly/rdo-
release-grizzly.rpm
```

Now you can install all heat packages and dependencies (it will provides mainly the services heat-api, heat-api-cfn, heat-api-cloudwatch and heat-engine)

```
# yum install openstack-heat-* MySQL-python
```

Heat from rdo requires a local mysql server, if there isn't any mysql server installed, this script is going to install, create and poupulate the database

$MYSQL-ROOT-PASSWORD: will be the mysql root password, if you have already a mysql server installed, enter the right password

$HEAT-DB-PASSWORD: will be the database password for heat user

```
# heat-db-setup rpm -r $MYSQL-ROOT-PASSWORD -p $HEAT-DB-PASSWORD
```

You have to set auth_encryption_key option in /etc/heat/heat-engine.conf with a random generated key:

```
# sed -i "s/%ENCRYPTION_KEY%/`hexdump -n 16 -v -e '/1 "%02x"' /dev/random`/"
/etc/heat/heat-engine.conf
```

### 5.2.3 KeyStone configuration

Now you have to add the user, services and endpoints for heat on the Identity Service

Add the heat user
$HEAT_USER_PASSWORD_OF_CHOICE: a password for heat user on the keystone
$SERVICES_TENANT_NAME: your services tenant on the keystone (you can use the command 'keystone tenant-list' to get it)

```
# keystone user-create --name heat --pass $HEAT_USER_PASSWORD_OF_CHOICE
#    keystone    user-role-add    --user    heat    --role    admin    --tenant
$SERVICES_TENANT_NAME
```

Add the services

```
# keystone service-create --name heat --type orchestration
# keystone service-create --name heat-cfn --type cloudformation
```

Add endpoints
$HEAT_CFN_SERVICE_ID and $HEAT_SERVICE_ID are the service ids of the services previously created. You can get them by running the 'keystone service-list' command.
$HEAT_CFN_HOSTNAME and $HEAT_HOSTNAME are heat host hostname or IP address

```
# keystone endpoint-create --region RegionOne --service-id $HEAT_CFN_SERVICE_ID
--publicurl            "http://$HEAT_CFN_HOSTNAME:8000/v1"            --adminurl
"http://$HEAT_CFN_HOSTNAME:8000/v1"                                --internalurl
"http://$HEAT_CFN_HOSTNAME:8000/v1"
# keystone endpoint-create --region RegionOne --service-id $HEAT_SERVICE_ID --
publicurl        "http://$HEAT_HOSTNAME:8004/v1/%(tenant_id)s"        --adminurl
"http://$HEAT_HOSTNAME:8004/v1/%(tenant_id)s"                      --internalurl
"http://$HEAT_HOSTNAME:8004/v1/%(tenant_id)s"
```

### 5.2.4 Heat configuration

Update the paste files /etc/heat/heat-api{,-cfn,-cloudwatch}-paste.ini with the credentials just created

```
admin_tenant_name = $SERVICES_TENANT_NAME
admin_user = heat
admin_password = $HEAT_USER_PASSWORD
```

Make sure that the following variables (in all heat-*-paste.ini files) are pointing to keystone host

```
service_host = $KEYSTONE_HOSTNAME
auth_host = $KEYSTONE_HOSTNAME
auth_uri = http://$KEYSTONE_HOSTNAME:35357/v2.0
keystone_ec2_uri = http://$KEYSTONE_HOSTNAME:5000/v2.0/ec2tokens
```

Then change in /etc/heat/heat-engine.conf the value 127.0.0.1 with the actual hostname, because these urls will be passed over to the VMs.

```
heat_metadata_server_url = http://$HEAT_CFN_HOSTNAME:8000
heat_waitcondition_server_url = http://$HEAT_CFN_HOSTNAME:8000/v1/waitcondition
heat_watch_server_url = http://$HEAT_CLOUDWATCH_HOSTNAME:8003
```

Now create the role used by heat to receive the progress data in according to heat_stack_user_role option in /etc/heat/heat-engine.conf (default is heat_stack_user)

```
keystone role-create --name heat_stack_user
```

Start all heat services

# cd /etc/init.d && for s in $(ls openstack-heat-*); do service $s start; done

Enable heat services autostart

```
# cd /etc/init.d && for s in $(ls openstack-heat-*); do chkconfig $s on; done
```

## 5.3 Testing Heat

Add a pre-built test image to glance (Fedora 17 32bit and 64bit)

```
# glance image-create --name F17-i386-cfntools --disk-format qcow2 --container-format bare --is-public True --copy-from http://fedorapeople.org/groups/heat/prebuilt-jeos-images/F17-i386-cfntools.qcow2
# glance image-create --name F17-x86_64-cfntools --disk-format qcow2 --container-format bare --is-public True --copy-from http://fedorapeople.org/groups/heat/prebuilt-jeos-images/F17-x86_64-cfntools.qcow2
```

Download a test template (Wordpress Single server instance)

```
# wget https://raw.github.com/openstack/heat-templates/master/cfn/WordPress_Single_ Instance.template
```

Start the test stack (you will need a valid keypair in OpenStack)

```
#          heat-cfn          create          $STACK-NAME          --template-
file=WordPress_Single_Instance.template                                    --
parameters="DBUsername=wp;DBPassword=wpass;KeyName=test;InstanceType=m1.small;L
inuxDistribution=F18"
```

Check if the stack is running

```
# heat-cfn list
```

## 5.4 Using Heat

A quick reference for the main heat commands.

#### 5.4.1.1 Install heat client

First of all is useful to install the heat client, it provides better outputs then heat-cfn client and it can be used from any host on the network.

```
# yum install python-heatclient python-heatclient-doc
```

**Start a stack**

```
# heat stack-create -f templatename.template -P
"KeyName=mykey;InstanceType=m1.tiny;LinuxDistribution=F17"  stack-name
```

**List stacks**

```
# heat stack-list
```

**List stack's event list**

```
# heat event-list stack-name
```

**Describe a stack**

```
# heat describe stack-name
```

**Delete a stack**

```
# heat stack-delete stack-name
```

# 6 Use case

This use case template deploys some webservers within an auto scaling group and defines policies that scale up and scale down the number of webservers according to their memory usage. All the webservers serve pages from a common nfs mountpoint, hosted on a dedicated nfs server and clients can access the webservers through a load balacer that distribuite the load within the auto scaling group.

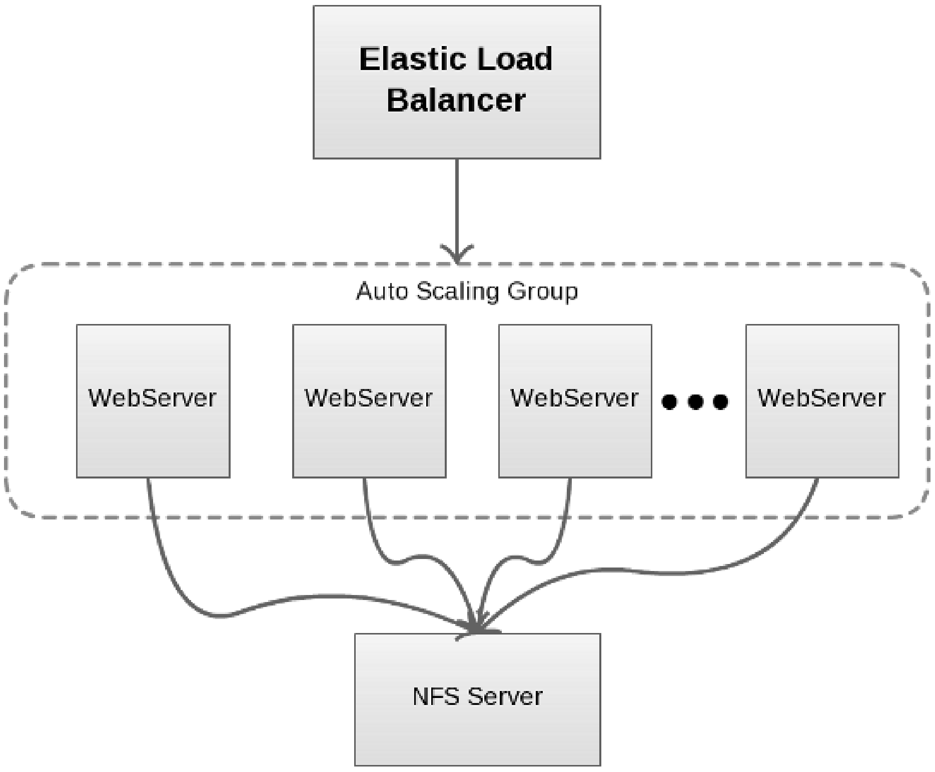


*Figure 6. Use case*

## 6.1 Stack workflow

The stack will be deployed creating resources in the following order:

•WebServerLoadBalancer [!]

•CfnUser

•NFSServer [!]

•NFSWaitHandle

•WebServerKeys

•WebServerLaunchConfig

•WebServerLoadBalancer

Then it waits that the NFS server is running successfully, so it can continue the stack deployment with:

•WebServerGroup [!]

•MEMAlarmHigh

•MEMAlarmLow

•WebServerScaleDownPolicy

•WebServerScaleUpPolicy

[!] Resources that create VMs

## 6.2 Use case main resource

### 6.2.1 Auto Scaling Group

It's the resource that deploys the webservers in the auto scaling group. It waits that the NFSWaitCondition resource is ready.

```
  "WebServerGroup" : {
      "Type" : "AWS::AutoScaling::AutoScalingGroup",
      "DependsOn" : "NFSWaitCondition",
      "Properties" : {
        "AvailabilityZones" : { "Fn::GetAZs" : ""},
        "LaunchConfigurationName" : { "Ref" : "WebServerLaunchConfig" },
        "MinSize" : "2",
        "MaxSize" : "8",
        "LoadBalancerNames" : [ { "Ref" : "WebServerLoadBalancer" } ]
      }
```

### 6.2.2 WebServer Launch Configuration

It's the resource that describe the configuration of each WebServer that the stack will deploy within the Auto Scaling Group.

```
    "WebServerLaunchConfig" : {
      "Type" : "AWS::AutoScaling::LaunchConfiguration",
      "Metadata" : {
        "AWS::CloudFormation::Init" : {
          "config" : {
            "files" : {
              "/etc/cfn/cfn-credentials" : {
                "content" : { "Fn::Join" : ["", [
                  "AWSAccessKeyId=", { "Ref" : "WebServerKeys" }, "\n",
                  "AWSSecretKey=", {"Fn::GetAtt": ["WebServerKeys",
                                    "SecretAccessKey"]}, "\n"
                ]]},
                "mode"    : "000400",
                "owner"   : "root",
                "group"   : "root"
              },
              "/tmp/stats-crontab.txt" : {
                "content" : { "Fn::Join" : ["", [
                "MAIL=\"\"\n",
                "\n",
                "* * * * * /opt/aws/bin/cfn-push-stats --watch ",
                { "Ref" : "MEMAlarmHigh" }, " --mem-util\n",
                "* * * * * /opt/aws/bin/cfn-push-stats --watch ",
                { "Ref" : "MEMAlarmLow" }, " --mem-util\n"
                ]]},
                "mode"    : "000600",
                "owner"   : "root",
                "group"   : "root"
              },

              "/tmp/index.html" : {
                "content" : { "Fn::Join" : ["", [
                  "<html>\n",
                  "<body>\n",
                  "Nfs server offline\n",
                  "</body>\n",
                  "</html>\n"
                ]]},
                "mode"    : "000400",
                "owner"   : "root",
                "group"   : "root"
              },


              "/tmp/nfs-site.conf" : {
                "content" : { "Fn::Join" : ["", [
                  "<VirtualHost *:80>\n",
                  "    ServerAlias /\n",
                  "    ServerAdmin root@localhost\n",
                  "    DocumentRoot /var/www/html/nfs\n",
                  "    DirectoryIndex index.html\n",
                  "</VirtualHost>\n"
                ]]},
                "mode"    : "000400",
                "owner"   : "root",
                "group"   : "root"
              }

            },
            "packages" : {
              "yum" : {
                "httpd"        : [],
                "nfs-utils"        : []
              }
            },
            "services" : {
              "sysvinit" : {
                "httpd"    : { "enabled" : "true", "ensureRunning" : "true" },
                "crond"    : { "enabled" : "true", "ensureRunning" : "true" },
                "iptables"      :  {  "enabled"  :  "false",  "ensureRunning"  :
"false" }
              }
            }
          }
        }
      },
      "Properties": {
        "ImageId"  :  {  "Fn::FindInMap"  :  [  "DistroArch2AMI",  {  "Ref"  :
"LinuxDistribution" },
                          { "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref"
: "InstanceType" }, "Arch" ] } ] },
        "InstanceType"   : { "Ref" : "InstanceType" },
        "KeyName"        : { "Ref" : "KeyName" },
        "UserData"       : { "Fn::Base64" : { "Fn::Join" : ["", [
          "#!/bin/bash -v\n",
          "/opt/aws/bin/cfn-init -s ", { "Ref" : "AWS::StackName" },
          " -r WebServerLaunchConfig ",
          " --region ", { "Ref" : "AWS::Region" }, "\n",

          "# Configure WebServer\n",
          "mkdir -p /var/www/html/nfs\n",
          "cp /tmp/index.html /var/www/html/nfs/\n",
          "cp /tmp/nfs-site.conf /etc/httpd/conf.d/\n",

          "# Mount nfs file system\n",
          "echo \"", { "Fn::GetAtt" : [ "NFSServer", "PublicIp" ] }, ":/nfs/www
/var/www/html/nfs/      nfs     ro              0 0\" >> /etc/fstab\n",
          "mount /var/www/html/nfs/\n",
          "# Restart httpd \n",
          "service httpd restart\n",

          "# install crontab\n",
          "crontab /tmp/stats-crontab.txt\n"
        ]]}}
      }
    }
  },
```

### 6.2.3 Scale up and down policies

It describes polices that will scale up and down the auto scaling group.

```
   "WebServerScaleUpPolicy" : {
      "Type" : "AWS::AutoScaling::ScalingPolicy",
      "Properties" : {
        "AdjustmentType" : "ChangeInCapacity",
        "AutoScalingGroupName" : { "Ref" : "WebServerGroup" },
        "Cooldown" : "60",
        "ScalingAdjustment" : "1"
      }
    },

    "WebServerScaleDownPolicy" : {
      "Type" : "AWS::AutoScaling::ScalingPolicy",
      "Properties" : {
        "AdjustmentType" : "ChangeInCapacity",
        "AutoScalingGroupName" : { "Ref" : "WebServerGroup" },
        "Cooldown" : "60",
        "ScalingAdjustment" : "-1"
      }
    },
```

### 6.2.4 Alarms

It describes the resources that will trigger the scale up and down policies.

```
    "MEMAlarmHigh": {
     "Type": "AWS::CloudWatch::Alarm",
     "Properties": {
        "AlarmDescription": "Scale-up if MEM > 50% for 1 minute",
        "MetricName": "MemoryUtilization",
        "Namespace": "system/linux",
        "Statistic": "Average",
        "Period": "60",
        "EvaluationPeriods": "1",
        "Threshold": "75",
        "AlarmActions": [ { "Ref": "WebServerScaleUpPolicy" } ],
        "Dimensions": [
          {
            "Name": "AutoScalingGroupName",
           "Value": { "Ref": "WebServerGroup" }
          }
        ],
        "ComparisonOperator": "GreaterThanThreshold"
      }
    },

    "MEMAlarmLow": {
     "Type": "AWS::CloudWatch::Alarm",
     "Properties": {
        "AlarmDescription": "Scale-down if MEM < 15% for 1 minute",
        "MetricName": "MemoryUtilization",
        "Namespace": "system/linux",
        "Statistic": "Average",
        "Period": "60",
        "EvaluationPeriods": "1",
        "Threshold": "50",
        "AlarmActions": [ { "Ref": "WebServerScaleDownPolicy" } ],
        "Dimensions": [
          {
            "Name": "AutoScalingGroupName",
            "Value": { "Ref": "WebServerGroup" }
          }
        ],
        "ComparisonOperator": "LessThanThreshold"
      }
    },
```

### 6.2.5 NFS Server Instance Resource

This is the resource that describes the NFS Server Instance.

```
    "NFSServer": {
      "Type": "AWS::EC2::Instance",
      "Metadata" : {
        "AWS::CloudFormation::Init" : {
          "config" : {
            "packages" : {
              "yum" : {
                "nfs-utils"        : []
              }
            },
            "services" : {
              "sysvinit" : {
                "nfs"    : { "enabled" : "true", "ensureRunning" : "true" },
                "rpcbind"    : { "enabled" : "true", "ensureRunning" : "true" },
                "nfslock"    : { "enabled" : "true", "ensureRunning" : "true" },
                "iptables"    : { "enabled" : "false", "ensureRunning" : "false"
}
              }
            },
            "files" : {
              "/etc/cfn/cfn-credentials" : {
                "content" : { "Fn::Join" : ["", [
                  "AWSAccessKeyId=", { "Ref" : "WebServerKeys" }, "\n",
                  "AWSSecretKey=", {"Fn::GetAtt": ["WebServerKeys",
                                    "SecretAccessKey"]}, "\n"
                ]]},
                "mode"    : "000400",
                "owner"   : "root",
                "group"   : "root"
              },

              "/etc/cfn/cfn-hup.conf" : {
                "content" : { "Fn::Join" : ["", [
                  "[main]\n",
                  "stack=", { "Ref" : "AWS::StackName" }, "\n",
                  "credential-file=/etc/cfn/cfn-credentials\n",
                  "region=", { "Ref" : "AWS::Region" }, "\n",
                  "interval=", { "Ref" : "HupPollInterval" }, "\n"
                ]]},
                "mode"    : "000400",
                "owner"   : "root",
                "group"   : "root"
              },

              "/tmp/cfn-hup-crontab.txt" : {
                "content" : { "Fn::Join" : ["", [
                "MAIL=\"\"\n",
                "\n",
                "* * * * * /opt/aws/bin/cfn-hup -f\n"
                ]]},
                "mode"    : "000600",
                "owner"   : "root",
                "group"   : "root"
              },

              "/etc/exports" : {
                "content": { "Fn::Join" : ["", [
                  "/nfs/www
192.168.32.0/255.255.255.0(ro,sync,wdelay,hide,nocrossmnt,secure,root_squash,no
_all_squash,no_subtree_check,secure_locks,acl,anonuid=65534,anongid=65534)\n"
                ]]},
                "mode"    : "000644",
                "owner"   : "root",
                "group"   : "root"
              },


              "/tmp/index.html" : {
                "content": { "Fn::Join" : ["", [
                  "<html>\n",
                  "<body>\n",
                  "nfs site!\n",
                  "</body>\n",
                  "</html>\n"
                ]]},
                "mode"    : "000444",
                "owner"   : "root",
                "group"   : "root"
              }


            }

          }
        }
      },
      "Properties": {
        "ImageId"  :  {  "Fn::FindInMap"  :  [  "DistroArch2AMI",  {  "Ref"  :
"LinuxDistribution" },
                          { "Fn::FindInMap" : [ "AWSInstanceType2Arch", { "Ref"
: "InstanceType" }, "Arch" ] } ] },
        "InstanceType"   : { "Ref" : "InstanceType" },
        "KeyName"        : { "Ref" : "KeyName" },
        "UserData"       : { "Fn::Base64" : { "Fn::Join" : ["", [
          "#!/bin/bash -v\n",

          "# Helper function\n",
          "function error_exit\n",
          "{\n",
          "     /opt/aws/bin/cfn-signal   -e   1   -r   \"$1\"   '",   {   "Ref"   :
"NFSWaitHandle" }, "'\n",
          "  exit 1\n",
          "}\n",
          "/opt/aws/bin/cfn-init -s ", { "Ref" : "AWS::StackName" },
          " -r NFSServer ",
          " --region ", { "Ref" : "AWS::Region" },
          " || error_exit 'Failed to run cfn-init'\n",

          "# Setup NFS configuration\n",
          "#iptables -D INPUT -j REJECT --reject-with icmp-host-prohibited\n",
          "mkdir -p /nfs/www\n",
          "cp /tmp/index.html /nfs/www/\n",
          "#exportfs -o ro 192.168.32.0/255.255.255.0:/nfs/www\n",
          "exportfs -a\n",
          "# install cfn-hup crontab\n",
          "crontab /tmp/cfn-hup-crontab.txt\n",

          "# All is well so signal success\n",
          "/opt/aws/bin/cfn-signal -e 0 -r \"NFS setup complete\" '",
          { "Ref" : "NFSWaitHandle" }, "'\n"
        ]]}}
      }
    },
```

### 6.2.6 Load Balancer Resource

This is the resource that defines the Load Balancer.

```
   "WebServerLoadBalancer" : {
      "Type" : "AWS::ElasticLoadBalancing::LoadBalancer",
      "Properties" : {
        "AvailabilityZones" : { "Fn::GetAZs" : "" },
        "Listeners" : [ {
          "LoadBalancerPort" : "80",
          "InstancePort" : "80",
          "Protocol" : "HTTP"
        } ],
        "HealthCheck" : {
          "Target" : "HTTP:80/",
          "HealthyThreshold" : "3",
          "UnhealthyThreshold" : "5",
          "Interval" : "30",
          "Timeout" : "5"
        }
      }
    },
```

### 6.2.7 Stack output

This is stack output, it will return the Load Balancer's url that users can use to access to the webservers.

```
 "Outputs" : {
    "PublicIp": {
      "Description": "LoadBalancer IP",
      "Value"  :   {  "Fn::Join"  :  [  "",  [  "http://",  {  "Fn::GetAtt"  :  [
"WebServerLoadBalancer", "DNSName" ]}, "/"]] }
    }
  }
}
```