

Using Heat and Ceilometer for providing Autoscaling in OpenStack

ABSTRACT

Cloud computing is being widely used for providing access to a shared pool of hardware as well as software resources because of its clear advantages like on-demand resource provisioning, elasticity, metered service. For a typical application like eCommerce on cloud, resource usage is highly variable with peak traffic on weekends or during sale period and low during night. For traditional deployment the user would make provision for highest required resources but this scheme does not ensure optimal resource usage and is also not cost effective for user. The characteristic feature of cloud computing elasticity comes to the rescue providing the ability to cloud to automatically scale up or down commensurate with demand. This paper discusses how OpenStack, a widely popular open source IaaS cloud provides auto scaling using heat and ceilometer.

KEYWORDS

Cloud Computing, OpenStack, Heat, High Availability, Ceilometer, Autoscaling.

1. INTRODUCTION

Cloud computing is the most recent trend in computing. It provides on-demand network access to shared pool of scalable computing resources (e.g., networks, servers, storage, applications, and services) on pay per use basis at anytime from anywhere. Users do not need to worry about technical/physical management and maintenance issue of the original resources [1]. Resources are provided as a set of services in a virtualized environment and accessed through a public network connection generally Internet. There are three kinds of service models:

- **IaaS (Infrastructure as a Service)** provides hardware resources like server space, network bandwidth, IP addresses, and storage on the cloud. The resources provided by cloud management company may be distributed across various data centres. The client uses these virtualized hardware resources for developing their own IT platforms.
- **PaaS (Platform as a Service)** helps users to develop and deploy applications on cloud platform. Examples include programming platforms for Java

and .NET which help users in developing programs without having to download them at their end.

- **SaaS (Software as a Service)** provides users access to application software and databases such as web-based email [2].

Clouds may be deployed as public, private, or hybrid resource. In a public cloud, services offered are provided to the general public by the third party provider. Private clouds are either owned or leased by the single organizations and the services are offered to people within an organization. Such a cloud exists either on-premises or off-premises of cloud provider. A hybrid cloud is a composition of both public and private clouds.

Using cloud computing for providing resources has several benefits:

- **On-demand self-service:** Cloud computing enables provision of resources to the clients on demand with minimal service provider interaction.
- **Broad network access:** Computing resources are available for access from a wide range of devices smartphones, tablets, PCs and workstations. These devices are also accessible from a wide range of locations that offer online access.
- **Resource pooling:** It allows cloud providers to pool large scale IT resources to serve multiple cloud consumers using multi-tenancy model. These resources are provided virtually or physically to users whenever demanded. Examples of resources include storage, processing, memory and network bandwidth.
- **Measured service:** Cloud provider measures the services provided to the client by keeping track of services consumed and charging them on pay per use basis.
- **Rapid elasticity:** Computing resources are allocated and released in proportion with demand i.e. the cloud can scale up or down automatically depending on usage. This is known as autoscaling. It also avoids the problem of over-utilization and under-utilization of in-house resources and is cost effective for the user [3]. Openstack provides autoscaling using Heat based on alarms raised by Ceilometer.

¹ Sri Aurobindo College, Computer Science Department, University of Delhi

² Dyal Singh College, Computer Science Department, University of Delhi

³ Department of Computer Science, University of Delhi

Email: ¹gpratibha11@gmail.com, ²sheetaltaneja1@gmail.com., ³aparna.datt@gmail.com

OpenStack, Nimbus, Eucalyptus and OpenNebula are some of the open source IaaS cloud softwares [4]. OpenStack is widely popular and has been supported by many companies including AT&T, Cisco, Dell, IBM, Oracle, VMware. It follows a six month release cycle [5]. In this paper, we will discuss how OpenStack provides high availability using Heat - the orchestration engine, based on alarms raised by Ceilometer, the telemetry.

Rest of the paper is organized as follows: section 2 discusses related work; section 3 introduces OpenStack, its architecture and components; section 4 discusses High availability, Heat, its architecture and How Heat and Ceilometer provide autoscaling; and finally section 5 concludes the paper.

2. RELATED WORK

High availability in cloud is an important characteristic of cloud that is required to cater to the ever-changing needs of the end users and also for optimal resource utilization. Jeong et. al. [26] in their paper present the idea of providing high availability by using the idea of grid infrastructure. Instead of deploying new servers, existing servers with available resources are used for resource provisioning.

Benz et. al. [9] present a dependability modeling framework for testing high availability capabilities of cloud architectures, specifically OpenStack.

An algorithm for achieving high availability and fault-tolerance while reducing the application cost and maximizing the resource load has been proposed by Frincu et. al. [27].

3. OPENSTACK

3.1 INTRODUCTION

OpenStack is a free and open source cloud computing software platform. The software controls large pools of compute, storage and networking resources throughout a data center, all managed through a web-based dashboard called *Horizon* or via the OpenStack API. Users primarily deploy it as an IaaS (Infrastructure as a service) solution. Openstack has been designed for simplicity in use and scales massively [5].

3.2 ARCHITECTURE

OpenStack has a modular architecture with various components providing different services. These services integrate together through public APIs. The APIs are used by end users for provisioning a service and by the services

themselves to use another service. Figure 1 shows the various components and their relationship:

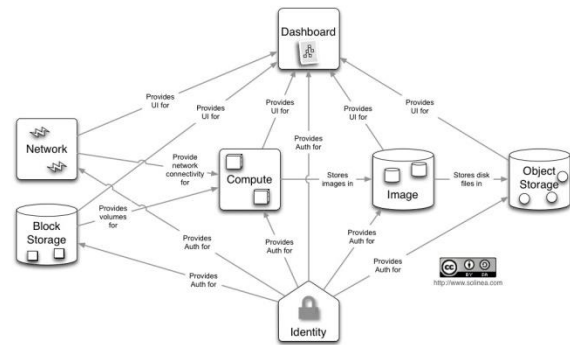


Figure 1[5]

3.2.1 Components of OpenStack

We list below the components of OpenStack [5][6][7]:

- **Dashboard (Horizon)** provides a web-based graphical user interface for accessing, provisioning and controlling various OpenStack services. Alternatively, the developers can access or write tools for managing resources using native OpenStack API.
- **Nova (Compute Node)** is used to create and manage virtual machines (compute instances) on demand. It is the main component of an IaaS system and is also known as cloud computing fabric controller.
- **Neutron (Network Node)** manages network connectivity for different OpenStack services such as OpenStack compute. It is accessed using APIs and provides massively scalable multitenant network connectivity. IP addresses can be provided as static or floating allowing for dynamic rerouting the traffic during maintenance or in case of failure. Additional network services like firewalls and virtual private networks can also be managed using pluggable extendible framework.
- **Swift (Object storage)** provides storage for unstructured data objects. It is also API-accessible, scalable and fault tolerant distributed storage system. Fault tolerance is achieved through redundancy by storing files in multiple locations spread throughout the system. When a server or hard disk crashes, OpenStack replicates its contents from a different location in a new node using OpenStack software logic.

- **Block Storage: Cinder** provides persistent block level storage devices for use with OpenStack compute instances. The user can provision storage volumes (cinder volumes) using OpenStack native API without any knowledge of their location or device type on demand.
- **Identity: KeyStone** provides identity service needed for authentication and authorization of various OpenStack services.
- **Image Service: Glance** stores and retrieves virtual machine disk images from object storage swift. OpenStack Compute can access virtual machine image via RESTful API for instance provisioning. Other modules like Heat can also query virtual image metadata for providing high availability.
- **Telemetry: Ceilometer** collects data about resource usage of deployed clouds. This data can be used for billing as well as generating alarms when usage exceeds the threshold limit.
- **Orchestration: Heat** orchestrates multiple composite cloud applications by using either the native HOT template format or the AWS CloudFormation template format. It automates provisioning of a running stack consisting of compute instances, storage, floating IPs using template written as a text file.
- **Database Service: Trove** provides scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines.
- **Messaging Service: RabbitMQ** provides the messaging service for internal communication between various OpenStack components.

4. HIGH AVAILABILITY

Availability is defined as the system ability to continuously provide service [8]. High availability aims to reduce the time period during which service is unavailable termed system downtime. It also aims at reducing accidental data damage. High availability in a cloud based environment focusses on layers, namely OpenStack, application, virtual machine, and control layer (responsible for managing and facilitating resources).

For OpenStack cloud, this means that cloud service is available to users whenever they need it. It is measured as

$$\text{Availability} = (1 - \text{downtime} / \text{service time}) * 100$$

[9][10]

A highly available system is a system which is available for duration percentage 99.99% or above which roughly equates to less than an hour of cumulative downtime per year [11][12]. High availability can be achieved in three ways. First approach deals with redundancy of data and services. Second way is based on detection technology which monitors the usage and reports failure. Third approach incorporates use of recovery module for recovering from failure.

4.1 AUTOSCALING

The capability of a system to automatically scale up or down in proportionate with demand is known as autoscaling [13]. Autoscaling is essential for availability and optimal usage of resources. For enterprise applications, services are available during peak hours as well as they need not pay any extra cost during low demand[14][15]. Cloud service users specify metrics to be observed, their threshold value and alarms. Whenever observed metric value crosses a particular threshold value, alarms are raised and either new resources are provisioned (scale up) or currently provisioned resources are released (scale down) based on scaling policy. For example a scale up trigger defined as “add 1 instance when cpu usage increases by 60% for 2 minutes” will result in creation of another web server instance when cpu utility metric goes up by 60% for 2 minutes. Scale down trigger defined as “Remove 1 instance when cpu usage decreases by 20% for 5 minutes” will result in removal of an instance when cpu usage decreases by 20% for 5 minutes. [16].

4.2 HEAT

Heat is the OpenStack orchestration engine. It launches multiple composite cloud applications based on AWS Cloud Formation template or native HOT template through Openstack - native REST API or CloudFormation compatible Query API. Heat manages whole lifecycle of cloud application. Initially, resources required for cloud application like servers, storage, floating IPs along with their relationships like this server on this storage are specified in a text file (template) by the user. These are then created in correct order by Heat using OpenStack API and complete application is launched. While the application is running, any change in resource required can be made by editing the template file and Heat does the necessary changes. At the end of application, Heat also does the entire cleanup. When scaling group is specified as a resource in a template, Heat provides autoscaling based on alarms raised by Ceilometer [17] [18].

4.2.1 Heat Architecture

Heat engine consists of a number of Python applications: heat, heat-api, heat-api-cfn and heat engine. Heat engine does the main work of orchestration based on API request sent either by heat-api or through the CLI tool, heat-api-cfn component provides Query API [18].

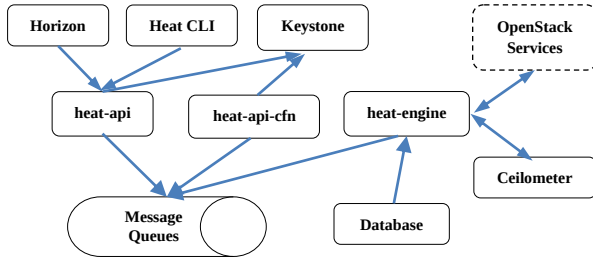


Figure 2 Heat from Infrastructure Side[19]

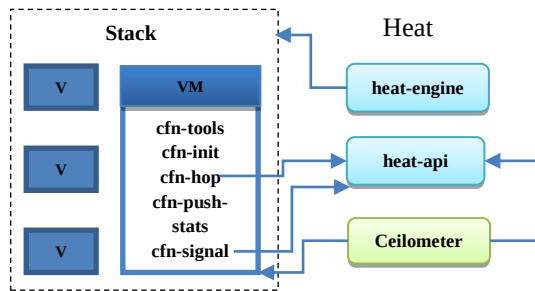


Figure 3 Heat from Instance Side [19]

Heat stacks are defined using templates. Hot (Heat Orchestration Template) is new template format meant to replace the Heat Cloud Formation compatible format – CFN as the native format supported by the Heat over time. HOT templates are defined in YAML [20]. Hot template structure comprises of heat_template_version, description, parameter_groups, parameters, resources and outputs. heat_template_version indicates the version of the specific template. Description is an optional key which specifies the description and the load that can be handled with this template. Section parameter_groups is an optional section which specifies the grouping of input parameters and the order of their specification. It provides the name of required input parameters i.e. parameters which should be specified while instantiating the template. Section resources include specification of resources of the template. Output section specifies the parameters returned (output parameters) on template instantiation.

4.3 Autoscaling with Heat and Ceilometer

Ceilometer is responsible for monitoring the entire OpenStack environment and collecting the information

about the resource utilization and the performance also termed metering. Ceilometer when integrated with Heat helps in achieving autoscaling [21][22][23], leading to better resource utilization and high availability. Ceilometer comprises of mainly three components as shown in Figure 4. First component is Compute Agent which is responsible for gathering statistics about resource usage termed metric. Second component is Alarm Evaluator which triggers the alarm if the metric crosses a particular threshold. API service component returns statistics over the recent time window which helps in determining the action to be taken when alarm is triggered. Rather than periodically evaluating the metrics as in case of Heat 1.0, API service asynchronously notifies the alarm to the heat engine. The API service also facilitates alarm history.

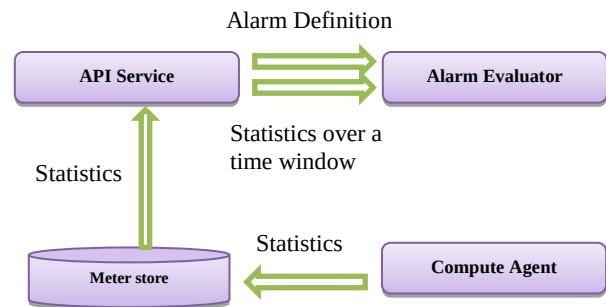


Figure 4 Ceilometer architecture design

Along with resources, instances, network elements (load balancer) required, heat template now also stores the alarm definitions which indicates business/idleness of instances, and the appropriate action to be taken when alarm is triggered. It also specifies membership of autoscaling group. Groups in ceilometer are recognized via nova user metadata which represents membership of autoscaling group. Nova user metadata is a namespace convention that allows heat to tag all the instances that run in particular autoscaling group such that it is recognizable to ceilometer.

Heat Engine saves user metadata in the application stack as shown in Figure 5. Then, it creates alarm that bounds high and low watermark. Ceilometer RESTful API service allows you to control an alarm lifecycle. Compute agent in ceilometer talks to local hypervisor for gathering statistics. Alarm evaluator fetches the alarm definition via API service. Then, it talks to API service and get the statistics over the configured time window and checks to determine whether it has crossed thresholds. In case, threshold is crossed, alarm evaluator triggers the alarm. Thresholds may be further categorized as low and high watermark. If metric (say, cpu_util) is found to be less than low water mark, then it indicates idleness of the application stack and urges the need to downscale pool of resources in application stack. However, if metric is found to be more than high water

mark, then it indicates business of the application stack and urges the need to upscale pool of resources in application stack. Since, crossing thresholds triggers low or high watermark alarm, it asks heat engine (orchestration) to do appropriate scaling. So, application stack scales with change in load conditions. Autoscaling API is responsible for generating the new template. Figure 6 depicts how Heat engine scales out stack in case of high watermark alarm.

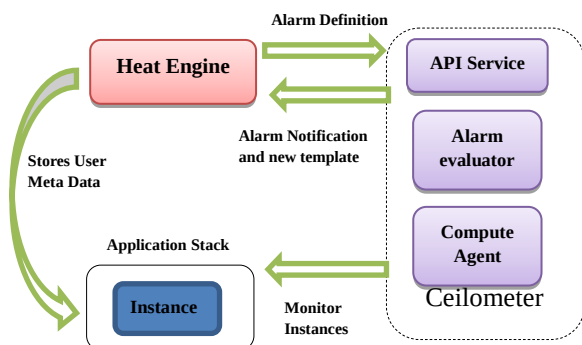


Figure 5 Ceilometer triggers alarm to Heat Engine

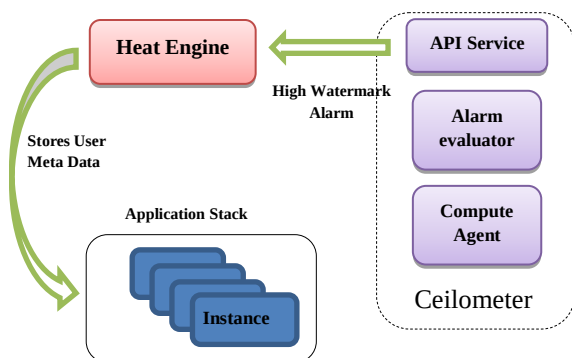


Figure 6 Heat Engine scales out stack

Next we describe AutoScaling Resources specified in Heat template file for providing autoscaling [24][25]:

ScalingGroups A scaling group specifies the resources to be scaled. Properties are defined for minimum and maximum size of group, cooldown period permitted between autoscale operations and the resources that will be duplicated for scaling. APIs exist for creating a scaling group, fetching a scaling group and deleting a scaling group.

Scaling Policies Scaling policies describe a particular way to change scaling group, such as "add -1 capacity" or "add +10% capacity" or "set 5 capacity". Properties are defined for cooldown period. Change in scaling group and its type (exact change or percentage change). APIs exist for creating

scaling policy, retrieving a list of scaling policies associated with a scaling group, fetching a specific scaling policy and deleting a scaling policy.

Scaling Policy Execution This resource allows executing a scaling policy directly, in a way which requires authentication. API is provided for executing a scaling policy.

WebHooks Webhooks are "capability URIs" -- auto-generated URIs with random secrets in them, which by themselves represent the capability to perform an action. Policy webhooks are URLs which, when POSTed to, will execute the policy (unless it is on cooldown). APIs are provided for creating a new webhook, retrieving a list of webhooks associated with the scaling policy, getting information about a specific webhook, deleting a specific webhook and executing a specific webhhok.

CeilometerAlarms This is used for defining Ceilometer alarms. The alarms determine when Heat should scale the application by setting a threshold on a metric that Ceilometer monitors. Properties are defined for specifying resource metric to be monitored, threshold value and the action to be taken by specifying the appropriate scaling policy. Generally two alarms are specified one for upscaling and one for downscaling.

5. CONCLUSION

OpenStack is popular IaaS cloud platform which facilitates sharing of resources. However, a typical usage of resources varies significantly with time. For example, in telecommunications, cloud resource usage is high during peak hours and low during nights. OpenStack offers autoscaling for providing high availability and optimal usage of cloud resources. The orchestration engine Heat automatically scales up or scale down based on alarms raised by ceilometer.

REFERENCES

- [1] Mell, Peter, and Tim Grance, "The NIST definition of cloud computing", 2011.
- [2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica and Matei Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing", Technical report EECS-2009-28, UC Berkeley, 2009.
- [3] "Cloud Computing", http://en.wikipedia.org/wiki/Cloud_computing.
- [4] Gregor von Laszewski, Javier Diaz, Fugang Wang, Geoffrey C. Fox, "Comparison of Multiple Cloud

- Frameworks". Proceeding of IEEE Cloud 2012, pp 734-741.
- [5] "Chapter 1. Getting Started", <http://docs.openstack.org/training-guides/content/operator-getting-started.html>.
- [6] Stefan Brenner, Benjamin Garbers, and Rudiger Kapitza, "Adaptive and Scalable High Availability for Infrastructure Clouds", Distributed Applications and Interoperable Systems Lecture Notes in Computer Science 2014, pp 16-30.
- [7] "Chapter 1. Getting started with OpenStack ". http://docs.openstack.org/admin-guide-cloud/content/ch_getting-started-with-openstack.html.
- [8] Vargas, Enrique, and Sun BluePrints. "High availability fundamentals." *Sun Blueprints series* (2000).
- [9] Benz K., Bohnert T., "Dependability Modeling Framework: A test procedure for High Availability in Cloud Operating Systems", Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th , pp1-8.
- [10] Rahul Ghosh, Francesco Longo, Flavio Frattini, Stefano Russo, and Kishor S. Trivedi, " Scalable Analytics for IaaS Cloud Availability", Cloud Computing, IEEE Transactions Volume:2, Issue: 1 , pp 57 - 70
- [11] Brock, Dean, "A Recommendation for High-Availability Options in TPC Benchmarks", Data General 1999.
- [12] "Chapter 1. Introduction to OpenStack High Availability", <http://docs.openstack.org/high-availability-guide/content/ch-intro.html>.
- [13] "Auto Scaling", <http://aws.amazon.com/autoscaling/>.
- [14] Wadia Y. , Gaonkar R. , Namjoshi J. , "Portable Autoscaler for Managing Multi-cloud Elasticity" , International Conference on Cloud & Ubiquitous Computing & Emerging Technologies, 2013 , pp 48 –51.
- [15] Kartheek Kanagala, K. Chandra Sekaran , "An Approach for Dynamic Scaling of Resources in Enterprise Cloud", Cloud Computing Technology and Science (CloudCom), 2013 IEEE, pp: 345 – 348
- [16] M. Mao , J. Li and M. Humphrey "Cloud Auto-Scaling with Deadline and Budget Constraints", Proc. 11th ACM/IEEE Int'l Conf. Grid, pp.41 -48, 2010
- [17] "Heat", <http://docs.openstack.org/developer/heat/>.
- [18] " Heat", <https://wiki.openstack.org/wiki/Heat>.
- [19] "Heat", <https://www.openstack.org/summit/openstack-paris-summit-2014/session-videos/presentation/using-heat-and-other-tools-to-deploy-highly-available-environments>.
- [20] "Heat Orchestration Template (HOT) Guide ", http://docs.openstack.org/developer/heat/template_guide/hot_guide.html.
- [21] Autoscaling with Heat and Ceilometer ". <http://techs.enovance.com/5991/autoscaling-with-heat-and-ceilometer>.
- [22] "Ceilometer + Heat = Alarming", <http://www.slideshare.net/NicolasBarcet/ceilometer-heatequalsalarming-icehousesummit>.
- [23] Hyungro Lee , Gregor von Laszewski, Fugang Wang, Geoffrey C. Fox, " Towards Understanding Cloud Usage through ResourceAllocation Analysis on XSEDE", Technical Report March 25 2014, Community Grids Lab Publications
- [24] "Heat/ AutoScaling", <https://wiki.openstack.org/wiki/Heat/AutoScaling>.
- [25] "Heat Autoscaling API v1", <http://docs.heatautoscale.apiary.io/>.
- [26] Young-Sik Jeong , Jong Hyuk Park, "High availability and efficient energy consumption for cloud computing service with grid infrastructure", Computers and Electrical Engineering 39 (2013) pp15–23
- [27] Marc E. Frincu, Ciprian Craciun , "Multi-objective Meta-heuristics for Scheduling Applications with High Availability Requirements and Cost Constraints in Multi-Cloud Environments", 2011, Fourth IEEE International Conference on Utility and Cloud Computing, pp 267 - 274