Monitoring of Virtual Routers of OpenStack with Monasca

(Session 2012-2016)

Submitted By

Hifza Khalid	**2012-CE-81**
Qurat-ul-Ain Zafar	**2012-CE-64**
Rubab Zahra Sarfraz	**2012-CE-67**

Project Supervisor

Dr. Irfan Ullah Chaudhary

Department of Computer Science & Engineering

University of Engineering & Technology, Lahore Pakistan

Monitoring of Virtual Routers of OpenStack with Monasca

(Session 2012 – Computer Engineering)

The thesis is to be submitted to the Department of Computer Science and Engineering, University of Engineering and Technology, Lahore for the partial fulfillment of the requirement for the Bachelor’s degree in Computer Engineering.

Approved on: ______________________________

Internal Examiner	**External Examiner**
Signature: ____________________	Signature: ____________________
Name: ____________________	Name: ____________________
Designation: ____________________	Organization and Designation: ____________________

Chairman	**Dean**
Signature: ____________________	Signature: ____________________
Prof. Dr. M. A. Maud	
Chairman Department of Computer Science and Engineering, UET, Lahore	Dean Faculty of Electrical Engineering, UET, Lahore

Department of Computer Science & Engineering

University of Engineering & Technology, Lahore Pakistan

Declaration

I declare that the work contained in this thesis is my own, except where explicitly stated otherwise. In addition this work has not been submitted to obtain another degree or professional qualification.

Signed: ____________________
Date: ____________________

Abstract

Cloud computing, also known as on-demand computing, is rapidly gaining fame in the IT sector. OpenStack is a cloud management software which provides us with IaaS cloud model. The opensource community is readily adding more features to OpenStack depending upon growing needs of cloud computing. Monasca is a monitoring project of OpenStack used to monitor the health of OpenStack services and cloud infrastructure. Network monitoring capabilities of Monasca are very limited. There is no mechanism to monitor the networking metrics of virtual routers distributed across OpenStack network and compute nodes. Interacting with the industry people, we came to know that there is a need of monitoring virtual routers in the OpenStack industry. To add this fuctionality, we have been understanding the working and architecture of OpenStack and Monasca.

Dedicated to our parents and advisor

Contents

List of Figures

Chapter 1

Introduction

We are writing a plugin that would enable the OpenStack network admins and operators to monitor the health and status of virtual routers per tenant in real-time. Let's establish our problem statement by considering an industrial use case:

> Suppose you're a company running OpenStack as your on-premise cloud management operating system. You're hosting a web-server on one of your virtual machines which gets external internet connectivity via a virtual router. As an OpenStack admin, while you're working on something important, you get a call from an angry user complaining about unavailability of web-server or even worse, you find FBI standing at your doors accusing you for a DDOS(distributed denial-of-service) attack. You're standing there blank wondering what could have possibly happened?

Problems like these are common when you are running a huge cloud environment where there are a lot of services interacting with each other in real-time. Whether you're a multi-millionaire banking company hosting millions of transactions per second like PayPal, or an average business firm hosting your own data in a cloud, you would like to know beforehand about any of the situations that could result in malfunctioning of your system. For instance, in above mentioned scenario, what if you get a notification warning you about unusual outbound traffic at a particular router interface or, while you're leisurely checking the performance of your cloud, you see unusual traffic spikes at a particular department router in the form of graph, you can look into it and see what's actually wrong. We intend to solve these kinds of monitoring issues in our project that would enable us to debug the virtual router related problems, and also minimize these issues by informing us beforehand about them.

1.1 Why open source development?

The idea of open sourcing has always been fascinating to us. Understanding large-scale complex systems, digging through thousands of lines of code and then adding your own code of that quality is something that students like us would love to challenge ourselves with. There's a lot of learning involved in the process and we wanted to learn as much as possible through our final year thesis.

1.2 Why OpenStack?

Although quite a lot of interesting and challenging opensource projects are going on currently including Docker, Ansible, Kubernetes etc. We chose OpenStack because it mainly utilizes our academic concepts of domains like Operating Systems, Networking, Data Structures, Computer Architecture etc. It is currently under heavy development which makes this an excellent opportunity to contribute to future technology powering our data and services.

1.3 Organization

This report covers the modules that we've been successful to complete till date. Chapter 1 introduces our problem statement, goals, motivation as well as the methodology and approach that we're using. Chapter 2 covers the literature review of our project. Chapter 3 and 4 give details about OpenStack and Monasca respectively including their deployments and some test commands. Chapter 5 describes how we came across this project idea and community members. Chapter 6 gives the design of the plugin that we are proposing to implement.

1.4 Contribution

Our contribution would help all kinds of users of OpenStack, whether it is a small firm comprising of 10-15 physical nodes or a large company consisting of 100-1000 nodes, because we are embedding our plugin within current well-established systems.

1.5 Methodology

To enter the opensource community whilst targeting a reasonable industry project, we initially searched for technologies that were famous, went with our research interests and were emerging rapidly around the world. During this process, we came across different technologies and decided to work with cloud computing because it is thought to be the future of IT sector. Getting to know that OpenStack is one of the major cloud managing softwares these days and many major companies have deployed it to manage their clouds, we went through different OpenStack projects and among these, Monasca, an OpenStack monitoring project, intrigued us the most. Monasca is still under active development.

We chose to develop Monasca to enhance its monitoring capabilities for OpenStack. Interacting with different people of the well-known companies, we came to know that Monasca offers very limited network monitoring capabilities. One of the network monitoring features that was lacking in Monasca and considered significant by the industry was monitoring of OpenStack virtual routers. We are in the process of implementing it now. For this we have been digging through the Monasca code to find alternate ways of getting access to the networking metrics of virtual routers.

Some of the limitations that we faced in this phase include shortage of resources. Deploying OpenStack with Monasca requires around 10 GB RAM and our computers support at most 8 GB RAM. We didn't have anybody around who was proficient with OpenStack and Monasca. Web was the only source to get any kind of assistance on the subject. Moreover, we also encountered issues regarding our interaction with the industry people because of our remote location.

1.6 Modules of our project

Our project can be logically divided into the following sub-modules which are described in later chapters.

1. Deployment and understanding of OpenStack
   (a) Learning about Keystone and Neutron
2. Deployment and understanding of Monasca
3. Problem hunt and finding community mentors
4. Design of virtual router plugin
5. Implementation
6. Testing
7. Uploading code to community
8. Getting community approval and merging to official git repo of Monasca

Chapter 2

Literature Review

Cloud computing, which provides computing, storage and networking on-demand, is rapidly gaining fame all over the world. Due to the growing trend of cloud computing, Rackspace and NASA developed OpenStack project to manage the cloud infrastructure. As new features are getting introduced in cloud, OpenStack is also getting richer in its features. One of the companies using OpenStack felt the need of a monitoring tool to monitor the health of OpenStack and the cloud it manages to get aware, beforehand, of any situation likely to cause problems for its customers. Monasca was thus developed. It also supports numerous plugins to monitor different resources in cloud. However, network monitoring capabilies of Monasca are very limited and although it supports addons, there is no plugin designed specifically to monitor virtual routers within the OpenStack cloud infrastructure. In this project, we intend to add the monitoring capability of OpenStack virtual routers in Monasca.

In order to get access to the documentation and videos of OpenStack, guides on how to deploy it for development purposes, OpenStack opensource community, *Rackspace cloud computing* provided OpenStack users with a very useful resource in the form of a website [1]. It helped us not only to get a headstart with OpenStack but also timely informed us about the new additions made in OpenStack.

To understack Monasca, its features and architecture, the website developed by *Hewlett Packard Enterprise* [2] proved very useful. As Monasca developers, we required an up-to-date and reliable information source: this website fully served the purpose by informing us regarding all the major upcomings in Monasca.

The developer blog of *James Denton* also served as one the major information resources for this project. It helped us to gain an in-depth understanding of Neutron routers (virtual), their functionality and how they are implemented in OpenStack [3].

Github [4] served as a primary source to understand Monasca on code level. It provided us with an in-depth documentation and code of all the components of Monasca including *monasca-api* and *monasca-agent*. Moreover, through this site, we came across three major Monasca deployment guides that helped us to choose one of these on the basis of our available compute and memory resources.

Since Neutron uses distributed virtual routers, this web page [5] assisted us to gain sound knowledge of the cause for using distributed routers on compute nodes instead of using single router on the network node in OpenStack. Using this knowledge, we were able to design our monitoring plugin so that information regarding the virtual routers is gathered from both the computer and the network nodes.

Chapter 3

Architectural Overview and Deployment of OpenStack

This chapter describes the concepts of OpenStack and its basic usage necessary for understanding this thesis. It provides description from what exactly cloud computing is to how OpenStack is playing its part in shaping cloud management across the globe.

3.1 Cloud Computing

Cloud computing enables its users to use compute, networking and storage resources as a utility rather than having to build and maintain storing and compute infrastructures in-house. It provides the consumers with a comprehensive virtualization model from infrastructure through application delivery where they have to pay for only what they use. It consists of three basic models namely software as a service(e.g. Google Apps, Workday), platform as a service(e.g. Cloud foundry, Google App Engine) and infrastructure as a service(e.g. OpenStack, Amazon web services), hence cloud computing is often referred to as *stack* because these services are stacked over each other.

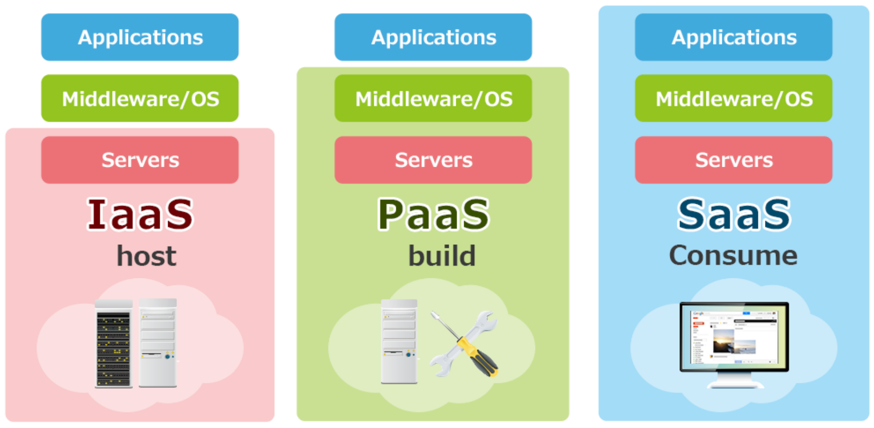


Figure 3.1: Three layer model of Cloud Computing

3.2 OpenStack

OpenStack provides us with IaaS which is, the ability to programmatically create, manage and consume infrastructure elements including storage volumes, network and compute resources. OpenStack enables its users to use cloud models based on different deployment use cases. These include private, public and hybrid clouds. Depending upon the needs of the user, one of the cloud models may be adopted. The functionality of these cloud models is achieved using services provided by the components of OpenStack [1]. Using code names, the services of OpenStack are briefly described below.

Nova
: It is the Compute service that allows the provisioning and management of large networks of virtual machines to enable high performance computing.

Cinder
: It provides Block Storage service to compute instances by maintaining the life-cycle of block devices, from their creation and attachment to virtual machines, to their release.

Neutron
: It provides Cloud Networking service to cloud users or tenants, such as IP address management, load balancing and security groups(firewall policies, network access rules, etc.). It provides a framework for software defined networking(SDN) in virtual compute environments.

Glance
: It provides Image service where users can upload and discover data that is to be used with other services. Currently, this includes discovering, registering and retrieving virtual machine images and meta-data definitions.

Swift
: It provides Object storage service that supports storage and retrieval of arbitrary data in cloud. It is best used for static data like media files, virtual machine images and backup files.

Keystone
: It is the identity service for authorization and authentication of users, services and endpoints.

Horizon
: It is the dashboard that provides web interface to cloud administrators and tenants. Through the interface, users can provision and manage cloud resources.

In addition to these services, there are additional projects under the big tent of OpenStack providing its users with numerous useful services like monitoring, orchestration, telemetry etc. Some of the services interacting with each other are shown below.

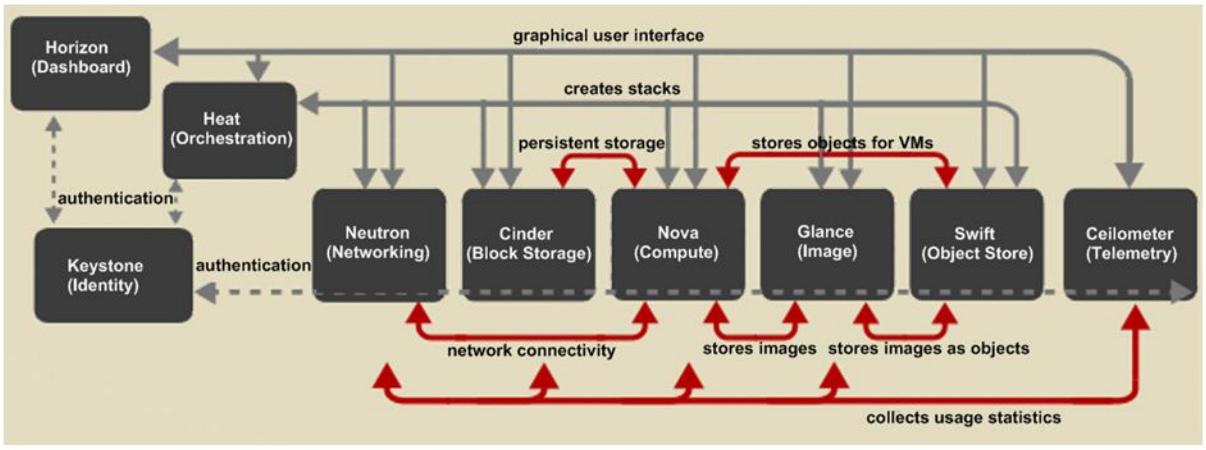


Figure 3.2: OpenStack Projects [6]

As OpenStack is an open source project, anyone around the globe can start a project catering a specific service and apply for it to be declared as official project. Developers upon coming across these projects will start to contribute if it matches their or the companies they are representing interest.

3.2.1 Example Architecture

The architecture of OpenStack depends on its use cases. of The example architecture described below consists of two nodes for launching a basic instance or virtual machine. If optional services like block storage or object storage are required, they would need additional nodes.

Controller Node

The services that run on controller node include identity for authentication and image for getting an operating system image that would eventually used by Nova service to boot an instance. In addition to these services, the controller node hosts management portions of compute and networking, a number of networking agents like neutron-l3-agent, ovs agent etc., and the dashboard for providing user interface. It may also include services that support other OpenStack services such as an SQL database and message queue. It requires two network interfaces at minimum.

Compute Node

The compute host runs hypervisor that operates virtual machines. It uses KVM as its default hypervisor. It also runs networking agents that are used to connect instances to virtual networks. Firewalling service is provided to instances through security groups. A user can deploy a number of compute nodes depending upon its requirements.

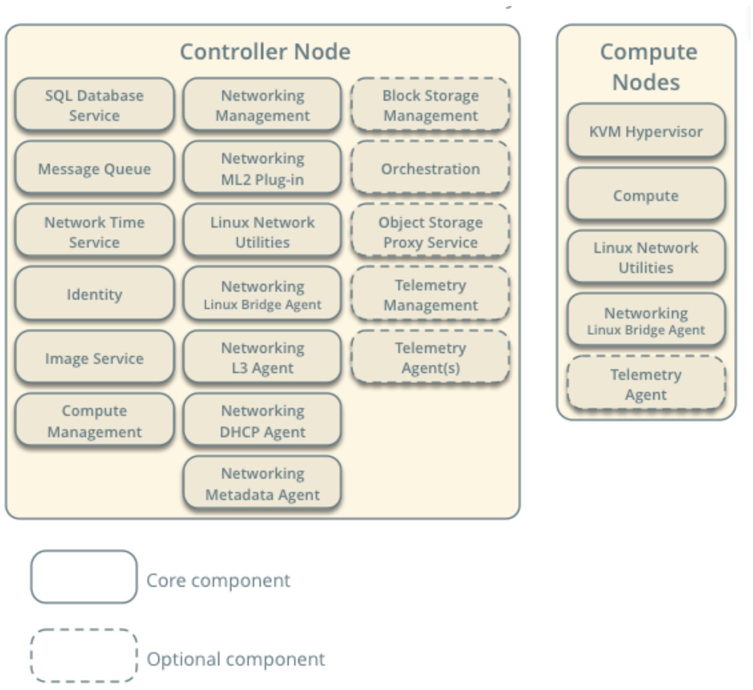


Figure 3.3: Example Architecture of OpenStack [1]

Above mentioned architecture is for two nodes whereas in production environment networking services are hosted on a separate network node.

3.3 Learning Keystone and Neutron

The core services which are directly under the scope of our project are

- Keystone
- Neutron

3.3.1 Keystone

As mentioned earlier, keystone is authentication and authorization service for OpenStack. It is the service that lets the users and other services to access OpenStack. For our project, it's necessary to understand some basic terminology related to Keystone.

Users

Users in Keystone today are generally people. When a user authenticates to Keystone, it actually presents its details to Keystone like his username X password Y and a tenant name Z to which he belongs. X can be a userid or username, Y is a password, but you can authenticate with a token too. Z is a tenant id or tenant name that would be unique. In previous releases of OpenStack, you didn't need to specify a tenant name, but in that case your token wouldn't be very useful, as the token wouldn't be associated with your tenant and you would then be denied of any ACLs on that tenant.

Tenants

A tenant is also known as a project in OpenStack. When you log into horizon, you see a drop down for your tenants. Each tenant corresponds to a tenant id. Tokens are associated with a particular tenant id which means you may require several tokens for a user if you want to work on multiple tenants the user is attached to..

Roles

Adding a user to the tenant id of admin that has administration privileges doesn't mean that user gets admin privileges. That's where roles come in handy. Although the user in admin tenant may have access to its virtual machines and quotas, that user wouldn't be able to query keystone for a user list and action like these that require admin authority. But if you want to achieve that level of functionality, you can add an admin role to that user. That user will be given the ACL rights through which it can act as an admin in the keystone API.

Regions

Regions are more like ways to geographically bind physical resources in groups in the OpenStack infrastructure environment. For instance, you have two segmented data centers and you might put one in region A of your OpenStack environment and another in region B.

Catalog

Keystone also provides another useful thing, the catalog. The Keystone catalog can be thought to be like the phone dictionary for the OpenStack APIs. Whenever you use a command line client, like when you might call neutron port-list to list your ports, neutron first authenticates to keystone and gets you a token to use the API, but it also immediately asks for API endpoints list from Keystone catalog. For keystone, cinder, nova, neutron, glance, swift... etc. Nova will really only use the neutron-api endpoint, though depending on your query you may use the keystone administrative API endpoint. But essentially the catalog is a canonical source of information for where APIs are in the world. In this way, the only thing that a client would need to know is public API of Keystone. Rest of the information can be retrieved from the catalog.

Keystone APIs

Keystone has two APIs. It runs an API on port 5000 and another one up in the 32000 range. The 5000 is the public port. This is endpoint from where you ask for a token or keystone catalog to talk to other services(APIs). The administrative

API would be used for performing actions that require administrative privilege like adding a new endpoint or changing a users password [7].

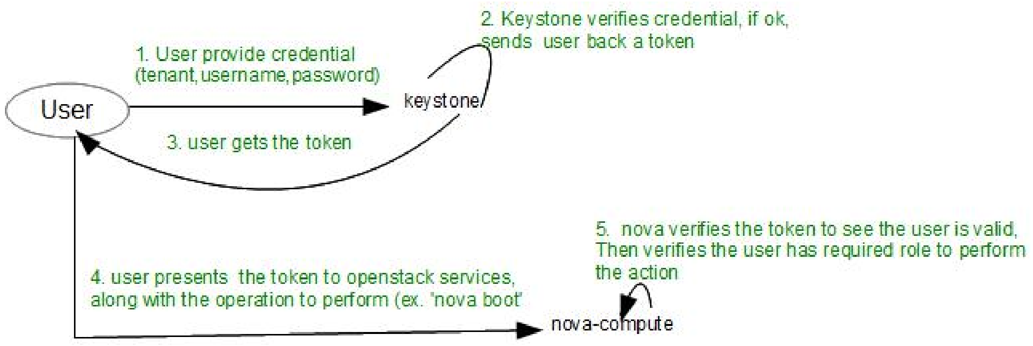

1. User provide credential
(tenant, username, password)
2. Keystone verifies credential, if ok, sends user back a token
User
keystone
3. user gets the token
4. user presents the token to openstack services, along with the operation to perform (ex. 'nova boot'
5. nova verifies the token to see the user is valid, Then verifies the user has required role to perform the action
nova-compute

Figure 3.4: Keystone Authentication Process in a nutshell [8]

3.3.2 Neutron

Since our project is mainly related to monitoring virtual routers of OpenStack, we had to go deeper with Neutron especially to understand how a virtual router is setup. Neutron is basically a Software defined networking project that focuses on delivering NaaS(networking-as-a-service) in a virtual environment. The concepts related to Neutron which are essential for our thesis are as follows

- **Neutron server:** It runs on controller node and handles all the API requests. It routes them to relevant networking plug-in.
- **Networking plug-ins and agents:** These perform the tasks like plugging and unplugging ports, creating networks and subnets, and providing IP addressing. Only one plug-in can be at a time.
- **Messaging queue** It is used for information exchange between server and agents. It is also used to route information between server and neutron database.

Types of Networks: Tenant and Provider

A provider network provides the instances with Internet access. By default, it allows Internet connectivity from virtual machines to outside world using NAT(Network Address Translation). We can then allocate floating IPs and security groups to instances for communication. of Internet. It is owned by administrator tenant as it provides Internet for various tenants.

A Tenant network provides the tenants with access to internal network. It is owned by a tenant and instances residing within a particular tenant use this network for communication between themselves. Multiple tenant networks can exist within the same host that would be isolated from each other. This is achieved by Linux namespaces.

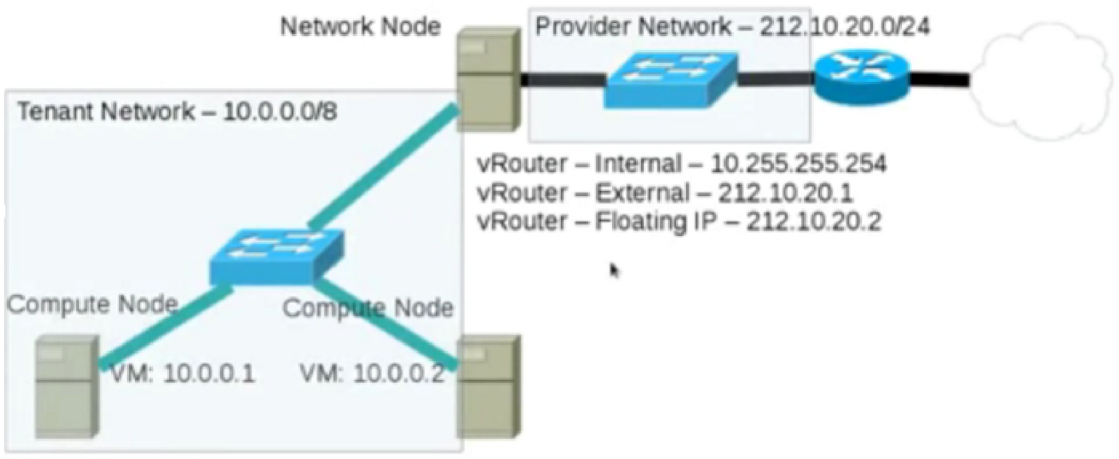

Network Node
Provider Network – 212.10.20.0/24
Tenant Network – 10.0.0.0/8
vRouter – Internal – 10.255.255.254
vRouter – External – 212.10.20.1
vRouter – Floating IP – 212.10.20.2
Compute Node
Compute Node
VM: 10.0.0.1
VM: 10.0.0.2

Figure 3.5: Network types and their functions

Neutron L3 agent

Neutron API allows tenants and admins to create logical routers that connect to layer-2 networks. neutron-l3-agent uses Linux iptables and IP stack to perform actions like NAT and L3 forwarding. In order to provide isolated packet forwarding to different routers related to different tenants, it uses Linux namespaces. Each router have its very own namespace that has its network UUID based name.

Virtual routers created in this manner handle communication between LAN interfaces(usually VLAN, GRE or tenant networks) that are directly connected and a WAN interface(usually VLAN or flat provider network). It is possible to use a bridge for this purpose but existing provider networks is what people mostly use these days.

Distributed Virtual Router(DVR)

With legacy networking, virtual routers used to lie on network nodes which causes problems such as

- Traffic between different instances belonging to different subnets but same tenant have to get routed through network node which affects, obviously, performance.
- Instances to which floating ips have been attached to also send and receive packets via network node router.

The solution of these problems is proposed via distrubuted virtual router that will reside on compute nodes instead of network node(s). For this purpose, the enhanced version of neutron-l3-agent will run on every compute node handling the routing requests.

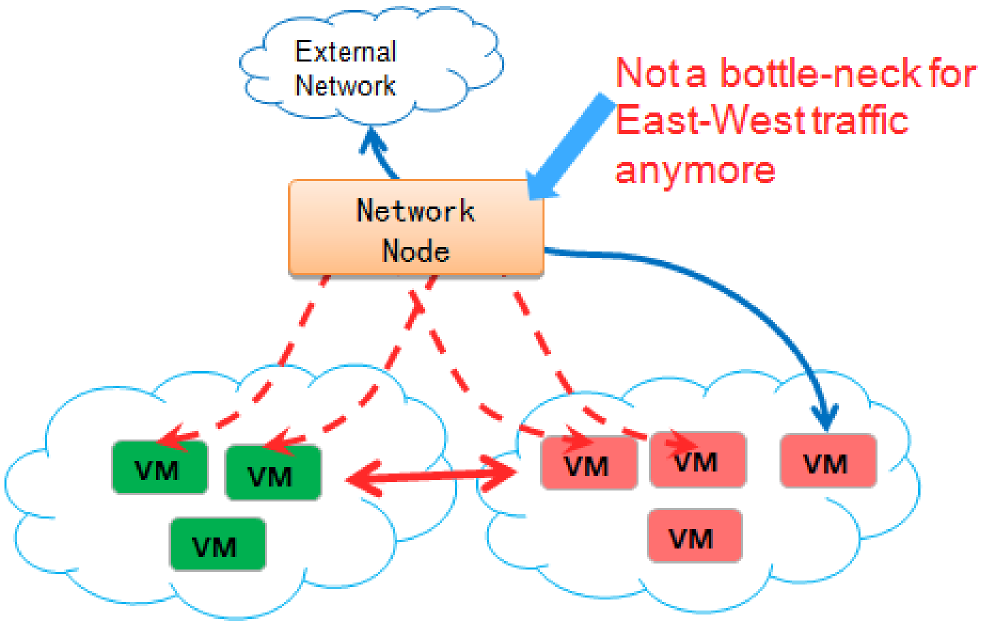


Figure 3.6: Networking with DVR [1]

3.3.3 Deployment

In OpenStack community, the most widely used development flavor of OpenStack is DevStack. It is mostly the first choice of OpenStack developers as it automates most of the deployment and is easy to use. It is available in different configurations, for instance

- Single node setup
- Single VM setup
- Multi-node setup

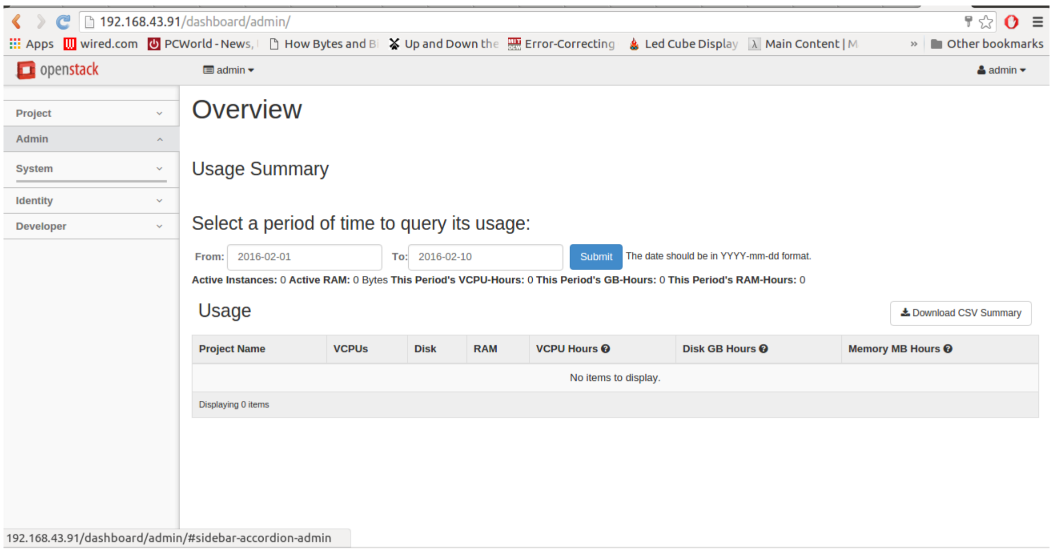
We are currently using single node setup on our bare metal machines. Although it is preferred to use DevStack VM because node setup can intervene with your operating system settings but due to memory constraints, we are bound to use bare metal setup. It requires minimum of 4GB RAM.

Here is how we brought it up [4]:

1. Installed Ubuntu 14.04 Trusy Tahr on our bare metal machines.

2. Installed git and grabbed the latest version of DevStack using the following commands
   (a) `sudo apt-get install git -y || sudo yum install -y git`
   (b) `git clone https://git.openstack.org/openstack-dev/devstack`
   (c) `cd devstack`
3. Setup the *local.conf* file according to our requirements which included the configurations for DevStack, for instance, which service to enable or disable, credentials, ip addresses etc.
4. Changed directory to devstack and ran `./stack.sh`

After its successful installation, we were presented with its dashboard(horizon). Logging in as an admin user, we were able to launch Vms, create different network topologies per tenant etc. These functions are also achievable via OpenStack command-line interface which is a python client.


192.168.43.91/dashboard/admin/

Overview

Usage Summary

Select a period of time to query its usage:

From: 2016-02-01 To: 2016-02-10 Submit The date should be in YYYY-mm-dd format.

Active Instances: 0 **Active RAM:** 0 Bytes **This Period's VCPU-Hours:** 0 **This Period's GB-Hours:** 0 **This Period's RAM-Hours:** 0

Usage

Project Name	VCPUs	Disk	RAM	VCPU Hours	Disk GB Hours	Memory MB Hours
No items to display.						

Displaying 0 items

192.168.43.91/dashboard/admin/#sidebar-accordion-admin

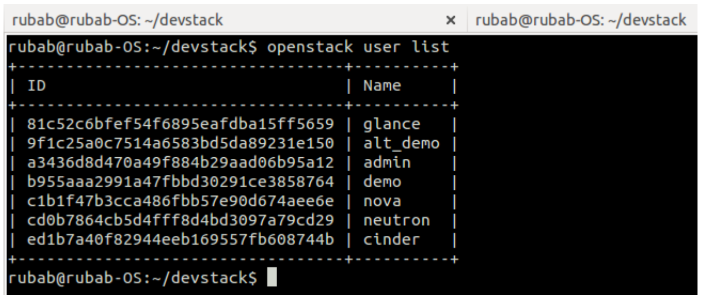
Figure 3.7: OpenStack Dashboard(Horizon)

Usage

We can use openstack command-line interface to perform different actions. Following are some of the many commands that we used during our project.

- to list OpenStack users, we can use following command-line

```
openstack user-list
```



```
rubab@rubab-OS:~/devstack$ openstack user list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 81c52c6bfef54f6895eafdba15ff5659 | glance   |
| 9f1c25a0c7514a6583bd5da89231e150 | alt_demo |
| a3436d8d470a49f884b29aad06b95a12 | admin    |
| b955aaa2991a47fbbd30291ce3858764 | demo     |
| c1b1f47b3cca486fbb57e90d674aee6e | nova     |
| cd0b7864cb5d4fff8d4bd3097a79cd29 | neutron  |
| ed1b7a40f82944eeb169557fb608744b | cinder   |
+----------------------------------+----------+
rubab@rubab-OS:~/devstack$
```

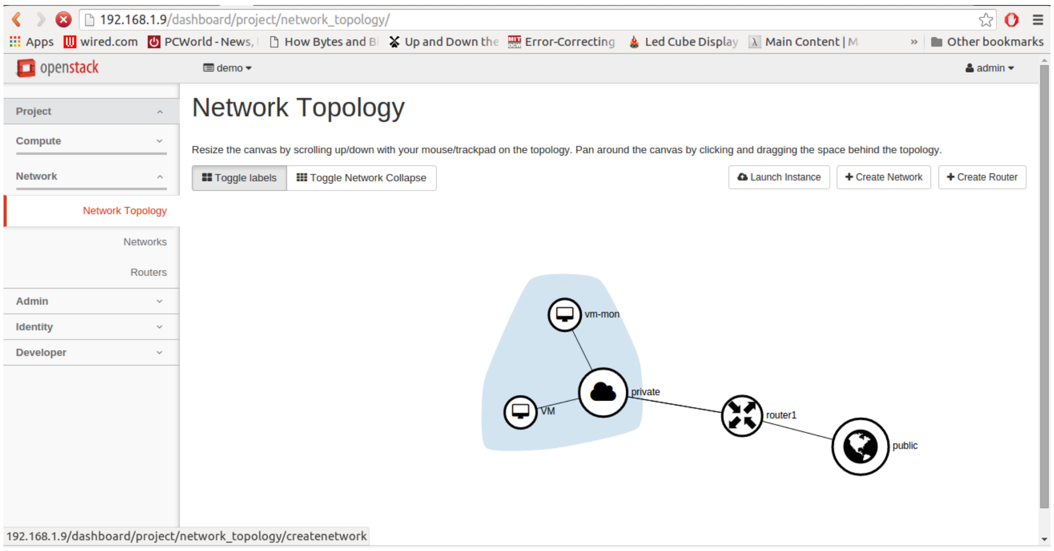
Figure 3.8: List of OpenStack users

- We can source the credentials of a user belonging to a particular tenant. We use this to reduce our work of explicitly providing credentials each time we need to perform an action.

  ```
  source openrc admin demo
  ```

  where admin is user name and demo is tenant name.

- Following is a simple network topology that we created in OpenStack for test purposes. It contains two instances attached to a subnet and getting external internet connectivity via a router which is connected to provider network.



Network Topology

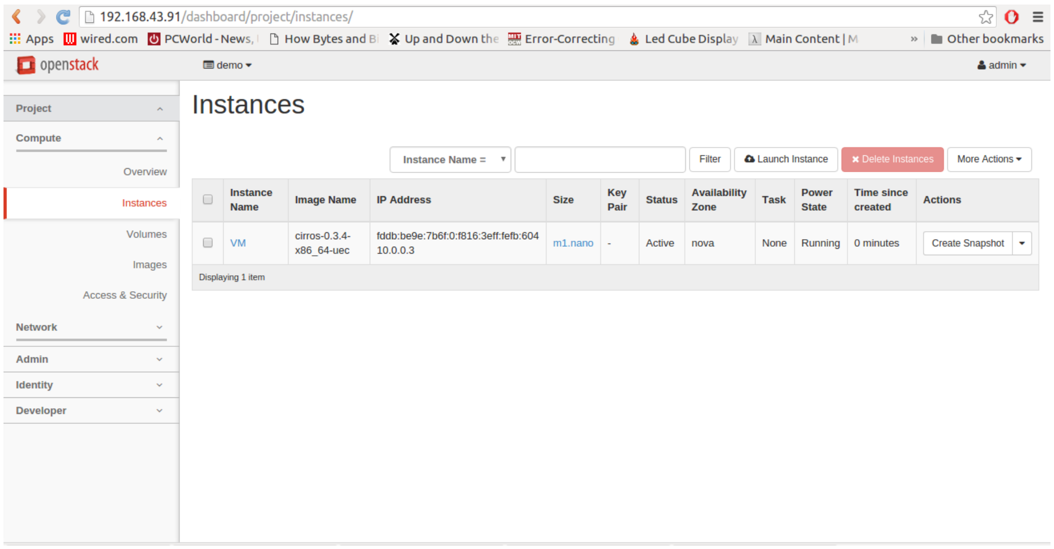
Resize the canvas by scrolling up/down with your mouse/trackpad on the topology. Pan around the canvas by clicking and dragging the space behind the topology.

Figure 3.9: Example Network Topology

- To launch an instance in OpenStack, Nova API is used with relevant information. We can boot it from cli as well as from dashboard. Following is a basic command to boot a cirros image. We can give it multiple options that it supports.

  ```
  nova boot --image cirros-0.3.1-x86_64-uec {flavor m1.nano {name
  Vmm
  ```

  How it appears in Horizon



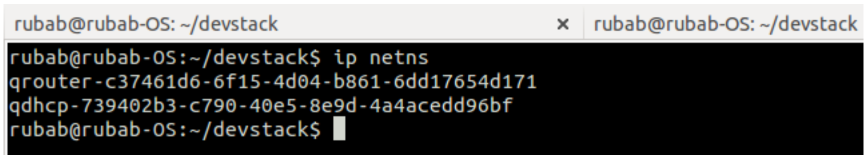
Instances

	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
	VM	cirros-0.3.4-x86_64-uec	fddb:be9e:7b6f:0:f816:3eff:fefb:604 10.0.0.3	m1.nano	-	Active	nova	None	Running	0 minutes	Create Snapshot

Displaying 1 item

Figure 3.10: Booted Instance in Horizon

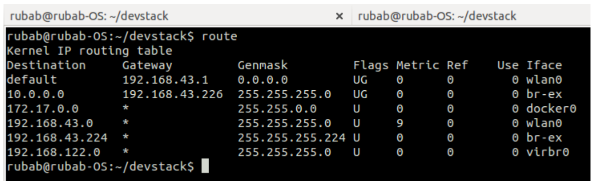
- We can list routers residing on a node using a Linux command, as router is a just a namespace in linux environment.



```
rubab@rubab-OS: ~/devstack          rubab@rubab-OS: ~/devstack
rubab@rubab-OS:~/devstack$ ip netns
qrouter-c37461d6-6f15-4d04-b861-6dd17654d171
qdhcp-739402b3-c790-40e5-8e9d-4a4acedd96bf
rubab@rubab-OS:~/devstack$
```

Figure 3.11: List of virtual routers on a Computer node

- Following command shows how host's routing table changes when an OpenStack's network topology is created.



```
rubab@rubab-OS:~/devstack$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.43.1    0.0.0.0         UG    0      0        0 wlan0
10.0.0.0        192.168.43.226  255.255.255.0   UG    0      0        0 br-ex
172.17.0.0      *               255.255.0.0     U     0      0        0 docker0
192.168.43.0    *               255.255.255.0   U     9      0        0 wlan0
192.168.43.224  *               255.255.255.224 U     0      0        0 br-ex
192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
rubab@rubab-OS:~/devstack$ 
```

Figure 3.12: Routing table at Host node

Chapter 4

Architectural Overview and Deployment of Monasca

Monitoring solutions to monitor the health of OpenStack infrastructure and its VMs have been around for quite a while but, in many respects, they fail to address the requirements of monitoring large-scale public and private clouds. Traditionally, these provide scalability, performance and retention of data for a very limited number of systems usually in the order of hurdredths. In a large-scale cloud, hundreds of thousands of physical servers and virtual machines (VMs) need to be monitored which results in huge amount of data to be monitored. The monitoring data needs to be stored in an on-line and queryable form for longer data retention periods depending upon the requirements of the business. Such long data retention periods are necessary for business continuity and analytics.

Cloud infrastructure is constantly evolving with VMs and auto-scaling to varying loads. Thus, elasticity in monitoring solutions is very important. Most of the current monitoring solutions assume a static infrastructure where the monitoring tool needs to be configured again every time a new VM or a physical server is added. This results in the monitoring team/server being the bottleneck. Self-service model that empowers teams to easily add new services and resources and monitor them without the participation of monitoring teams is necessary [9].

Self-describing data that allows flexible name, value pairs is necessary to identify metrics. Most of the current monitoring solutions allow very limited meta-data offering no flexibility such as host name and service which results in ambiguities and mismatch [9].

Dynamic adjustment of alarms at run-time is necessary to be able to tune the system over time, which is not support by most of the systems. Generalization of alarm definitions is also necessary to avoid to manually declare alarms which may share many common attributes and differ in only one of them. Moreover, alert fatigue 1 is also a very common problem with every thresholding system.

Monitoring-as-a-service use cases addressed by proprietary solutions are not opensource. Multi-tenancy, which is the ability to stream metrics from cloud resources of customers, is significant while isolating the data from other customers. Consolidation of health alerting, notifications, metrics and monitoring-as-a-service from multiple systems must be done on a higher level to minimize complexity and do analysis on the data.

Monasca caters for the needs of a comprehensive monitoring solution. It is a scalable, high-performance, open-source monitoring solution that allows multi-tenancy and longer data retention periods as compared to any other monitoring solution. It is build up of high-speed message queues, databases and computational engines making it one of the best monitoring solutions in town. All of its components can be horizontally scaled out thus supporting elasticity in the cloud infrastructure. All external interaction with the Monasca service is done through an API which can be made available on as many systems as required thus allowing a self-service model. In Monasca, the number of dimensions for a metric is also pretty flexible. Moreover, alarm definitions are created automatically by the threshold engine from matching metrics or dimensions thus reducing the overhead of managing alarms [9].

4.1 Architecture

Following figure show different components of Monasca and how they are interralated to one another.

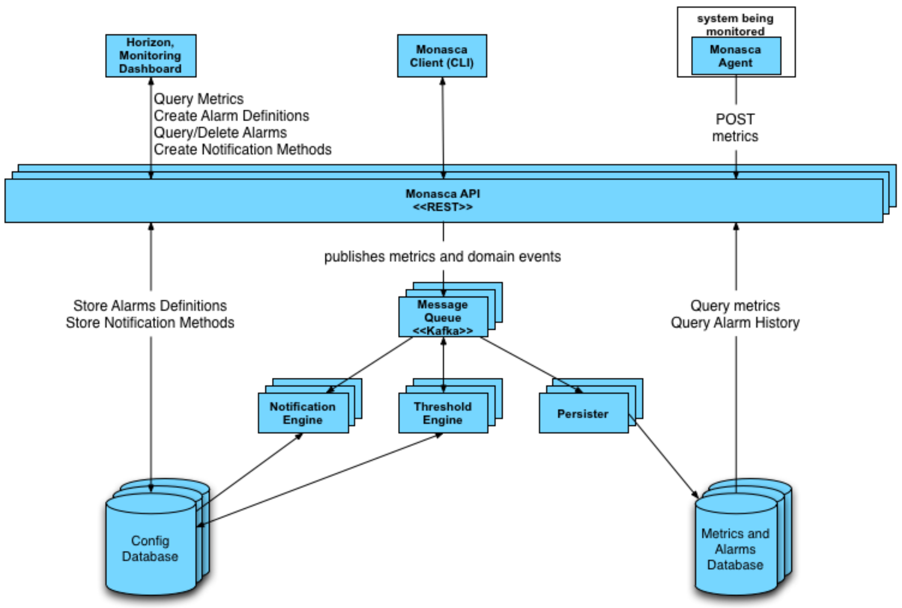


Figure 4.1: Architecture of Monasca [2]

All the major components of Monasca and their functions are described below.

Monasca agent

It is one of the major components of Monasca that resides on the system being monitored and is responsible for posting metrics, which can be based on system, nagios plugins, statsd or other kind of checks available in Monasca by default, to the monasca-api. Within the Monasca agent, its sub-components work in the fashion described below.

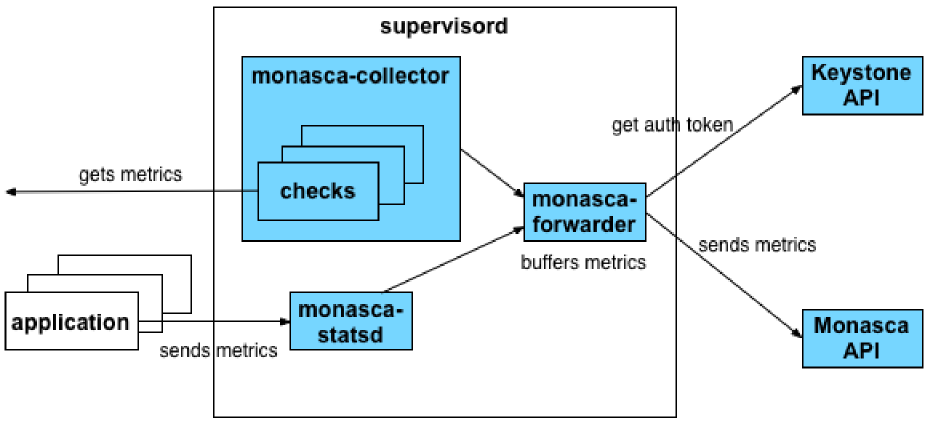


Figure 4.2: Architecture of Monasca agent [4]

- Collector: This component collects sytems metrics or other metrics from additional plugins after a specific configurable time-interval.
- Statsd deamon: This componenst is responsible for sending statsd type metrics to the monasca agent. These message are sent by the users asynchronously. However, after a fixed time period, these messages are flushed to monasca-agent.
- Forwarder: It collects metrics from both collector and statsd deamon and after getting authentication from Keystone, sends them to the monasca-api.

Monasca API

It is the gateway to all the communication taking place with Monasca. It is responsible for

- storing metrics, alarms definitions, alarm history and notification methods
- querying metrics, alarms, statistics for stored metrics
- creating and deleting of alarm definitions and notification methods
- updating alarm definitions

- sending notifications to the users via email when alarms trigger

Perister
It recieves metrics and alarm state transitions from the message queue and stores them in the metrics and alarms database. It can be horizonatally scaled to any number of Persisters for consuming more messages from the message queue.

Threshold engine
It evaluates alarms by comparing metric values with the threshold values defined in alarms. It publishes alarms to the message queue if the value exceeds the threshold value. It is based on a real-time computation system *Apache Storm*.

Notification engine
As the name suggests, this engine is responsible for consuming alarm state transitions from the message queue and notifying the user via email or some other method as define in the notification methods.

Message Queue
Metrics and alarm transition state messages, published by the Monasca API, are recieved by the message queue. These messages can be consumed by other Monasca components for different purposes. It can also receive published messages from the Threshold engine which are received by the Persister and the Notification engine. It is based on Kafka.

Config Database
Notification methods and alarm definitions and stored in this database. Currently, Monasca supports MySQL database.

Metrics and Alarms Database
Metrics published by Monasca API and alarm state transitions published by the Threshold engine are stored in this Database. It maintains history of metrics and alarm state transitions over a period of time. Monasca supports Influxdb and Vertica.

Monasca CLI
It is a command line client and library that allows communication with the Monasca API. Throuth this clinent, users can control Monasca API.

Monitoring UI
It is a horizon dashboard that allows operators/users to visualize the health of Monasca, OpenStack and the cloud managed by OpenStack.

4.2 Deployment

Since Monasca is still under development, it is currently difficult to setup without any assisstance. There are two famous solutions available for deploying it:

- monasca-vagrant
- monasca-devstack

We used monasca-vagrant setup having the following steps

1. run `sudo apt-get install virtualbox`
2. Download and install latest vagrant from http://www.vagrantup.com/downloads.html.
3. run `sudo pip install ansible`. Ansible version 1.8+ is required.
4. run `ansible-galaxy install -r requirements.yml -p ./roles`

We are using department's computers for monasca vagrant deployment and testing environment.

4.3 Usage

When Monasca gets deployed, the OpenStack dashboard looks something like as show in the following figure. In the sidebar of OpenStack, a new tab gets added named as *Monitoring* tab. Clicking on this tab, user can see a number of services enabled in OpenStack. Through these subtabs, users or administrators can view their health.

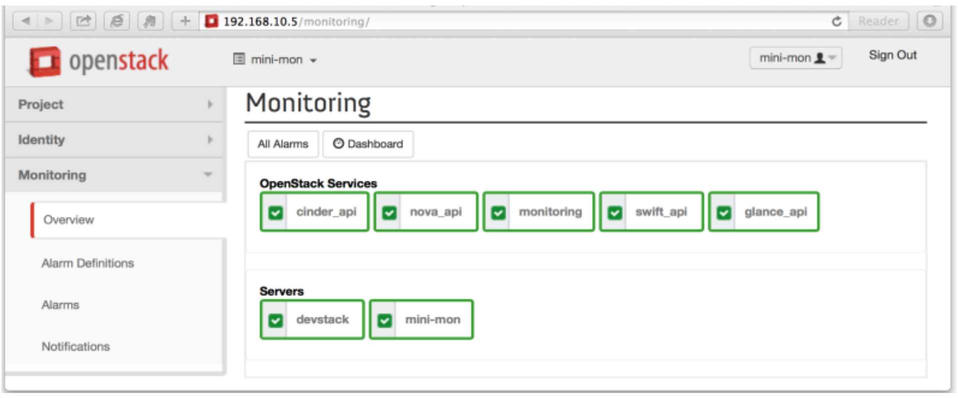


Figure 4.3: Monasca UI integrated with OpenStack dashboard

As shown in the following figure, an opensource graphing tool is used to show the health of services being monitored with Monasca. Each of the graph is specified for a different metric and shows its measurements over a period of time.

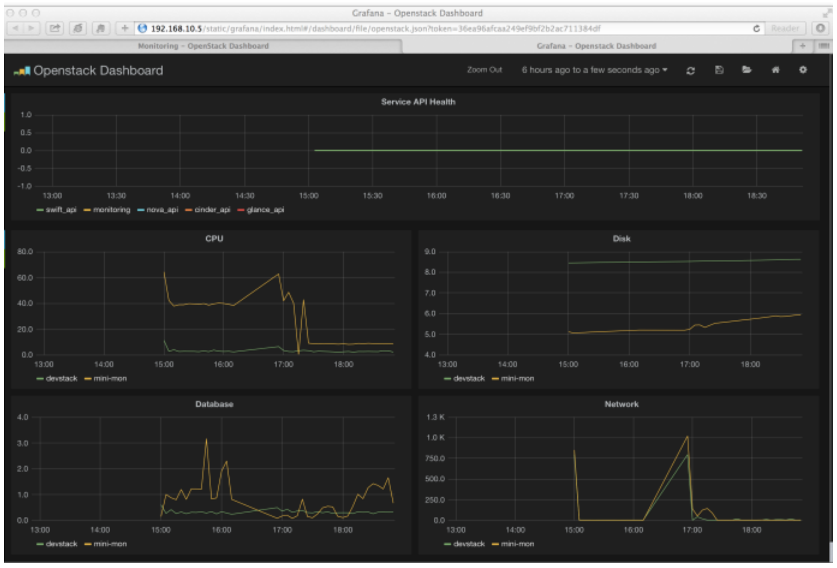


Figure 4.4: Graphs showing health of Monasca

Chapter 5

Problem hunt and finding community mentors

In order to contribute to open source projects, it is necessary to establish yourself in community. OpenStack is one of biggest opensource projects that is also industry oriented. Since we are still students, we don't have much exposure to industry use cases of OpenStack. So, for us to target a specific problem, we needed to know about the current difficulties that OpenStack users and developers are facing. As mentioned earlier, the domains that we were interested in included virtualization, networking and monitoring in OpenStack. We setup our community identities and talked to network admins(especially Neutron developers) about the features that they would like to monitor. We also talked to the companies who are actually using Monasca in production environments asking them about short-comings of Monasca in terms of network monitoring.

After thorough discussions and active contributions from industry people, we identified some problems which are currently not being catered but users would like them to be solved. Out of these problems, we prioritized virtual router monitoring problem. Following are the other idea examples that we encountered

- **Generation of FwaaS reports** firewall as a service is considered an important service in networking and is already up for work. One thing that can be done to enhance its functionality and help network admins in to add report generation feature of the traffic blocked/passed (via the ip tables or by fetching from Neutron logs). This is not yet incorporated. Monasca is still adding support for logging-as-a-service so in order to integrate the FwaaS reporting with Monasca, we thought it'd be better to do it once LaaS is mature enough in Monasca.
- **QoS** Quality of service is already implemented in Neutron. We could integrate it with Tap-as-a-service(an extension of Neutron for port mirroring) to determine if Service Quality is assured or not.

We intend to do these projects in future but currently decided to pursue virtual router monitoring project because it appeared to be unique, the industry people gave very enthusiastic feedback over it and it seemed to be challenging enough for us to dive with full energy in open source development.

We were also able to find community mentors that agreed to help us during this project. These people belong to some famous companies like Hewlett Packard, Time Warner Cable, NEC Technologies and PLUMgrid.

Chapter 6

Design of Virtual Router Plugin

Since our goal is to implement something that works well with the currently established systems, our approach is to design our plugin following the convensions/architecture of already implemented plugins working seamlessly with OpenStack and Monasca. One such plugin is Libvirt[github link] which is used to monitor network traffic per instance.

Our proposed design includes three modules

- Virtual router driver
- Virtual router plugin
- Visualization of data

6.1 Virtual router driver

Monasca collects OpenStack's data via its agent which resides on physical nodes(commonly compute nodes). Each compute node can host multiple virtual routers belonging to different tenants but there's only one router per tenant. Our target is to visualize bandwidth, throughput per tenant. As the network traffic targeted for any virtual machine related to any tenant eventually gets in/out of the network through physical NIC of the compute node so, the crucial part of our project is to devise such a mechanism that would filter out the traffic based on the tenant ids and IP addresses.

We are writing a driver for that purpose that would access the data about routers currently residing on a particular compute node through Neutron database via Neutron API. It will also collect data about which router belongs to which tenant, how many interfaces does a particular router has at the very moment and what are the ip addresses assgined to them. Our driver would get integrated with Neutron's packet forwarding mechanism based on routing table of host. It would then perform calculations based on incoming and outgoing packets targeted for a particular VM residing on a particular subnet attached to a particular interface of router.

6.2 Virtual router plugin

Our router plugin will use the driver discussed above to fetch the calculations related to a particular router and convert this data into metrics understood by Monasca. When monasca-agent asks for the metrics after regular intervals by performing checks on system, our plugin will forward these metrics to it. Monasca agent will eventually post these metrics to message queue via monasca-api, from where the metrics get written to time-series database with monasca-persister. Following are the metrics that we are currently proposing to monitor

1. `net.in_bytes_sec`
2. `net.out_bytes_sec`
3. `net.in_packets_sec`
4. `net.out_packets_sec`
5. `net.in_packets_dropped_sec`
6. `net.out_packets_dropped_sec`

6.3 Visualization of data

The data would be visualized with the help of monasca user-interface that has been integrated with OpenStack's horizon. It uses an open source graph plotting tool called grafana for plotting the measurements of a particular metric after regular intervals. Following the same approach, we would enhance grafana to show us the graphs related to virtual router, with our added metrics, that will include bandwidth and throughput per interface per tenant.

Bibliography

[1] “How to get started with openstack,” 2012.

[2] “Monasca: Monitoring-as-a-service (at-scale),” 2015.

[3] J. Denton, “Neutron networking: Neutron routers and the l3 agent,” 2014.

[4] “Openstack repositries,” 2016.

[5] “Neutron ovs dvr - distributed virtual router,” 2016.

[6] P. Inc., “Saas, paas, iaas?,” 2014.

[7] M. Joyce, “The relationship between endpoints, regions, etc in keystone openstack,” 2013.

[8] P. Biswas, “Authentication and authorization model in openstack,” 2014.

[9] C. B. Roland Hockmuth, Deklan Dieterly, “Monasca: Monitoring as a service (at scale),” 2006-2013.

[10] S. Angaluri, “Providing the best infrastructure-as-a-service solution with openstack,” 2015.