Cloud Resource Monitoring for Facilitating Administration

by

Sumit Kadyan
B.Tech., Rajasthan Technical University 2012

A Mater's Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Cloud Resource Monitoring for Facilitating Administration

by

Sumit Kadyan
B.Tech., Rajasthan Technical University 2012

Supervisory Committee

Dr. Sudhakar Ganti, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Sudhakar Ganti, Supervisor
(Department of Computer Science)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science)

ABSTRACT

This project presents the design and implementation of a web dashboard used for monitoring cloud resources. Cloud computing provides sharing and managing data and performing computations on a shared resource via the Internet. In this dissertation, the cloud that is being monitored by the dashboard is SAVI testbed. The dashboard provides a bird's eye view for the monitoring of SAVI tested resources. The dashboards allows the user to monitor different resources: parameters such as cpu utilization, memory usage, cpu time, virtual cpu and instances. It also allows users to browse through information about different resources and then make a decision on resource migration. This dissertation outlines some of the identified weaknesses and it lists suggestions to further improve the dashboard.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank my supervisor **Dr. Sudhakar Ganti** for his support and mentoring throughout the degree.

Finally, I would like to express a special word of thanks to my friends and family who tirelessly listened to my ideas and offered encouragement when it was most needed.

DEDICATION

To my parents and my mentor and friend, my brother Amit.

Chapter 1

Introduction

Cloud computing is a big word in today's world and it will be even more common in the coming years "Cloud is just a metaphor for the Internet". It is a type of computing that relies on sharing resources, managing data and performing computations with these shared resources. Nowadays companies are using cloud computing to consume computing resources as a utility, like electricity, rather than having to build and maintain computing infrastructures in-house. But the resources available in the cloud have to be constantly monitored and managed in order to determine the load on each resource. Resource information provided by the dashboard would help in the migration of machines across edges, which is the future of this project.

In order to provide the monitoring of these resources, web-based dashboards are considered a useful tool to allow a look into today's cloud-computing platform.

A dashboard is a user interface that organizes and presents information in a way that is easy to read and interpret. It helps the administrator or the user to interact with the important and critical information without having to go through multiple web pages. The dashboard also reduces the complexity and the cost of dealing with a large amount of information. A dashboard typically indicates items that require urgent action at the top of the page, moving to less important statistics at the bottom.

The remainder of the dissertation is organized as follows:

Chapter 2 gives background and related work on different dashboards for the monitoring of virtual machines in a cloud-computing environment.

Chapter 3 presents the technologies used in the development of the dashboard and also the back-end architecture for the dashboard.

Chapter 4 describes the implementation of the SAVI testbed dashboard. It also provides the user with the experience of using the dashboard for monitoring different edges across the SAVI testbed.

Chapter 5 includes the experimental results.

Chapter 6 contains the concluding remarks and provides directions for future work.

1.1 OpenStack

OpenStack [14] is an open-source cloud-computing software platform. NASA and Rackspace [22] originally developed the software. It is simple to implement and is an open and massively scalable Infrastructure as a Service [22]. It provides cloud-computing services over standard hardware that is flexible, compatible and scalable. OpenStack is a set of software tools for building and managing cloud computing platforms for public and private clouds. OpenStack computing uses an SQL database to store and retrieve stateful information. MySQL is the popular database choice in the OpenStack community. Figure 1.1 represents OpenStack architecture with its different components and the interface between each of the components.

Listed below are the different components of the OpenStack:

Nova [18] is the primary computing engine for OpenStack. It is built on a shared nothing, message-based architecture. It is used for deploying and managing large numbers of virtual machines and other instances to handle computing tasks. Nova drivers provide the functionality to communicate with different Hypervisors running on the host operating system to determine their functionality over the Internet. It has components that communicate through the messages. Its components can be distributed, which makes it readily scalable. All the requests related to the resources are processed through Nova and then Nova makes use of its other components to process the request further.

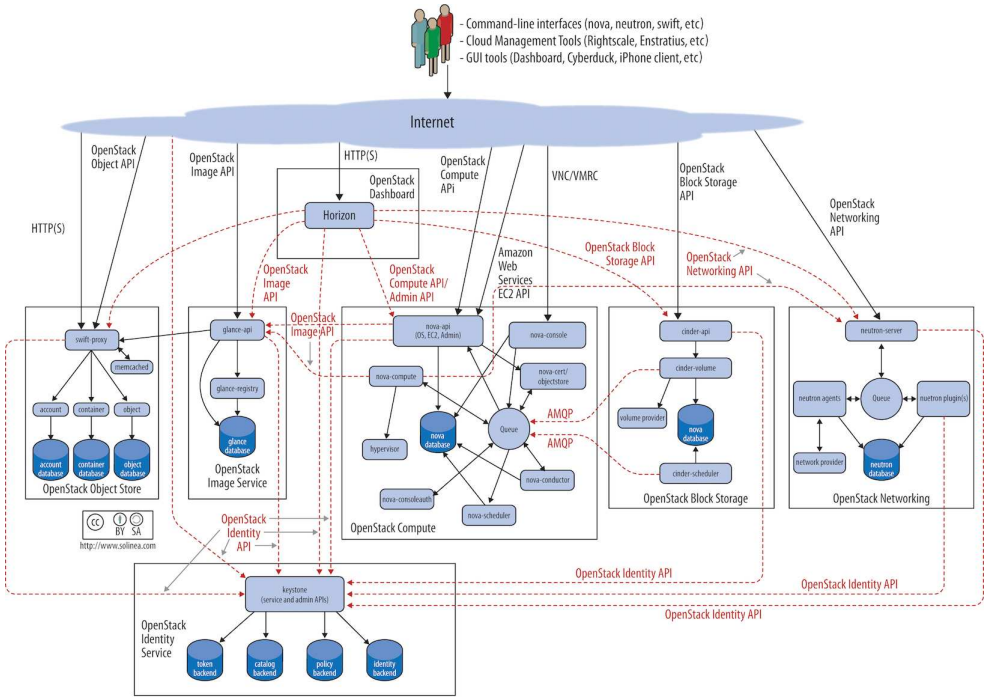


Figure 1.1: OpenStack-Architecture

Swift provides a storage system for objects and files. The referencing of files in Swift is based on the unique identifier referring to a file or piece of information irrespective of where the file is stored. It leads to great benefits such as scaling and having the system take care of backing up the data in case of machine failure or network failure. It is used by Glance to store its images and by Cinder to create a back up for all its virtual machine volumes.

Cinder provides block storage services to the end-users of OpenStack. It is used for providing volumes to virtual machines. It has the feature of snapshots where the complete machine state at a particular moment can be stored and later on used to clone the machine. It is really useful in scenarios in which the data access speed is an important consideration.

Keystone provides the identity service for OpenStack. It is essentially a central list of all the users of the OpenStack cloud, mapped to all of the services provided by the cloud, which they have permission to use. It provides token-based authentication. All the requests that come into Nova are sent to Keystone for authentication. All

components are dependent on Keystone for validation of credentials.

Neutron provides the networking capability for OpenStack. It provides networking as a service between interface devices. It helps to ensure that each component of an OpenStack deployment can communicate with one other quickly and efficiently.

Horizon is the dashboard provided for OpenStack. It is a graphical interface that presents all the information about the different components of OpenStack. It is an end-user interface and provides insight into all the components in a web application interface. It is also helpful to the system administrators, allowing them to monitor the cloud and to manage what is needed.

Glance provides image services to OpenStack. In this case, "images" refers to images (or virtual copies) of hard disks. Glance allows these images to be used as templates when deploying new virtual-machine instances.

Ceilometer provides telemetry services, which allow the cloud to provide billing services to individual user's of the cloud. It also keeps a verifiable count of each users system usage of each of the various components of an OpenStack cloud. It is used for metering and usage reporting.

1.2 About SAVI

SAVI stands for Smart Applications on Virtual Infrastructure, which is an NSERC Strategic Research Network. It is a research and testbed on application platforms. The research goal of the SAVI network is to address the design for the future application platforms that are built on flexible and versatile infrastructure that can readily deploy, maintain and retire the large-scale distributed applications for future applications [10].

National distributed application platform testbed is designed by SAVI for creating and delivering future Internet applications. The SAVI testbed provides flexible, virtualized converged infrastructure for the support of experimental research in application-oriented networking, cloud computing, integrated wireless/optical access networks, and the future Internet architecture.

1.2.1 SAVI testbed

The SAVI testbed is a network of virtualized converged resources clusters. It is used by the SAVI themes to demonstrate control and management of virtual infrastructure. The SAVI testbed provides access to heterogeneous resources such as computing blades, storage, and networking. The SAVI testbed will also support experimentation in applications built on advanced services that provide intelligence through analytics and advanced media processing. Figure 1.2 shows the SAVI testbed architecture.

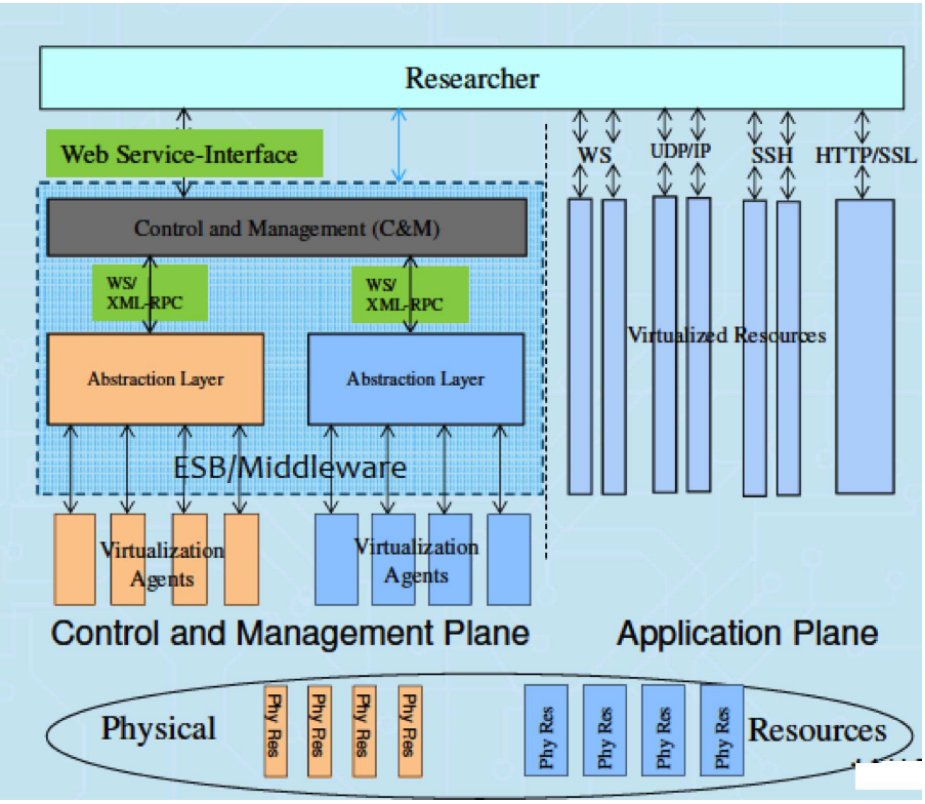


Figure 1.2: SAVI Testbed Architecture

The SAVI testbed consists of the Tenant, Edge and Instances.

Tenant is used to differentiate the experimenters at high level because tenant names are name of the organisation in the SAVI testbed. Experimenters from the same organization are grouped under the same tenant name. Grouping users by name of the organization is to categorize users in the SAVI testbed and restrict users to

their organizations for some specific task. For example, a user from the University of Toronto, which has a tenant ID as "UoT" is legitimated to see information of another tenant.

Edge is defined as a rack location where all the physical hardware is located. When experimenters issue a request for a resource allocation they need to define the Edge name. On submission of demand, the rack is identified and resource allocation takes place.

Instances are resources assigned to SAVI users. The instances are the ones that will be monitored by the dashboard for a particular Edge. The instances can be virtual machines or physical machines.

1.3 Motivation

The motivation behind this project was that OpenStack lacks a resource-monitoring dashboard. OpenStack provides the SAVI testbed users with the dashboard known as "Horizon" but this dashboard is just a high level view of the resources. It does not provide any information about the resource usage. Hence the development of such a web application is vital for the SAVI testbed.

The documentation provided on using a ceilometer for the SAVI testbed is quite limited, which is another motivation to develop this dashboard and provide developers with ceilometer documentation about how to use it. Furthermore, the underlying dissertation explains the project implementation details which can serve as a user guide for using the ceilometer for the SAVI testbed.

Chapter 2

Related Study

Up till now the resource constraints of the SAVI [11] testbed are being monitored by Horizon, which is a dashboard provided by OpenStack [24]. Horizon is an abstract view of the resources and does not provide a detailed view of the resource constraints. So, in order to solve this problem, a SAVI testbed dashboard that can monitor the resource constraint for each edge and provide a better picture of the details of the resource constraint is really helpful.

Another reason behind developing this dashboard is to provide an insight into the resource constraints that would equip administrators with the control to move the resources from one edge to another edge based on factors such as load on a particular edge, and the capacity of that edge. This would greatly reduce the load on edges that are limited in their physical capacity.

2.1 OpenStack

OpenStack makes use of the "Horizon" for the OpenStack dashboard, which is a web interface that helps users manage resources and services. It provides a medium for the users to find information about an Edge and by clicking on their specific resource they can get information about that resource.

2.2 Horizon

OpenStack's dashboard uses Horizon as its underlining foundation, which provides a web interface for OpenStack services including Nova, Swift, Keystone, Glance, Neutron, Cinder, and Heat.

Some of the key features of Horizon are as follows:

- **Core Support:** It supports all the core OpenStack projects.
- **Extensible:** Adding a new component can be done by anyone.
- **Manageable:** It is simple and easily-to-navigate in between the codebase.
- **Consistent:** Interaction and visual paradigms are standardized throughout.
- **Stable:** Provides a reliable API.
- **Usable:** This is one of the key highlights that attract people to Horizon because it provides an amazing user interface which is easy to navigate.

Figure 2.1 shows the OpenStack Dashboard.

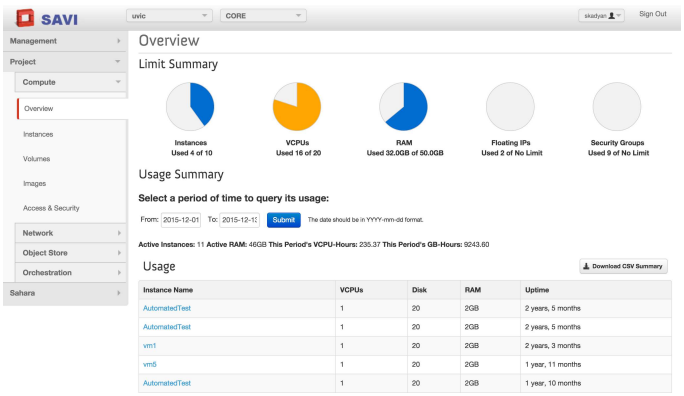


Figure 2.1: Horizon

Figure 2.1 displays an overview of the resources at the UVic core and it allows users logged in with tenant ID to look at other Edges.

The OpenStack dashboard makes use of Horizon and displays resource constraint for complete Edge but it does not provide the same information for a single resource.

This is one of the limitations of the OpenStack dashboard, which underscores the need for a dashboard that would provide resource constraint information specific to a resource.

Figure 2.2 displays a detailed view of a resource in Horizon.

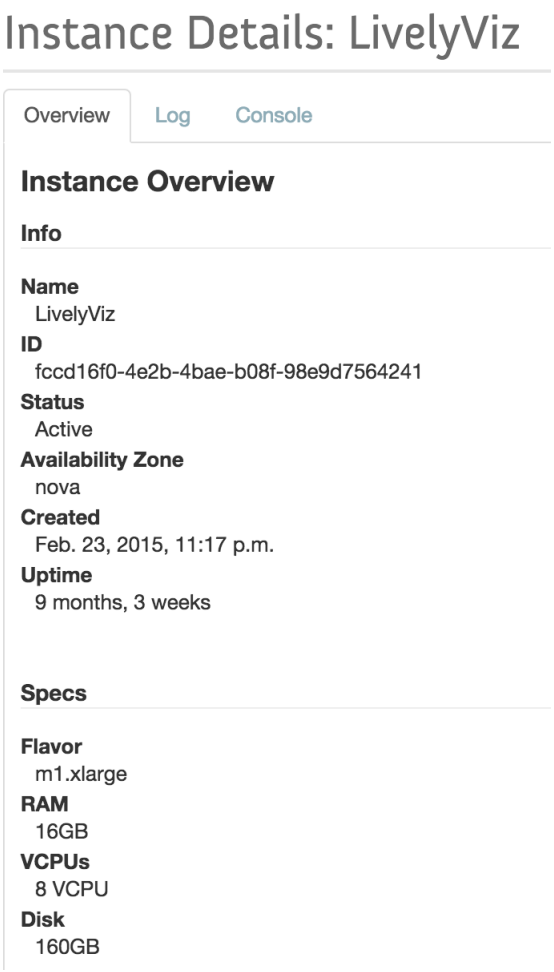


Figure 2.2: Horizon

2.3 Other approaches

In this section we look at some of the different approaches that were taken in building dashboards for the monitoring of cloud services by other researchers.

In [5] the authors designed a framework in the form of a dashboard, which is used to capture all the elements of a comprehensive cloud solution management framework. The dashboard created provides users with the capability to manage the virtual machines individually or in groups. The dashboard comes with features such as the ability of the user to take a snapshot of a single VM or multiple VM or reboot an individual or multiple VM from the dashboard with a single click.

The complete setup was placed on the IBM Smart Cloud. The dashboard used by the authors provided validation of the calls getting routed from Point A to Point B and then validated these calls based on the snapshots provided by the dashboard. Figure 2.3 is snapshot of the dashboard.

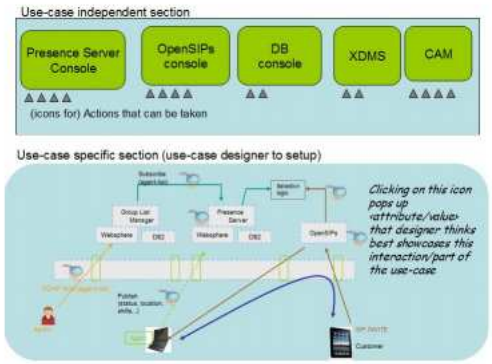


Figure 2.3: Dashboard Design

This dashboard is an easy way to visually verify that the condition specified in the experiment holds true. The dashboard is used for managing and validating a solution. The efficiency of this dashboard was also measured by video-based routing. In contrast to the SAVI testbed dashboard, we are monitoring different resource constraints for a single resource and providing the user with the ability to look at different resources based on a particular Edge.

The authors in [3] designed a dashboard dynamically accessing the sustainability of cloud-computing services. The Cloud sustainability dashboard [3] is used to access the overall impact of the services that are hosted on the cloud. The evaluation model for the dashboard is based on four factors sustainability, economic reasons, ecological factors, and social factors. The dashboard's value is best realized when different users can select their own sustainability preferences for measurement. However this paper does not provide in-depth information about implementation details of the dashboard but presents only the evaluation models based on the factors listed [3].

Cloudlet architecture [8] for dashboards in Cloud proposed some methods for developing dashboards that can be used by the manufacturing sector, which was the target audience for this dashboard. The authors [8] show how the application's presentation layers are now structured into a set of widgets. Each widget has its own graphical representation, which supports a service that can easily be incorporated into a dashboard, thus improving the decision-making by capitalization on human perceptual capabilities. The architecture for the basic design was the MVC (Model View Controller) and the interface followed the RIA (Rich Interface Architecture) presentation design pattern. The dashboard was built using the web 3.0 technologies (HTML5, JQuery, CSS3). Figure 2.4 shows a dashboard for monitoring the cloud services.

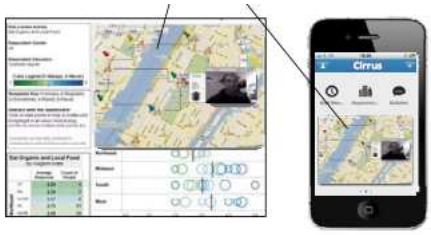


Figure 2.4: Dashboard View

After having gone through all the design architectures presented above for designing a dashboard for Cloud it is evident that they are used for a specific Cloud. An existing OpenStack dashboard does not provide the resource constraint monitoring for a specific resource and is not customizable for resource-specific information. All the above architectures are very well defined in their domain and area of research but building a dashboard for a SAVI testbed specific to a particular Edge has been

missing and is much needed.

The dashboard is built using the web 3.0 technologies [8] and employs the Node.js framework. The reason for using Node.js was that it is easy to scale up the application, which plays a vital role in Cloud services. The dashboard can later be implemented for monitoring multiple Edges.

2.4 Summary

The chapter discussed various approaches towards developing a dashboard for monitoring Cloud resources. All the designs and architecture we looked are domain-oriented and specific to a particular Cloud federation. Although there is a dashboard for the SAVI testbed, we developed a dashboard that provides greater details about individual resource constraints and information related to a particular Edge.

Chapter 3

Design Methodology and Tools

3.1 Ceilometer

The Ceilometer is a component of OpenStack [15], which provides the telemetry services to its users. Telemetry services are wireless transmission and reception of measured quantities for the purpose of remotely monitoring the Cloud computing resources to gather data about them.
The Ceilometer [17] was released with the Havana version of OpenStack. It is used to reliably collect measurements of resource utilization and virtual machines that are deployed on the Cloud. The data is stored in a database and then later for analysis and to trigger actions when defined criteria are met.
The Ceilometer component of OpenStack is used primarily by billing systems to establish a particular point of contact in order to provide them with the customer billing, across all the OpenStack Components.

The Ceilometer is a useful component of OpenStack and helps with some of the following:

- Provides efficient collection of metering data; for example, CPU and network costs.
- Data can be collected either by notifications sent by existing systems or polling the infrastructure.
- Deployers can configure the type of data they want to collect, based on their operating requirements.
- Data collected is made available through the REST API to other users.

3.2 Ceilometer Architecture

Ceilometer [16] architecture is designed in such a way that each of its services can be scaled in order to incorporate more load by adding workers and nodes to the underlining architecture. It provides five services at its core: polling, notification, collection, API, and alarming.

These services work independently from collection and alarming but they can also work as a whole to provide a complete solution.

Figure 3.1 is an image showing the architecture of a Ceilometer.

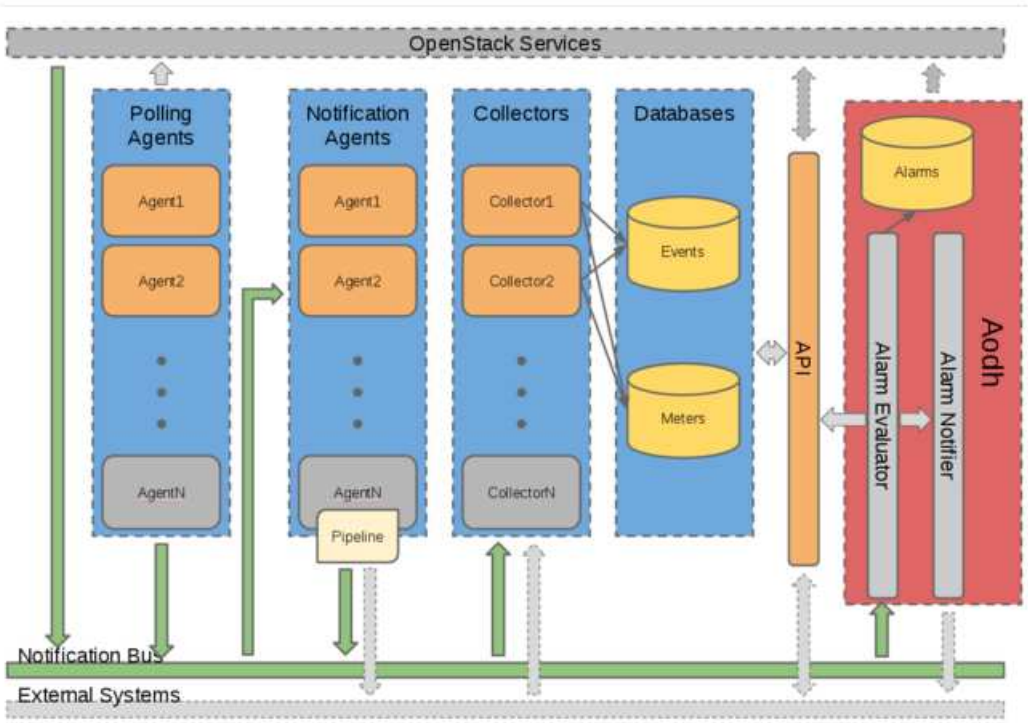


Figure 3.1: Ceilometer Architecture

The Ceilometer's [16] architecture comprises the following five components:

Polling agent: The working of polling agent is a daemon designed to poll services of OpenStack and build meters for those services.

Notification agent: Is a daemon (computer program that runs as a background process) designed to constantly listen to messages that are there on the notification queue and converts those messages into samples and events and then applies pipeline actions to them.

Collector: All the data that is produced by the polling and notification agent are gathered and recorded by this daemon. It can run on one or more central management servers to monitor the message queues.

API: Is a service that provides access to the data from the data store. The collector and API [2] server have access to the data store. It is a type of service to query and view data that is collected by the collector.

Alarms: These are set of rules on which the evaluation and notification is based.

The data in the database is captured in a time-series database to optimise storage and querying. The resource metering used for this is known as "Gnocchi" which is part of the databases.

3.3 Ceilometer Implementation

3.3.1 Data Collection Method

The collectors and agents that gather data from multiple resources do the data collection. Figure 3.2 shows how the polling agent and notification agent create the data collection.

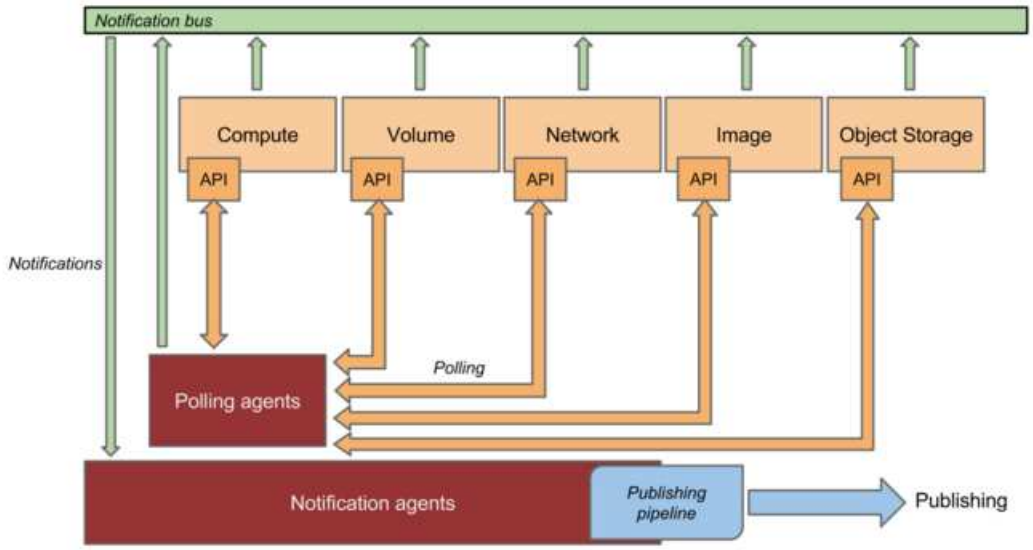


Figure 3.2: Data Collection by Collector and Agents.

The data is collected by two methods in Ceilometer:

Polling Agent

In this method polling is done at a set time interval and then data are collected at each time interval. Polling is done with the help of API or tools to collect information. The polling agent is configured to poll a local Hypervisor or remote API.

Bus Listener Agent

In this method the events generated on the notification bus are taken and then converted in Ceilometer samples. In this method the notification agent monitors the message queues for notifications.

Notification Agent: Listening for data

The notification daemon is the heart of the data collection system which, monitors the message bus for data being provided by Swift, Nova, Glance, Neutron, Cinder, Keystone, and Heat and also takes care of the internal communication. The messages are grabbed by the listener and then redistributed based on the defined topics to the endpoints for them to process them into events and samples. Notifications can be converted into events, which can be filtered based on the event types.

Polling Agent: Asking for data

It queries services for data. It runs on the compute agent and the polling agent handles the polling for computer resources. The polling agent is responsible for querying all the data related to compute resources. For non-compute resources the central agent handles the polling. The frequency of the polling can be controlled by the pipeline configuration.

Processing Data

It takes the data gathered by the agents, manipulates it and then publishes that data using multiple pipelines (set of transformers mutating data points into something that the publishers know how to send to external systems) and using different combination. The notification agent is responsible for handling the processing of the data.

Transforming Data

There are various transformers provided by the Ceilometer to manipulate data in the pipeline. The polling and notification agents gather enormous amounts of data. This data can also be more meaningful if is combined with temporal or historical data to provide more insightful data.

Publishing the Data

A sample of data can be published using any of the four transports:(a) notifier, (b) publisher based on notification which can publish data to a message queue which can be consumed by the collector,(c) UDP (user data gram packet): samples published using UDP packets, and (d) Kafka: published data to a Kafka message queue to be consumed by system that supports Kafka.

Storing the Data

The data collected by the collector can be stored in a database or file. The data is collected before being stored in the database, which is then validated and written in the database.

Accessing Data through API service

The data collected is made available to the developer by the use of the REST API that is offered by the Ceilometer.

3.4 Ceilometer command-line installation and commands

The following section describes how to install OpenStack [4] command-line packages. The installation is described for the installation of the Ceilometer command-line client for Mac OS X. The prerequisite for client installation are:

- Python 2.7 or later;
- Setup tools package which is installed by default on Mac OS X; and
- Pip package: *# easy _ install pip.*

For installing the Ceilometer client, the following pip command is used:
 # pip install python-ceilometerclient

To make sure that the Ceilometer client is installed, the following command is used: *# ceilometer help*
This command also provides all the kinds of commands that are available for Ceilometer. This is like the reference dictionary of Ceilometer for commands.

3.4.1 Ceilometer commands

Meter-List command

Ceilometer [16] is capable of monitoring the OpenStack resources: Compute (Nova), Network (Neutron), Image (Glance), and Storage (Cinder/Swift). A meter is an entity that is attached to each of these resources and used for monitoring purposes. The meters can be of one of three types: gauge (discrete items), cumulative (increasing over time), or delta (changing over time).

The following command can be used to retrieve the complete list of the meter names with their types and units.

 # ceilometer meter-name-list

Name	Type	Unit
cpu	cumulative	ns
cpu_util	gauge	%
disk.ephemeral.size	gauge	GB
disk.read.bytes	cumulative	B
disk.read.requests	cumulative	request
disk.root.size	gauge	GB
disk.write.bytes	cumulative	B
disk.write.requests	cumulative	request
image	gauge	image
image.size	gauge	B
instance	gauge	instance
instance.scheduled	delta	instance
instance:m1.large	gauge	instance
instance:m1.medium	gauge	instance
instance:m1.small	gauge	instance
instance:m1.tiny	gauge	instance
instance:m1.xlarge	gauge	instance
ip.floating	gauge	ip
ip.floating.create	delta	ip
ip.floating.update	delta	ip
memory	gauge	MB
network.incoming.bytes	cumulative	B
network.incoming.packets	cumulative	packet
network.outgoing.bytes	cumulative	B
network.outgoing.packets	cumulative	packet
port	gauge	port
port.create	delta	port
port.update	delta	port
vcpus	gauge	vcpu

Figure 3.3: Meter Name List

From Figure 3.3 we can see with 'the image' meter is related to monitoring the Glance service and the 'networking.incoming.bytes' meter is monitoring the Neutron

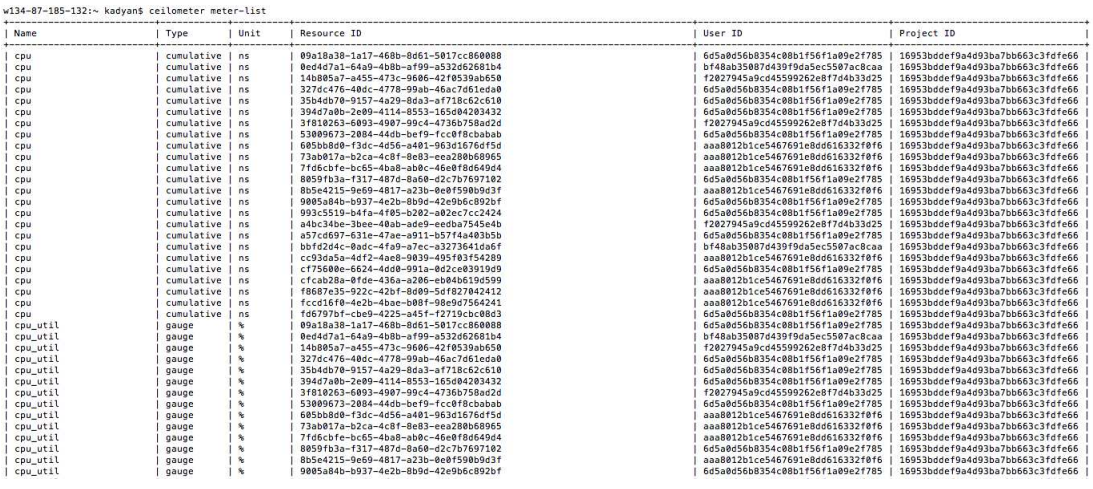
service.

The following command can be used to retrieve more detailed information about the meters with their resource ID and user ID, and provide detailed information about the meters associated with a particular resource ID.

 # ceilometer meter-list

In order to find information about the meters associated with a specific resource ID, the following command provides all the meters associated with a specific resource ID.

 # ceilometer meter



```
w134-87-185-132:~ kadyan$ ceilometer meter-list
+--------------------------+------------+------+------------------------------------------+----------------------------------+----------------------------------+
| Name                     | Type       | Unit | Resource ID                              | User ID                          | Project ID                       |
+--------------------------+------------+------+------------------------------------------+----------------------------------+----------------------------------+
| cpu                      | cumulative | ns   | 09a18a38-1a17-468b-8d61-5017cc860088     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 0ed4d7a1-64a9-4b8b-af99-a532d62681b4     | bf48ab35087d439f9da5ec5507ac8caa | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 14b805a7-a455-473c-9606-42f0539ab650     | f2027945a9cd45599262e8f7d4b33d25 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 327dc476-40dc-4778-99ab-46ac7d61eda0     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 35b4db70-9157-4a29-8da3-af718c62c610     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 394d7a0b-2e09-4114-8553-165d04203432     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 3f810263-6093-4907-99c4-4736b758ad2d     | f2027945a9cd45599262e8f7d4b33d25 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 53009673-2084-44db-bef9-fcc0f8cbabab     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 605bb8d0-f3dc-4d56-a401-963d1676df5d     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 73ab017a-b2ca-4c8f-8e83-eea280b68965     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 7fd6cbfe-bc65-4ba8-ab0c-46e0f8d649d4     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 8059fb3a-f317-487d-8a60-d2c7b7697102     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 8b5e4215-9e69-4817-a23b-0e0f590b9d3f     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 9005a84b-b937-4e2b-8b9d-42e9b6c892bf     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | 993c5519-b4fa-4f05-b202-a02ec7cc2424     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | a4bc34be-3bee-40ab-ade9-eedba7545e4b     | f2027945a9cd45599262e8f7d4b33d25 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | a57cd697-631e-47ae-a911-b57f4a403b5b     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | bbfd2d4c-0adc-4fa9-a7ec-a327364ldab6f    | bf48ab35087d439f9da5ec5507ac8caa | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | cc93da5a-4df2-4ae8-9039-495f03f54209     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | cf75600e-6624-4dd0-991a-0d2ce03919d9     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | cfcab28a-0fde-436a-a206-eb04b619d599     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | f8687e35-922c-42bf-8d09-5df827042412     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | fccd16f0-4e2b-4bae-b08f-98e9d7564241     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu                      | cumulative | ns   | fd6797bf-cbe9-4225-a45f-f2719cbc08d3     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 09a18a38-1a17-468b-8d61-5017cc860088     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 0ed4d7a1-64a9-4b8b-af99-a532d62681b4     | bf48ab35087d439f9da5ec5507ac8caa | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 14b805a7-a455-473c-9606-42f0539ab650     | f2027945a9cd45599262e8f7d4b33d25 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 327dc476-40dc-4778-99ab-46ac7d61eda0     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 35b4db70-9157-4a29-8da3-af718c62c610     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 394d7a0b-2e09-4114-8553-165d04203432     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 3f810263-6093-4907-99c4-4736b758ad2d     | f2027945a9cd45599262e8f7d4b33d25 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 53009673-2084-44db-bef9-fcc0f8cbabab     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 605bb8d0-f3dc-4d56-a401-963d1676df5d     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 73ab017a-b2ca-4c8f-8e83-eea280b68965     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 7fd6cbfe-bc65-4ba8-ab0c-46e0f8d649d4     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 8059fb3a-f317-487d-8a60-d2c7b7697102     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 8b5e4215-9e69-4817-a23b-0e0f590b9d3f     | aaa8012b1ce5467691e8dd616332f0f6 | 16953bddef9a4d93ba7bb663c3fdfe66 |
| cpu_util                 | gauge      | %    | 9005a84b-b937-4e2b-8b9d-42e9b6c892bf     | 6d5a0d56b8354c08b1f56f1a09e2f785 | 16953bddef9a4d93ba7bb663c3fdfe66 |
```

Figure 3.4: Meter List

From Figure 3.5 we can see the meter associated with different resource IDs and the meter type.

Sample list command

This command is used to display the individual data points for a particular meter name. In order to get the results for the specific resource meter you provide the meter name in the arguments. If you want to get results associated with a specific resource ID, it can be achieved by the following command shown in Figure 3.5. To query based on a specific resource ID, use the query flag as shown in Figure 3.5.

Statistics list command

This command is used to restrict the query to a sample for a particular instance that occurred in a specified time-frame window. This can be achieved by specifying the

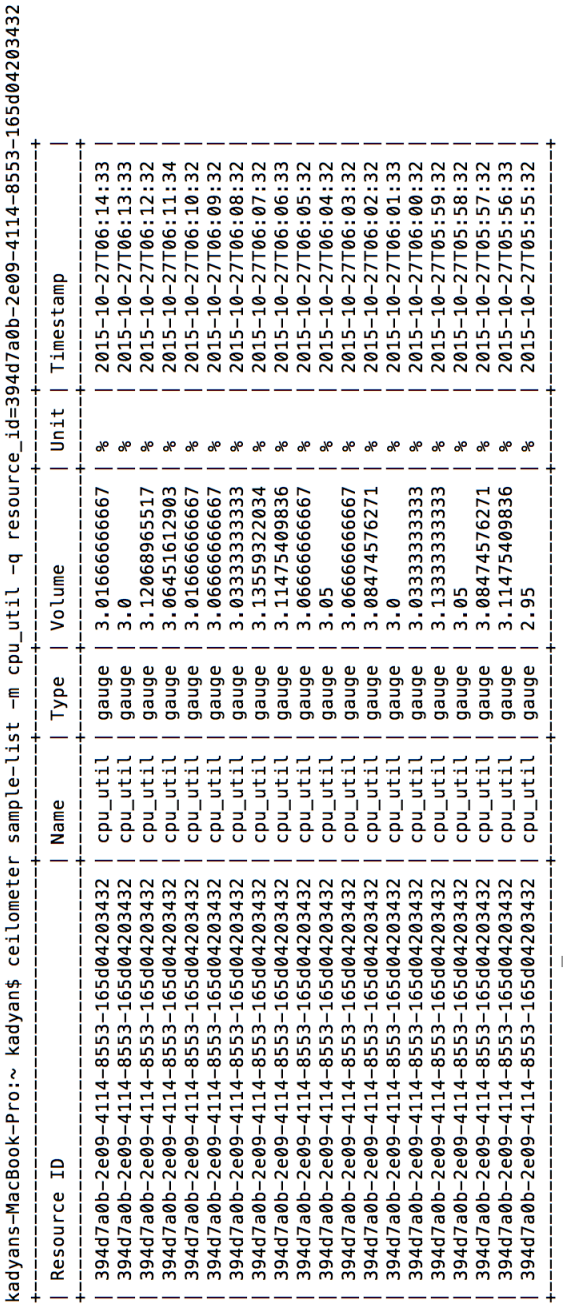
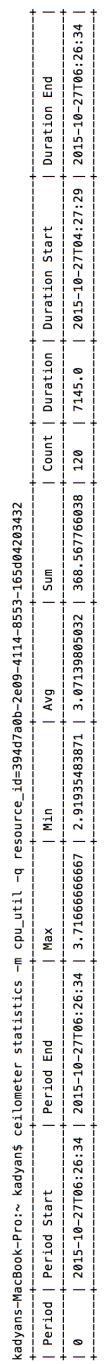


Figure 3.5: Sample List

start and end time stamp. These bounded durations are further divided into time slices. Figure 3.6 shows a statistics command for a given resource ID



```
kadyans-MacBook-Pro:~ kadyan$ ceilometer statistics -m cpu_util -q resource_id=394d7a0b-2e09-4114-8553-165d04203432
+--------+---------------------+---------------------+---------------+---------------+---------------+-------+---------------+---------------------+---------------------+
| Period | Period Start        | Period End          | Max           | Min           | Avg           | Sum           | Count | Duration      | Duration Start      | Duration End        |
+--------+---------------------+---------------------+---------------+---------------+---------------+---------------+-------+---------------+---------------------+---------------------+
| 0      | 2015-10-27T06:26:34 | 2015-10-27T06:26:34 | 3.71666666667 | 2.91935483871 | 3.07139805032 | 368.567766038 | 120   | 7145.0        | 2015-10-27T04:27:29 | 2015-10-27T06:26:34 |
+--------+---------------------+---------------------+---------------+---------------+---------------+---------------+-------+---------------+---------------------+---------------------+
```

Figure 3.6: Statistics of a Specific Resource ID

Alarms command

An alarm [19] that contains a set of rules used for monitoring the resources. An alarm is triggered when one of the services of OpenStack crosses a particular threshold. Alarms have three states:

- **OK:** When the received statistic of interest does not cross the threshold specified by the alarm rules.
- **Alarm:**It is a kind of alarm that is raised when the received statistics cross the given threshold.
- **Insufficient data:**It is triggered when the given statistics are not received from the services.

Chapter 4

SAVI Testbed Dashboard Implementation

The SAVI testbed dashboard is a web application that provides access to information related to resource constraints for a particular Edge. It is an easy-to- use application with different figures displaying the resource constraints to the users to provide a complete view of that Edge or Core to the authorized user. The SAVI testbed dashboard is designed using some of the modern-day technologies. These technologies help achieve concurrency and scalability [13].

This chapter covers the design and implementation of the SAVI testbed dashboard. Readers who are not familiar with OpenStack [12] and Ceilometer API are encouraged to review the concepts and terminology discussed in Chapter 3.
The implementation is divided into four phases, outlined below; it covers each phase in detail with plenty of code snippets plus figures to ease the learning process.

1. Understanding SAVI testbed dashboard architecture.
2. Technologies and tools used in the SAVI testbed dashboard.
3. Ceilometer API used for fetching data for SAVI testbed dashboard.
4. SAVI testbed dashboard user interface.

The remainder of the chapter discusses each of the above stages in detail.

4.1 SAVI Testbed Architecture

The SAVI testbed dashboard provides monitoring of resources based on an Edge or Core and displays the resource constraint for the same. The high-level architecture of fetching data using the Ceilometer API for the dashboard is shown in Figure 4.1 and consists of the following components:

- User interface known as the "SAVI testbed dashboard".
- Node.js, which is hosting the server.
- OpenStack on which the SAVI testbed is hosted.
- Web service: Ceilometer for performing querying of data, which is part of OpenStack.

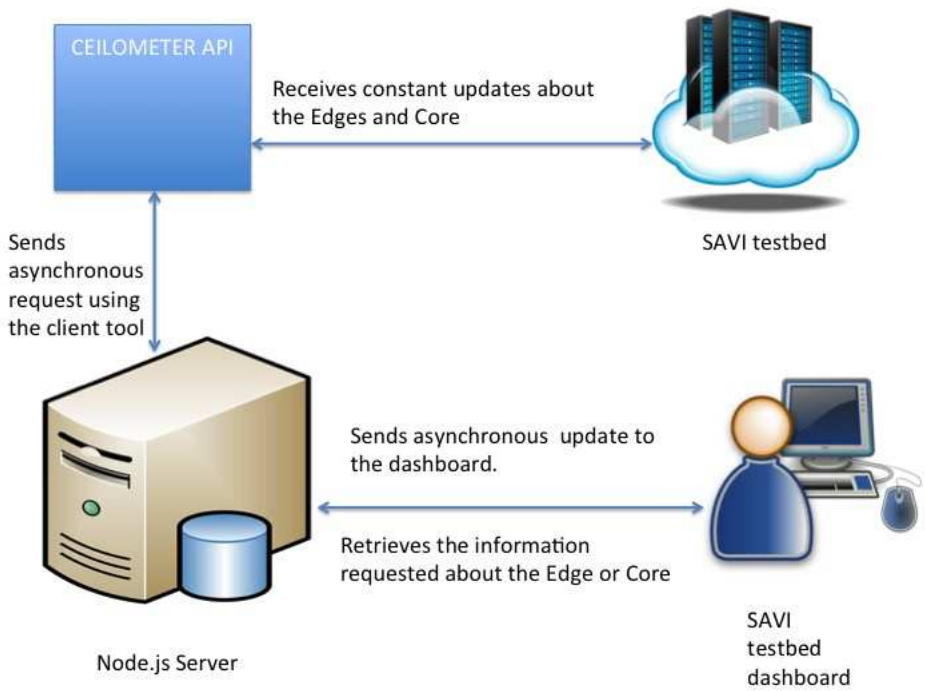


Figure 4.1: High Level Architecture of SAVI Testbed Dashboard

1. Node.js is at the heart of the high level architecture. The server is hosted on Node. This is the central point of data flow between the server and user interface.

2. The user interface known as the SAVI testbed dashboard displays all the information related to resource constraint.

3. SAVI testbed, setup over OpenStack, is the Cloud whose resources are being monitored by the dashboard.

4. OpenStack provides telemetry services and the Ceilometer provides that service to the SAVI testbed for monitoring the resource constraints; therefore the Ceilometer-client tool is used for querying information about resources.

4.2 Technologies and Tools

The SAVI testbed dashboard uses various technologies and tools to provide a scalable, optimised dashboard for resource monitoring.

4.2.1 Node.js

Node.js is a platform built on Chrome's JavaScript run time for building scalable, fast, efficient network applications. The engine is known as the "V8" engine. Node.js is mostly implemented in C and C++ with a primary focus on performance and low memory consumption [25].
Node.js uses the event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

The server for the SAVI testbed dashboard is hosted using Node.js. Node.js is a server-side java-script environment, referred to as Node [25]. The Node.js version, used for developing the SAVI testbed dashboard, is v0.12.7.

Node.js contains a built-in library to allow applications to act as a server without the software such as Apache HTTP server or IIS. Installing packages in the Node is done with the help of NPM (node package manager). It is useful for installing Node.js

programs from the NPM registry.
The SAVI testbed dashboard is built using the Node.js [9] because of all the advantages listed above. In addition to the main concern when monitoring data related to an Edge, the virtual machines can increase rapidly. The dashboard is robust even for large-scale Edge monitoring when the resources being monitored could be from a few hundreds to thousands.

The framework used by the SAVI testbed project is express. It provides the functionality of routing (refers to the definition of end points URLs to an application and how it responds to client requests) for Node.js. A Node package installer is used to install the express framework.
All the incoming requests from the dashboard requesting data are taken care of by the express framework. The requests are routed through to the desired path and, based on the type of request, a response is sent to the client.

4.2.2 Knockout

Knockout [21] is a JavaScript library that helps to create rich, responsive displays and editor-user interfaces with a clean underlying data model. What KO (knockout) achieves is that whenever some parts of the user interface change or are updated, which in the case of SAVI testbed dashboard happens all the time, KO helps in it. It provides a mechanism of two-way binding, which means that any changes made in the user interface are reflected back in the data model and vice versa.

Some of the key features of KO are that it provides elegant dependency tracking. It automatically updates the right part of the UI whenever any changes in the data models take place; it provides great browser support; it can be added on top of your existing web application; and it is a pure JavaScript library.

Code snippet 4.2 shows the usage of the Knockout in the SAVI testbed dashboard project.

```
var serverDataModel = function () {
    this.message = ko.observable("This represents the server data.....");
    shouter.subscribe(function(newValue) {
        this.message(newValue);
    }, this, "messageToPublish");
};
var resourceListModel = function(resourceList) {
    resourceListOptions = ko.observableArray();
    var resourceListArray = Object.keys(resourceList).map(function (key) {return resourceList[key]});
    for(var resourceIndex in resourceListArray) {
        resourceListOptions.push(resourceListArray[resourceIndex]);
    }
    resourceData = ko.observable();
    resourceData.subscribe(function(newValue) {
        shouter.notifySubscribers(newValue, "messageToPublish");
        console.log('newValue');
    });
};
var getResourceIDData = function (resourceName) {
    console.log('Selected Value ' + resourceName);
};
$.get('/resource', function(data) {
    ko.applyBindings(new masterVM(data));
});
var masterVM = function(data) {
    resourceListModel =  new resourceListModel(data);
    serverDataModel = new serverDataModel();
    d3Model = new d3Model();
};
```

Figure 4.2: Knockout

The code snippet consists of three models that are used for dashboard data binding. The master-VM is master data model that consists of these sub models. Each sub-model is independent and provides scalability just adding different models to the main model.
The resourceListModel consists of a list of all the resources that are used as selection criteria to view the constraints of a specific resource.
The serverDataModel consist of the resource constraint information for each resource.
D3Model consists of the information related to binding this data to various D3 figures.

4.3 Celiometer used by the SAVI testbed dashboard for fetching data

The Ceilometer tool provides telemetry services for OpenStack [12]. To query data related to resource constraints, different commands need to be executed. For a complete explanation on the Ceilometer commands and detailed understanding of how the Ceilometer works, please refer to Chapter 3.

The mechanism related to the fetching of data for resource constraint requires an authentication token for each request. The authentication is used by Keystone to validate whether the user is a SAVI testbed authorized user.

Code snippet 4.3 is the script that is executed with each request for the authentication.

```
export OS_USERNAME=skadyan
export OS_PASSWORD=****
export OS_REGION_NAME=EDGE-VC-1
export OS_TENANT_NAME=uvic
export OS_AUTH_URL=http://iam.savitestbed.ca:5000/v2.0/
```

Figure 4.3: Configuration file for SAVI testbed

From code snippet 4.3, it can be seen that the user provides a user name and password. The tenant name is University of Victoria and in the region name you

specify the name of the Edge whose resources constraints you would like to monitor for each request you make.

However, there is a disadvantage to embedding the password in the script because this bash file is not encrypted; hence, each time you make a request the bash file is sent over the network, which can easily be targeted by any malicious user.

After the user is authenticated by Keystone by providing the credentials with each request, the Nova list command is used to get all the resource _ ID for the specific Edge, in this case the University of Victoria Edge.

To fetch data related to all the resource constraints, we need to fetch all the resource _ ID which can then be given to resource-specific script to get the resource constraints for that specific resource. From code snippet 4.4, the credentials for the request are provided in the script and then the Nova list command is executed. In the script when we execute the Nova list we only request the first, second, and third columns.

```
export OS_USERNAME=skadyan
export OS_PASSWORD=****
export OS_REGION_NAME=EDGE-VC-1
export OS_TENANT_NAME=uvic
export OS_C=http://iam.savitestbed.ca:8777
export OS_AUTH_URL=http://iam.savitestbed.ca:5000/v2.0/
nova list|awk 'NR>3'|cut -d'|' -f1 -f2 -f3|sed \$d
```

Figure 4.4: Nova List Executed to Fetch all the Resource_ ID

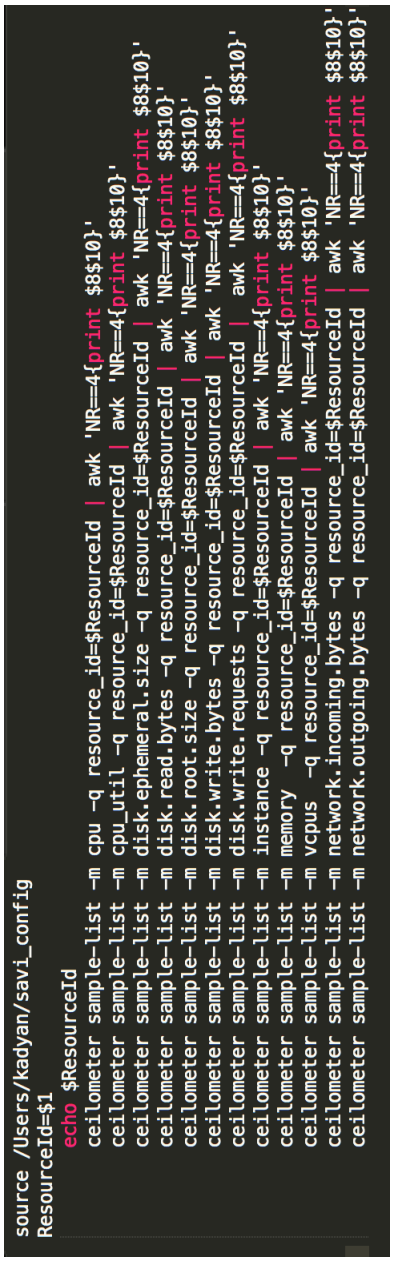
The code snippet 4.5 is the output we get after executing the above bash script. In the code snippet the first column displays the resource _ ID and third column displays the name of the resource _ ID.

Once we have gathered all the resource _ID for a specific Edge by executing the Nova list bash script, we provide each resource _ ID to the resource-specific script to fetch all the resource constraints demonstrated in the previous chapter.

```
kadyans-MacBook-Pro:~ kadyan$ ./resourcename.sh
| 394d7a0b-2e09-4114-8553-165d04203432 | FDAServiceTester
| cc93da5a-4df2-4ae8-9039-495f03f54289 | GreenPower
| 53009673-2084-44db-bef9-fcc0f8cbabab | Kaleidoscope Gstreamer Services
| cf75600e-6624-4dd0-991a-0d2ce03919d9 | Kaleidoscope Locality Service
| a57cd697-631e-47ae-a911-b57f4a403b5b | Kaleidoscope Media Service II
| fd6797bf-cbe9-4225-a45f-f2719cbc08d3 | Kaleidoscope Registry Service
| 327dc476-40dc-4778-99ab-46ac7d61eda0 | Kaleidoscope SnapChat Billboard
| 993c5519-b4fa-4f05-b202-a02ec7cc2424 | Kaleidoscope Web Application
| 9005a84b-b937-4e2b-8b9d-42e9b6c892bf | Kaleidoscope-PubSubService
| fccd16f0-4e2b-4bae-b08f-98e9d7564241 | LivelyViz
| 3f810263-6093-4907-99c4-4736b758ad2d | SDN-Waseem
| 8059fb3a-f317-487d-8a60-d2c7b7697102 | TestKubernetes
| 8b5e4215-9e69-4817-a23b-0e0f590b9d3f | andikaleidoscope
| 35b4db70-9157-4a29-8da3-af718c62c610 | yakkit demo
```

Figure 4.5: ResourceName

Code snippet 4.6 contains Ceilometer commands that are executed in bash script with the credentials provided prior to executing the script. The Ceilometer commands mentioned in this bash script have been explained in detail in Chapter 3. This bash script provides all the resource constraints that are to be monitored. When executed each command returns results related to that specific resource _ id.



```
source /Users/kadyan/savi_config
ResourceId=$1
        echo $ResourceId
        ceilometer sample-list -m cpu -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m cpu_util -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m disk.ephemeral.size -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m disk.read.bytes -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m disk.root.size -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m disk.write.bytes -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m disk.write.requests -q resource_id=$ResourceId |  awk 'NR==4{print $8$10}'
        ceilometer sample-list -m instance -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m memory  -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m vcpus  -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m network.incoming.bytes -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
        ceilometer sample-list -m network.outgoing.bytes -q resource_id=$ResourceId | awk 'NR==4{print $8$10}'
```

Figure 4.6: Ceilometer Commands Executed using a Bash Script

Code snippet 4.7 shows the resource constraint for a single resource _ id. The resource constraint retrieved for each resource is the meter list that the Ceilometer API returns when queried for data specific to that resource.

```
kadyans-MacBook-Pro:~ kadyan$ ./resourcespecific.sh 394d7a0b-2e09-4114-8553-165d04203432
394d7a0b-2e09-4114-8553-165d04203432
1.7348047e+14ns
3.065573770049%
0.0GB
324574720.0B
20.0GB
10426356736.0B
426778.0request
1.0instance
2048.0MB
1.0vcpu
```

Figure 4.7: List of Resources Using SAGEFed

To execute all the scripts in Node asynchronously and publish all the data at a url (uniform resource locator) it is done by the help of child process which is useful in executing bash script in Node.

The following code snippet 4.8 is an illustration of that process

```
var run_cmd=function(cmd, args, callBack ) {
    var spawn = require('child_process').spawn;
    var child = spawn(cmd, args);
    var resp = "";
    var resourceIDArray = [];

    child.stdout.on('data', function (buffer) {
        resp += buffer.toString()
    });
    child.stdout.on('end', function(error) {
        var lines=resp.split('\n');
        for(var resourceID in lines) {
            resourceIDArray.push(lines[resourceID]);
        }
        callBack (resourceIDArray);
    });
}
function resource_Name(){
    run_cmd("/Users/kadyan/resourcename.sh",[], function (resourceIDArray) {
        for (var resourceID in (resourceIDArray))
        {
            var dataFiltered=resourceIDArray[resourceID].replace(/[|]\s/gm, "");
            var dataSplit = dataFiltered.split('\n');
            if(dataSplit[0]) {
                resourceName[dataSplit[0].split(" ")[0]] = dataSplit[0].split(" ")[1] + '_' + dataSplit[0].split(" ")[0].split('-')[0];
                resourceNameInverse[ dataSplit[0].split(" ")[1] + '_' + dataSplit[0].split(" ")[0].split('-')[0]] = dataSplit[0].split(" ")[0];
                //resourceNameInverse[dataSplit[0].split(/\s(.+)?/)[1]] = dataSplit[0].split(" ")[0];
                if (dataSplit[config.RESOURCE_ID] === ' ') {
                    return;
                }
            }
        }
    });
}
```

Figure 4.8: ChildProcess

4.4 SAVI Testbed dashboard

This section describes the flow of information from the backend, i.e, Node to the dashboard that is the user interface. In the previous section we talked about how data were fetched using the script and querying the Ceilometer API [17]. This section talks more in detail about the user interface and different ways in which resource constraints are being monitored.

The server is hosted using the Node.js, which can be setup on any of the VMs (virtual machines) to access the resource constraint for all the virtual machines in the same Edge.

Figure 4.9 provides a view of the dashboard that monitors the different resource constraints and displays them for that resource.

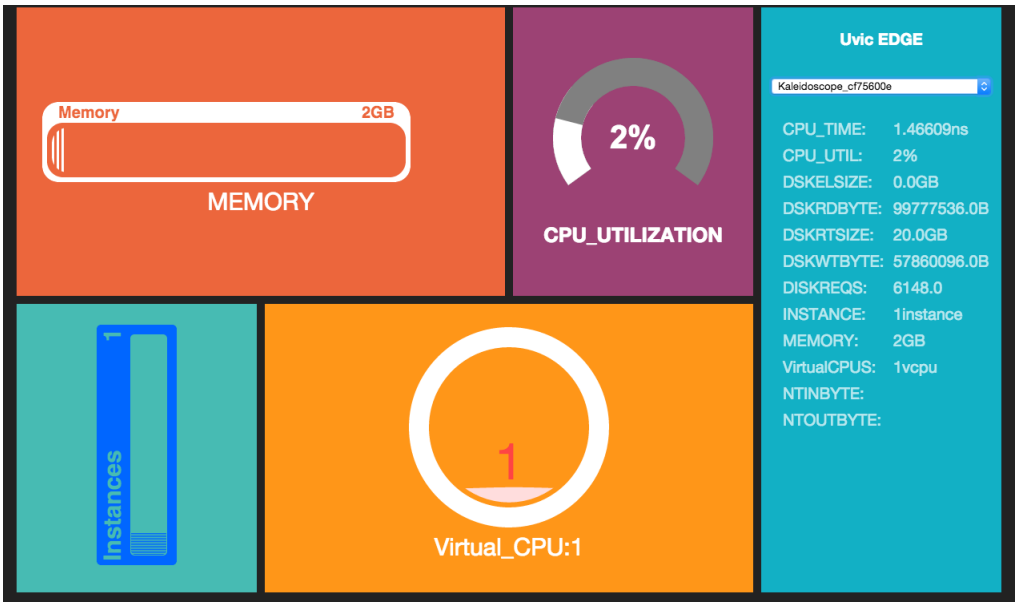


Figure 4.9: SAVI Testbed Dashboard

Figure 4.9 shows the dashboard for the monitoring of SAVI testbed resource constraints. The dashboard is divided into five widgets (a widget is an auxiliary application, which means that it occupies a portion of the web page and provides information in a useful manner to the user). Each widget represents resource-specific information with the help of some figures. The five widgets are as follows:

- **Memory:** Displays the memory allocated for that resource. The units used to represent the memory are in GB [Gigabytes].
- **CPU Utilization:** Displays the CPU utilization for that particular resource. There can be alarms set-up using the Ceilometer commands to keep a check on the CPU utilization. The CPU utilization is represented in percentage.
- **Instances:** Displays the number of instances for that resource. The information is represented using a rectangular vertical bar, which is done using D3(data driven document).
- **Virtual CPU:** Displays the number of virtual CPU associated with a particular resource. The number of virtual CPU are represented by a figure drawn using d3 (data driven document) and based on the number of virtual CPU, the white color fills in the circle.
- **Miscellaneous:** Displays the name of the Edge that is being monitored and all the resource constraints for that particular resource selected from the drop-down menu. The resource constraints being displayed in the dashboard are the meter information extracted from the Ceilometer API.

Now to view information related to a different resource name. The resource name can be selected from the drop-down menu and the dashboard is updated to show the selected resource constraints. Figure 4.10 shows the drop- down menu with different resource names; upon selecting any one of the resource names, its constraints are displayed.

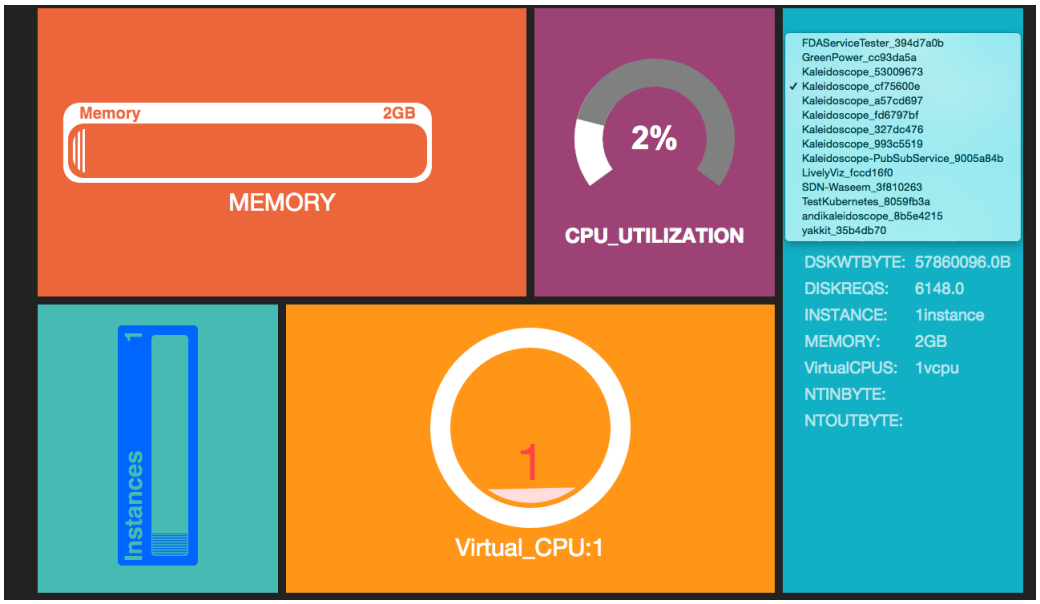


Figure 4.10: Specific Resource

4.4.1 Data Publishing

This section descries how the dashboard requests information by making use of the GET to retrieve all the resource names. The following code snippet 4.11 shows how the request for the resource URL is made which contains all the resource names.

```
$.get('/resource', function(data) {
    ko.applyBindings(new masterVM(data));
});
var masterVM = function(data) {
    resourceListModel =  new resourceListModel(data);
    serverDataModel = new serverDataModel();
    d3Model = new d3Model();
};
```

Figure 4.11: ajaxresourcename

The request comes in to the requesting URL, which contains key/value pairs of the resource name to be mapped to resource id. When the request is made, all the resource names are retrieved and are displayed in a specific format that is the name of the resource hashed with the first segment of the resource ID. Code snippet 4.12 shows the URL containing all the resource names with their key/value pair.

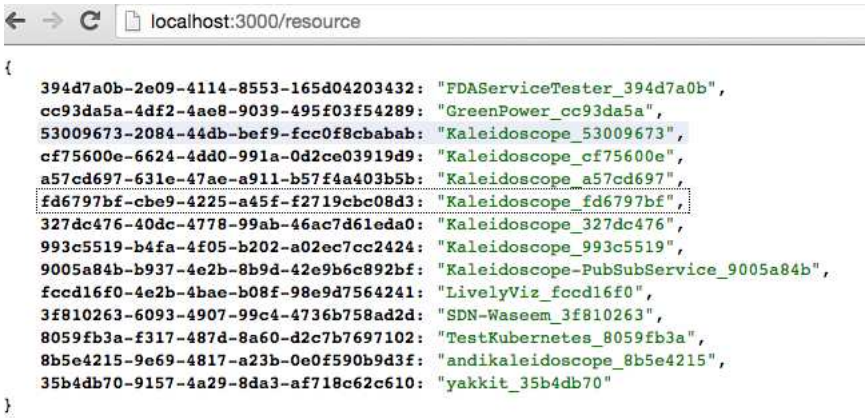


Figure 4.12: Resourceurl

The request comes to the server (Node) using the express routing framework and based on the URL requested, a response is sent back to the client.

The following code snippet 4.13 shows how a GET request handler is requesting data and the response object with data is sent for each corresponding incoming GET request. So, when the request is routed to the designated URL mentioned in the GET request the server responds with the information from the requested URL.

```
app.get('/serverdata', function(req, res) {
    res.json(serverData);
});
app.get('/serverdata/:id', function(req, res) {
    res.json(serverData[req.params.id]);
});
app.get('/resource',function(req,res){
    res.json(resourceName);
});
app.get('/resourceNameInverse',function(req,res){
    // parses the request url
    var theUrl = url.parse( req.url );
    // gets the query part of the URL and parses it creating an object
    var resourceName = theUrl.query;
    var ResourceID=resourceNameInverse[resourceName];

    var serverSpecificData=serverData[ResourceID];
    res.send(serverSpecificData);
    console.log(serverSpecificData);
    //res.json(resourceNameInverse);
});
app.get('/getServers', function(req, res) {
    run_cmd("source savi_config",[],function(text) {
        console.log(text)
    });
});
```

Figure 4.13: Nodeurl

4.4.2 AJAX

AJAX (asynchronous JavaScript and XML) [20] is used to create the asynchronous web application used on the client side. AJAX [6] allows the web application to send and retrieve data from a server (which in this project is Node) asynchronously without interfering with the display and behavior of the existing page. Put Simply, it allows the developer on the client side to update a section or portion of the web page without having to interfere with the other portion of the web page. These AJAX [23] calls can run in the background without interfering with the other functionality of the web page.

We talked about AJAX at the beginning of this section to explain AJAX to readers. It has been used for the dashboard to fetch data from the server.

Code snippet 4.14 shows an AJAX request being sent to a URL that is listed in the code snippet.Along with the AJAX call we are passing in the server name(resource name) for which we are requesting the resource constraint. AJAX calls can have different formats. The type can be any one of the two types GET or POST. The URL from which data is to be retrieved is specified. After the successful completion of the request the success function is executed. The code snippet 4.14 shows how the dashboard retrieves information about all the resource constraints once their resource name is provided.

```
shouter.subscribe(function(serverName) {
    $.ajax({
        type: 'GET',
        data: serverName,
        contentType: "application/json",
        dataType:'json',
        url:"http://localhost:3000/resourceNameInverse",
        success: function(data) {
            updateServerdata(data);
            canvas(data);
            barstacker(data);
            liquidFillGauge(data);
            console.log(data);
        },
        error: function(error) {
            console.log("some error in fetching the notifications");
        }
    });
    console.log('D3 Model Refreshed');
}, this, "messageToPublish");
};
```

Figure 4.14: AJAX

Chapter 5

Evaluation

It is important to evaluate the SAVI testbed dashboard, as it is the only dashboard of its kind being used for monitoring resources. Although there is a dashboard provided by OpenStack known as "Horizon", it does not provide detailed information about the resource constraint. The factors against which the dashboard is evaluated are defined in the evaluation criteria, as follows.

5.1 Evaluation Criteria

Usability: It is defined as ease of use and learn ability of something designed and made by humans. It is something that is available or convenient for use.
In this dissertation we are evaluating the dashboard. Usability describes the quality of user experience across websites, software, products, and environments. The dashboard is evaluated against this parameter to determine its usability.

Scalability: Scalability is the capability of a system, network, or process to handle a growing amount of work or its potential to be enlarged in order to accommodate that growth. It is defined as the capacity of the system to handle a growing amount of work or be enlarged accommodate the growth.
The SAVI testbed dashboard is very scalable and the latter section proves that. As the number of users increase, the dashboard does not slow down in terms of responsiveness.

Documentation: The official documentation on using OpenStack Ceilometer for the SAVI testbed is limited.
For novice developers of the SAVI testbed who find it difficult to use the Ceilometer client tool for the testbed, this paper would be of great help.Chapter 3 of this report talks in detail abut the Ceilometer commands and its installation. Then it discusses its usage in the SAVI testbed.

Performance:

The performance of the dashboard is evaluated based on the response time for a given set of resource constraints being monitored. However there is initial loading time for fetching data so the dashboard takes 40 seconds to load initially. This happens when we are monitoring too many resource constraints. But after a certain number of resource constraints this time becomes fixed. In order to correct this fine-tuning of the database to handle so many queries and respond to them more quickly in less time.

Security: Security is the primary concern of any dashboard monitoring application because you do not want non-members of the SAVI testbed Cloud to be able to monitor the data Cloud. Each time a query is made to the database, OpenStack needs an authentication token that is evaluated by Keystone. The script that contains the user name and password credentials are in a bash file which is not encrypted and sent with each request. This kind of process is vulnerable to attack. This section requires addressing by the SAVI federation to make it more secure.

Refresh interval:

It is the time interval defined between the previous data that is being rendered by the dashboard and updated data in the dashboard. The refresh interval is a time interval designed to solve some of the constraints that are on the existing data-fetching mechanism.

The information for the dashboard is being polled every minute. This time period was chosen because in order to retrieve data for one single resource the minimum time taken from the query to the results is 40 seconds. Therefore the polling time was kept more than 40 seconds otherwise there would be too many requests being made to the database even before the results from the previous query were returned, thus creating a deadlock.

5.2 Experiment

5.2.1 Experiment 1: Response Time

The experiment was performed to determine the response time by increasing the resources from one onwards and the response time to fetch all resource constraints was noted. This was done repeatedly to test the scalability and to determine how the time increased as the resources to be monitored increased.

Table 5.1 shows the result for 20 resources.

Serial No.	Number of Resources	Response Time [Seconds]
1	1	20 sec
2	5	25 sec
3	10	40 sec
4	15	40 sec
5	20	40 sec

Table 5.1: Data Fetching Time

Table 5.1 shows the results of the experiment carried out five times, with altering the number of resources and response time for each change in number of resources. From Table 5.1 it can be seen that for one resource the time taken is 20 seconds to fetch all the resource constraints but as the number of resources increases, the time becomes constant. This occurs when the number of resources is greater than 10. This supports our claim that the system is scalable.

The reason for this behavior is because the server is built in Node and it provides the advantage of doing asynchronous queries to the database, which makes the dashboard really scalable. The numbers increase but it does not affect the server because all these commands are being fired asynchronously. This gives us the power to test it on a larger number of resources.

One of the limitations in testing the dashboard was because of the SAVI testbed limits of the number of resources on each Edge; the system could not be tested for more than 20 resources.

5.2.2 Experiment 2: Scalability

This experiment was carried out to determine the scalability of numbers of users that can use the dashboard simultaneously.
In order to perform this test the server was set-up on one of the virtual machines in the SAVI testbed. After the server was set up, the URL for the dashboard was made available to the users across the SAVI testbed.
The aim of this experiment was to perform load testing on the dashboard. The parameter to be considered for load testing is the number of users the server can handle. The results are amazing, as the server is set up in Node and supports very high scalability. Table 5.2 below shows that the server is scalable.

Serial No.	Number of Users	Delay [Seconds]
1	10	no delay
2	20	no delay
3	40	no delay
4	80	no delay

Table 5.2: Data Fetching Time

From Table 5.2 it is evident that as the number of the users increases there is no time delay for the users except for an initial delay once the server starts to poll for data. The initial delay to fetch data for resource constraint is discussed in detail at the end of this chapter.

5.2.3 Experiment 3: User Experience

In this experiment we are evaluating the user experience. The experiment provides the qualitative analysis and provides feedback on how to improve the system. User satisfaction is an important criterion that we are going to use to evaluate the dashboard. The dashboard was provided to users and their satisfaction was noted. We based the measurement of user satisfaction on a scale from one to ten with the value of one denoting that the user is not at all satisfied and ten being totally satisfied.

- **Useful:** The content should fulfill a need and be original.

- **Usable:** It should be easy to use.
- **Desirable:** Design elements such as identity and images are used to evoke emotions and appreciation.
- **Findable:** The contents should be navigable and locatable.
- **Accessible:** How easily is the content on the site accessible?
- **Credible:** The users should trust and believe what they see.

5.2.4 Evaluation Summary

This chapter evaluated the effectiveness of using the SAVI testbed dashboard. The highlighted feature of the dashboard is that it provides resource constraint monitoring on an Edge. The resource constraint monitoring help us determine the load on an Edge. This dashboard also lays the foundation for future work, which can examine the migration of these resources from one Edge to another, based on the different constraints of an Edge. During the development of the SAVI dashboard, we identified a few short comings that need to be resolved to improve the dashboard in terms of initial load time.

Chapter 6

Conclusions and Future Work

This chapter summarizes the dissertation by outlining the contributions and presenting ideas for future improvement of the SAVI testbed dashboard.

6.1 Summary

The project provided a useful tool for monitoring the SAVI testbed resources. The first chapter of the report provides an introduction to OpenStack and Ceilometer. We undertook a comprehensive study in Chapter 2 to analyze some of the dashboard used for Cloud monitoring services. We then concluded that none of the applications presented in the literature have accomplished what we have accomplished by designing and developing this dashboard. Also we presented the reasons why this dashboard design and architecture is different from others. Next in Chapter 3, we discussed the backbone of the dashboard that is the Ceilometer service of OpenStack and a user guide. This chapter familiarizes the reader with the commands of the Ceilometer and its usage. We also discussed the capabilities of the Ceilometer for the SAVI testbed. Another goal of the project was to contribute to the available documentation regarding the usage of the Ceilometer in SAVI testbed resource monitoring. This was inspired by the lack of available resources that describe using the Ceilometer for the SAVI testbed. Chapter 3 provided a step-by- step tutorial of Ceilometer usage and then set the stage for the implementation of the SAVI testbed dashboard. Chapter 4 presented in detail the architecture of the dashboard as well as the implementation, using plenty of code snippets and diagrams. Chapter 4 described the technology used in the development of the dashboard and why it was chosen, then customized

the dashboard based on the administrator's need. In Chapter 5, we evaluated the dashboard. The evaluation was based on the response time per number of resources being monitored. Another evaluating criterion was based on scalability as to how many users the dashboard could handle at any given time. We also evaluated the dashboard based on its usability. We concluded that the response time was very low even when the number of resources increased. In terms of scalability, the dashboard is highly scalable and user friendly.

6.2 Future work

This section covers possible future work for improving the dashboard SAVI testbed. Since this was the first attempt to develop a dashboard of this kind, there are many areas with scope for future work. The customizable part of the dashboard can be further improved by giving users the option to select the number of resources they want to monitor at a particular time. Another area with scope for future work is live machine migration [7], which the SAVI testbed research community is working on. Based on the resource constraint [1] monitoring, a user should be able to move a resource from a particular Edge to another Edge, thereby providing live machine migration. The user needs to be an authentic SAVI testbed registered user.
Future work needs to examine the security aspect when requesting data from the SAVI testbed, as the bash script is not encrypted and it contains the user name and password. The dashboard is monitoring the resource and then displaying the values for those resource constraints. In future, a user should be able to set limits on the resource constraint and whenever these are violated there should be an alarm raised for the same.

Overall, there is considerable scope for future work on the SAVI testbed dashboard to comprehend it fully and reveal its potential.

Bibliography

[1] GENI Engineering Conference 23. Geni-savi federation. `http://groups.geni.net/geni/wiki/GEC23Agenda/GENISAVI`, June 2015.

[2] GENI Engineering Conference 23. Web api 2. `http://docs.openstack.org/developer/ceilometer/webapi/v2.html`, June 2015.

[3] Martin Arlitt, Sujata Banerjee, Cullen Bash, Yuan Chen, Daniel Gmach, Christopher Hoover, Priya Mahadevan, Dejan Milojicic, Eric Pelletier, RN Vishwanath, et al. Cloud sustainability dashboard. In *Sustainable Systems and Technology (ISSST), 2010 IEEE International Symposium on*, pages 1–1. IEEE, 2010.

[4] Juan Angel Lorenzo del Castillo, Kate Mallichan, and Yahya Al-Hazmi. Openstack federation in experimentation multi-cloud testbeds. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 51–56. IEEE, 2013.

[5] Pralhad Deshpande, Shachi Sharma, Arup Acharya, Kirk A Beaty, and Ashish Kundu. Use-case centric dashboard for cloud solutions. In *2014 IEEE International Conference on Services Computing (SCC)*, pages 840–841. IEEE, 2014.

[6] Rohit Dhand. Reducing web page post backs through jquery ajax call in a trust based framework. In *Computing Sciences (ICCS), 2012 International Conference on*, pages 217–219. IEEE, 2012.

[7] Erik Elmroth and Lars Larsson. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In *Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on*, pages 253–260. IEEE, 2009.

[8] Luís Ferreira, Goran Putnik, Manuela Cunha, Zlata Putnik, Hélio Castro, Catia Alves, Vaibhav Shah, and Maria Leonilde R Varela. Cloudlet architecture for dashboard in cloud and ubiquitous manufacturing. *Procedia CIRP*, 12:366–371, 2013.

[9] Xiao-Feng Gu, Le Yang, and Shaoquan Wu. A real-time stream system based on node. js. In *Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014 11th International Computer Conference on*, pages 479–482. IEEE, 2014.

[10] Joon-Myung Kang, Hadi Bannazadeh, and Alberto Leon-Garcia. Savi testbed: Control and management of converged virtual ict resources. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 664–667. IEEE, 2013.

[11] Joon-Myung Kang, Thomas Lin, Hadi Bannazadeh, and Alberto Leon-Garcia. Software-defined infrastructure and the savi testbed. In *Testbeds and Research Infrastructure: Development of Networks and Communities*, pages 3–13. Springer, 2014.

[12] Knockout. `http://knockoutjs.com`.

[13] Kai Lei, Yining Ma, and Zhi Tan. Performance comparison and evaluation of web development technologies in php, python, and node.js. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 661–668, Dec 2014.

[14] Openstack. Basic architecture of openstack. `http://docs.openstack.org/developer/ceilometer/architecture.html`.

[15] OpenStack. Ceilometer openstack. `http://docs.openstack.org/developer/ceilometer/`.

[16] OpenStack. Ceilometerarchitecture. `http://docs.openstack.org/developer/ceilometer/architecture.html`.

[17] OpenStack. Ceilometerquickstart. `https://www.rdoproject.org/install/ceilometerquickstart/`.

[18] Openstack. Nova. `http://docs.openstack.org/developer/nova/runnova/`.

[19] OpenStack. Ceilometeropenstack. http://ieeexplore.ieee.org.ezproxy.library.uvic.ca/stamp/stamp.jsp?tp=&arnumber=6735395, June 2015.

[20] Programming. Ajax. https://en.wikipedia.org/wiki/Ajax_(programming).

[21] Programming. Knockout. http://knockoutjs.com.

[22] Tiago Rosado and Jorge Bernardino. An overview of openstack architecture. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pages 366–367. ACM, 2014.

[23] Ridwan Sanjaya. Trade-off analysis for web application using green ajax. In *Management of Innovation and Technology (ICMIT), 2010 IEEE International Conference on*, pages 1050–1054. IEEE, 2010.

[24] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, 2012.

[25] S. Tilkov and S. Vinoski. Node.js: Using javascript to build high-performance network programs. *Internet Computing, IEEE*, 14(6):80–83, Nov 2010.