SAN JOSÉ STATE UNIVERSITY

CMPE283 Virtualization Technology

Project: Openstack with Enhanced User Interfaces

Team Members

Sravya Dara

Srividhya Kuppam

Sowmya Vuddaraju

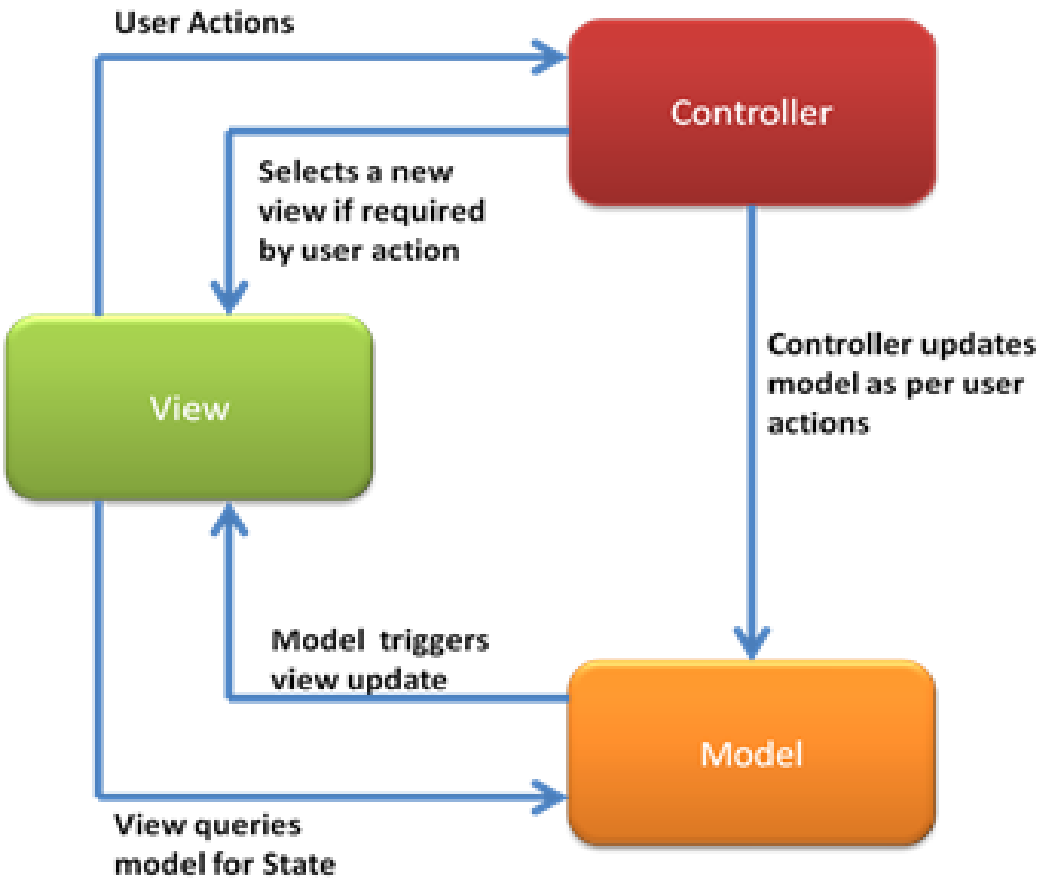
Sravya Yennamreddy

Ramyakrishna Yarram

Preface

This document guides through OpenStack components and implementation of the same functionality in our custom dashboard project. This report is produced in response to the semester project for CMPE283 at San Jose State University Fall 2014. The expected readership of this document includes Professor Dr. Thomas Hildebrand and CMPE283 students. The primary purpose of this project is to create a web based application which uses OpenStack REST APIs, command line interface and a Mobile based application.

MVC Architecture



- The Model represents the structure of the data in the application, as well as application-specific operations on that data
- A View (of which there may be many) presents data in some form to user, in the context of some application function
- A Controller translates user actions (mouse motions, keystrokes, words spoken, etc.) and user input into application function calls on the model, and selects the appropriate View based on user preferences and Model state

Technologies Used

- **Eclipse**

Eclipse is an integrated development environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications. By means of various plug-ins, Eclipse may also be used to develop applications in other programming languages: JavaScript, Perl, PHP and Python. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

- **Play**

Play is an open source web application framework, written in Scala and Java, which follows the model–view–controller (MVC) architectural pattern. It aims to optimize developer productivity by using convention over configuration, hot code reloading and display of errors in the browser.

- **PhoneGap**

PhoneGap is a mobile development framework it enables software programmers to build applications for mobile devices using JavaScript, HTML5, and CSS3, instead of device-specific languages such as Objective-C. It enables wrapping up of HTML, CSS and Javascript code depending upon the platform of the device. It extends the features of HTML and Javascript to work with the device.

- **Python**

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. The language provides constructs intended to enable clear programs on both a small and large scale.

- **Tomcat Apache**

Apache Tomcat is an open source web server and servlet container. Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Oracle, and provides a “pure java” HTTP web server environment for Java code to run in. In the simplest config, Tomcat runs in a single operating system process. The process runs a Java virtual machine. Every single HTTP request from a browser to Tomcat is processed in the Tomcat process in a separate thread.

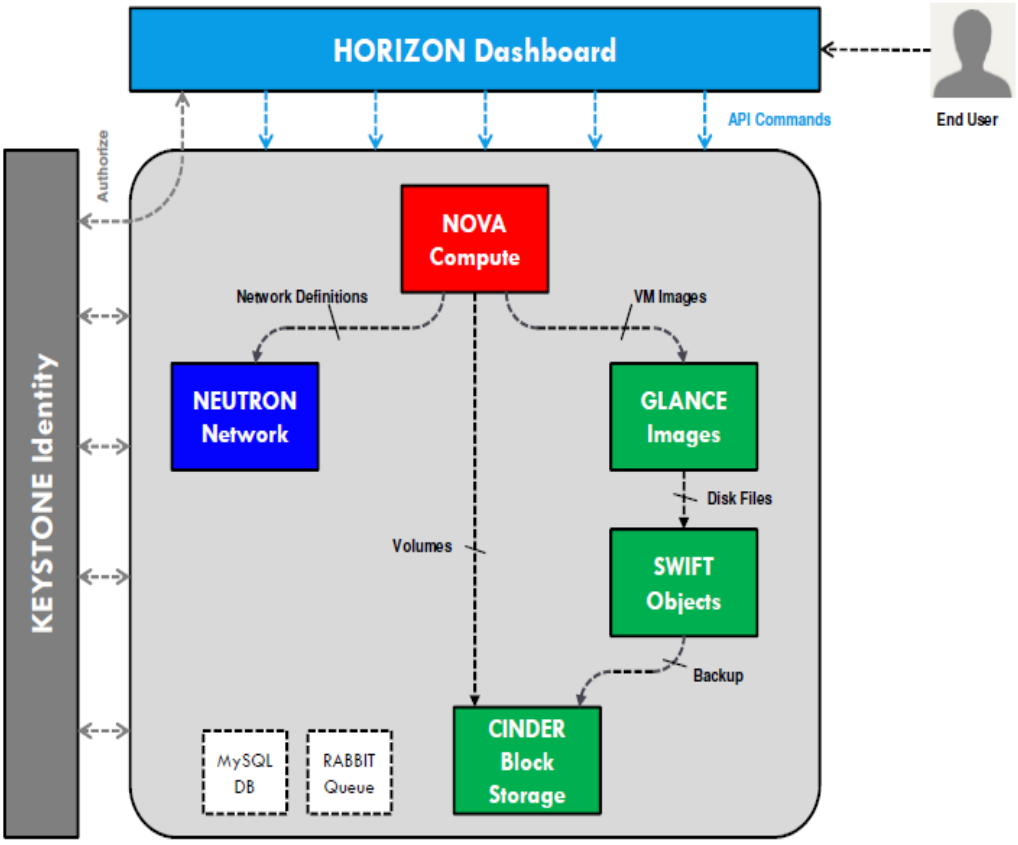
- **JClouds**

Apache jclouds is an open source multi-cloud toolkit for the Java platform that gives you the freedom to create applications that are portable across clouds while giving you full control to use cloud-specific features.

Openstack Architecture

Introduction to Openstack

Openstack is a community of open source developers, participating organizations and users who are building and running the open source cloud operating system. Openstack is an Infrastructure as a service which is known as a Cloud Operating systems, that takes resources such as compute, storage, network, virtualization technologies and controls those resources at a data center level.



NOVA (COMPUTE)

Nova is the project name for Openstack Compute, a cloud computing fabric controller, the main part of an IaaS system. Individuals and organizations can use Nova to host and manage their own cloud computing systems.

GLANCE (IMAGES)

The Glance project provides services for discovering, registering, and retrieving virtual machine images. Glance has a Restful API that allows querying of VM image metadata as well as retrieval of the actual image.VM images made available through Glance can be stored in a variety of locations from simple file systems to object-storage systems like the Openstack Swift project.

SWIFT (OBJECTS)

OpenStack Object Storage is an object-based storage system that stores content and metadata as objects. You create, modify, and get objects and metadata by using the Object Storage API, which is implemented as a set of Representational State Transfer (REST) web services

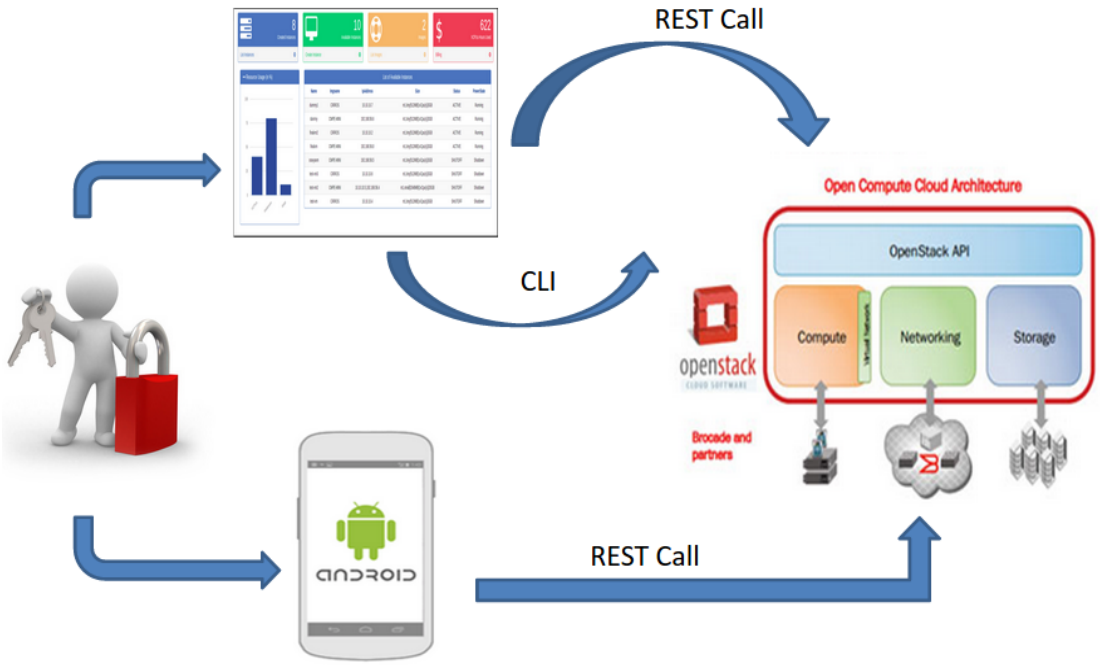
NEUTRON (NETWORK)

Neutron is an Openstack project to provide "networking as a service" between interface devices (e.g., vNICs) managed by other Openstack services (e.g., nova).Give cloud tenants an API to build rich networking topologies, and configure advanced network policies in the cloud.

CINDER (BLOCK STORAGE)

Cinder is an OpenStack project to provide “block storage as a service”. Cinder provides an infrastructure for managing volumes in OpenStack. It was originally a Nova component called nova-volume, but has become an independent project since the Folsom release.

Project Overview & Implementation



Project Goals

Goal 1: Web UI using REST API

Developed a web UI that displays the following functions which uses a REST call to access the OpenStack API's.

- Authentication using Keystone.
- List VM's: Displays list of created VM's.
- Create Instance: Create instances with the details (flavor, image) provided.
- List Images: Displays list of images for a particular zone.

Goal 2: Web UI using CLI

Developed a web UI that creates an instance which uses CLI (Command Line Interface) to access the OpenStack server.

Goal 3: Mobile UI using REST API

Developed a mobile UI which uploads an image to the container using a REST call. It communicates with Swift API to store image in the container.

Goal 4: Resource Usage

Component which pulls dynamic resource usage information and plot the data on a graph.

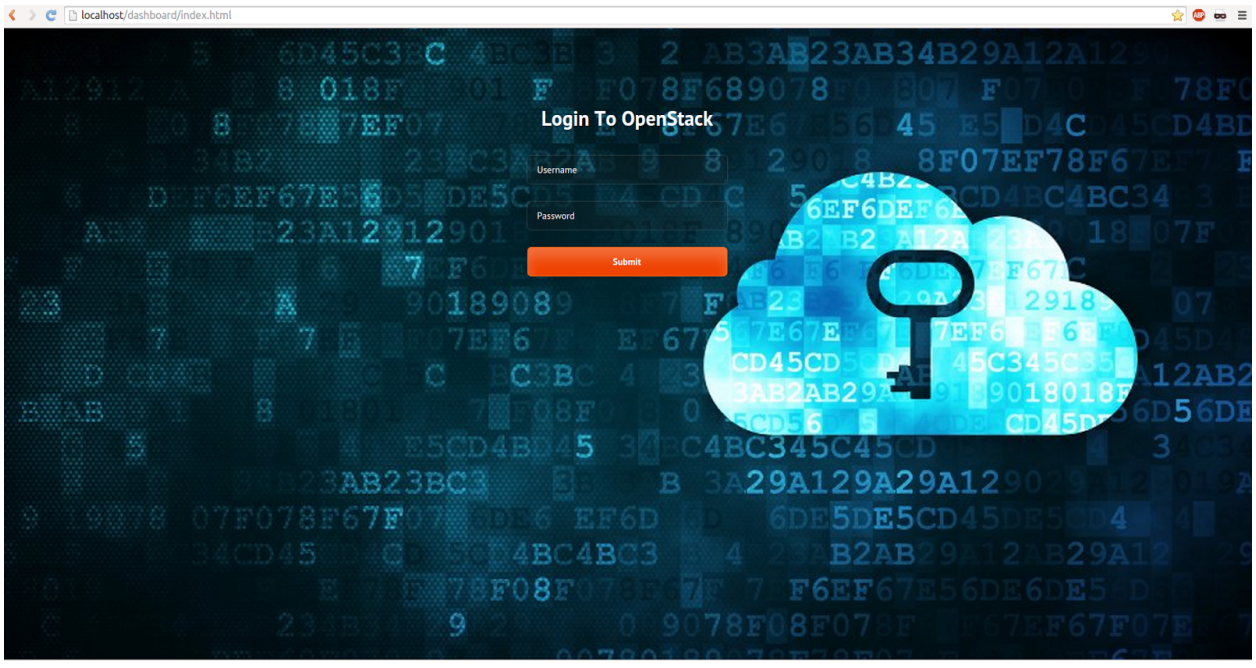
Goal 5: Billing

Billing component provides the usage summary of number of VCPU hours used by the project or under a tenant for a given month and apply rates accordingly.

Implementation

User Authentication

The user inputs his credentials on the login page. On submitting the details the user is authenticated and redirected to the Dashboard (charts.html). A python script is used to perform authentication using KeyStone API (via REST calls)



Pseudo Code:

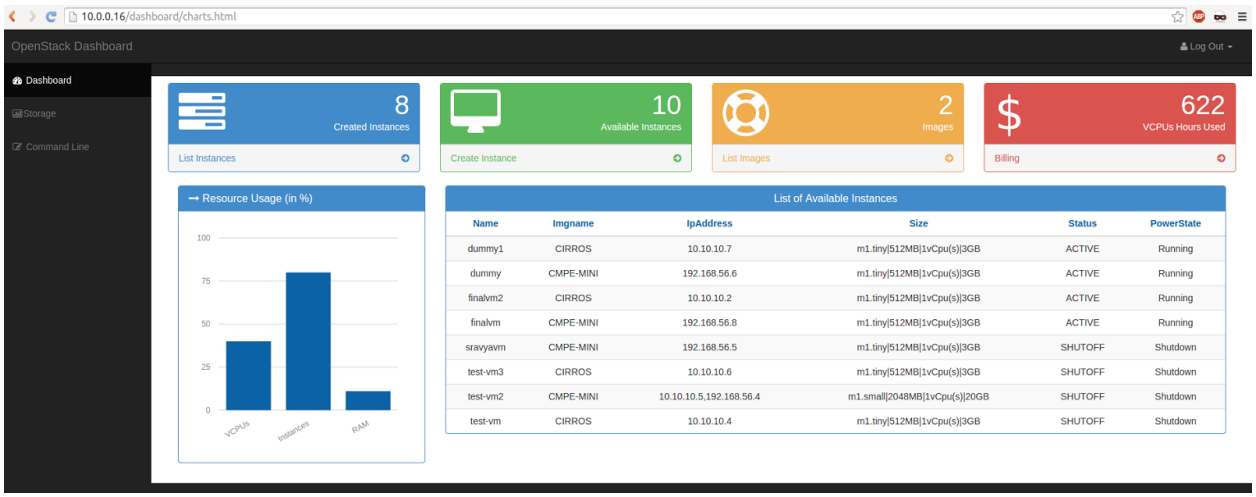
- User provides username, password and submits the form
- A post method is used to call a python script
- In the script, a REST call is used to authenticate with Keystone using JSON request.
- If the authentication is successful, the username and password are stored in a cookie
- If the authentication fails, the user is redirected to an error page.

Method	**URI**
POST	/v2.0/tokens

JSON Request

```
{
    "auth": {
        "tenantName": "demo",
        "passwordCredentials": {
            "username": "demo",
            "password": "devstack"
              }
          }
}
```

The created cookie is used to provide the username and password for the functionalities in the application. “Logout” functionality is implemented by a function which deletes the cookie.



Web UI using Rest API

Features:

- ❖ Create VM
- ❖ List Instances
- ❖ List Images

- **List Instances :**

This displays list of all the VM's created. This includes the details of VM name, VM size, VM ipaddress, etc. for displaying these details several API's are involved which are stated as follows:

- Nova API: Fetches the image and server API for a particular zone
- Image API: Name of the image can be fetched which will be used in creating VM
- Server API: Will fetch the details of all the servers for a particular zone
- Flavor API: Fetches the details of flavor for a particular zone

Method	**URI**
GET	**/compute/Virtual-machines**

The response object is as follows:

{"project":"2114b5a75d1248d19a1f6a8d85ea9e1c","host":"localhost.localdomain",

"vmName":"vm_test",

"imageName":"CMPE-MINI",

"ipAddress":"10.10.10.2",

"size":"m1.tiny|512MB|1VCPU(s)|3GB",

"status":"SHUTOFF",

"powerState":"4",

"zone":"RegionOne"},

{"project":"2114b5a75d1248d19a1f6a8d85ea9e1c",

"host":"localhost.localdomain","

vmName":"MY-SECOND-VM",

"imageName":"CMPE-MINI",

"ipAddress":"20.20.20.2",

"size":"m1.tiny|512MB|1vCpu(s)|3GB",

"status":"SHUTOFF",

"powerState":"4",

"zone": "RegionOne}

Instances listed:

List of Available Instances

Name	Imgname	IpAddress	Size	Status	PowerState
test-vm3	CIRROS	10.10.10.6	m1.tiny|512MB|1vCpu(s)|3GB	SHUTOFF	Shutdown
test-vm2	CMPE-MINI	10.10.10.5,192.168.56.4	m1.small|2048MB|1vCpu(s)|20GB	SHUTOFF	Shutdown
test-vm	CIRROS	10.10.10.4	m1.tiny|512MB|1vCpu(s)|3GB	SHUTOFF	Shutdown

- **Create Instance:**

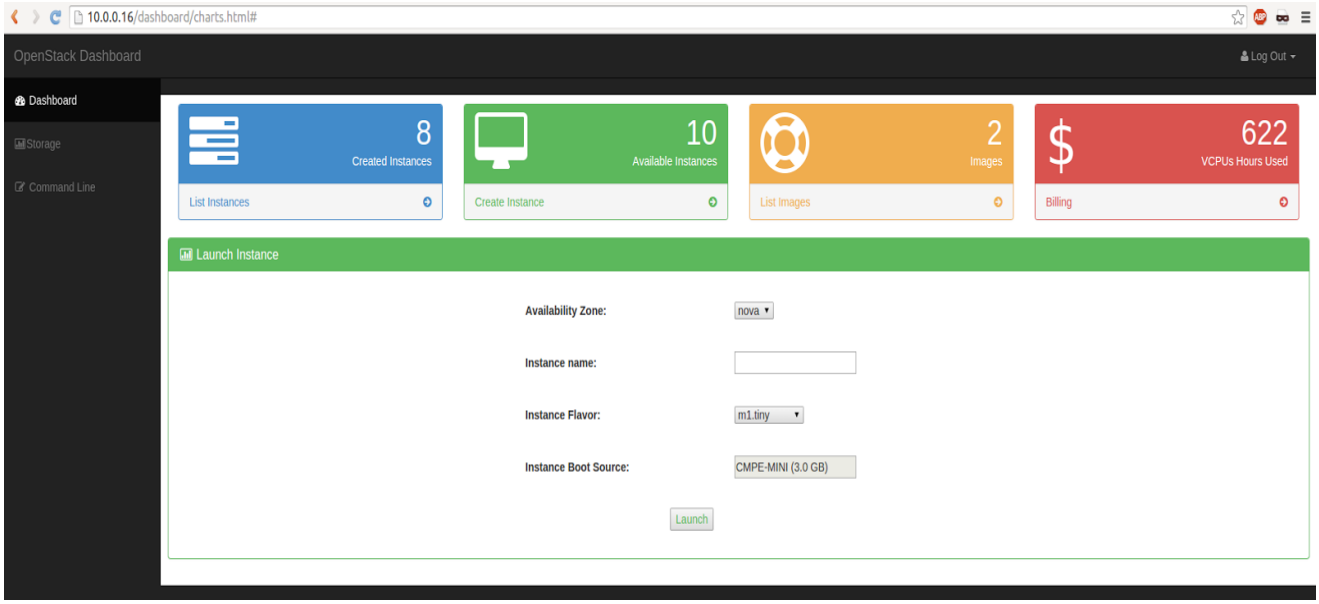
Details of the Instance such as Instance name, Zone, Flavor Id, ImageId should be entered by the user. On submitting the details, an Instance is created using REST call and can be viewed in the Dashboard (Charts.html). It uses the following APIs:

- Nova API: Fetches the server API for a particular zone
- Neutron API: Fetches the networkId by taking zone
- Server API: Uses it to create an instance

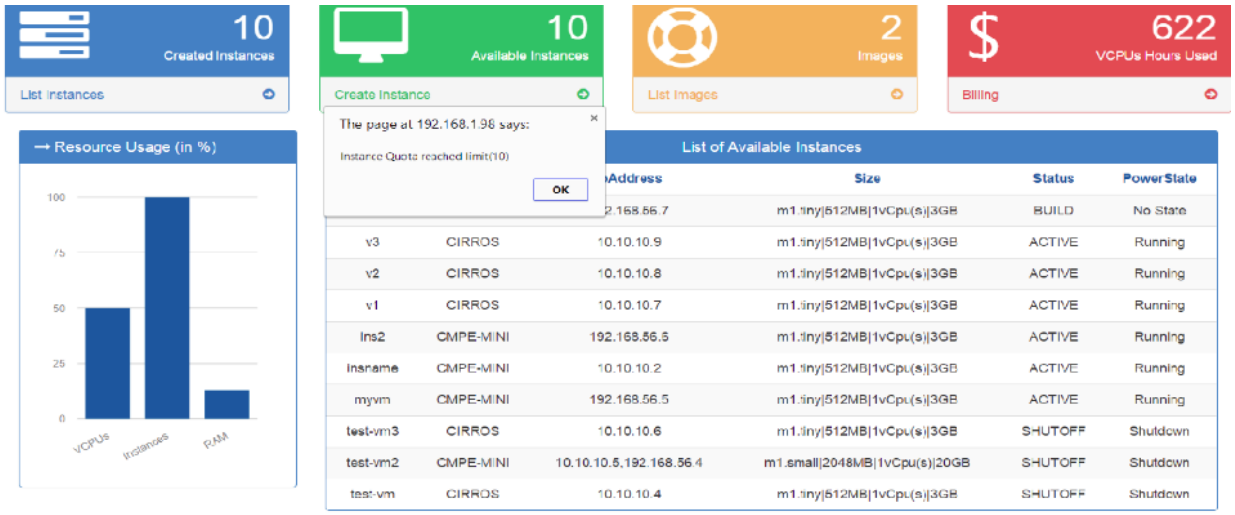
Pseudo Code:

- User should provide all the required inputs and submits the form using a POST method.
- Play framework is used to redirect the request to a java function.
- User should authenticate with Nova Api and Neutron Api to access those resources.
- Neutro nAPI is also used to obtain the network ID.
- Finally, the instance is created on the network specified by network ID.

After creating the instance, a list instances method is called to show the created Instance in the list.



User is allowed to create only upto 10 instances.If the user wants to create more than 10 a message generated saying “Instance Quota has reached limit(10)”



- **List Images:**

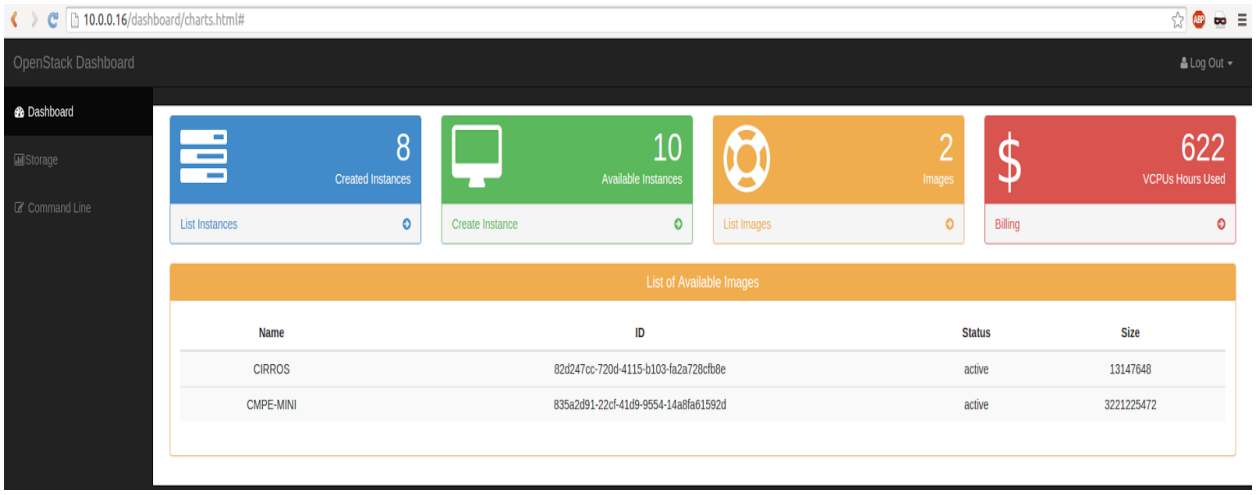
Displays the list of all images for a particular zone using Image API.

Method URI

GET /compute/Virtual-machines/RegionOne/images

The response object is as follows:

{"name":"CMPE-MINI","id":"06fcf2f9-2614-425b-91a3-5a6170fabbe7"}



- **Billing:**

Billing provides the usage information like total VCPUs, memory and disk used by the project from the start of the month till current date. It has two components: Usage summary and Bill generation.

In order to implement usage summary feature we used the openstack compute api as shown below.

Method URI

GET

/v2/{tenantId}/os-simple-tenant-usage?start=<start_date>&end<end_date>

The response object is as follows:

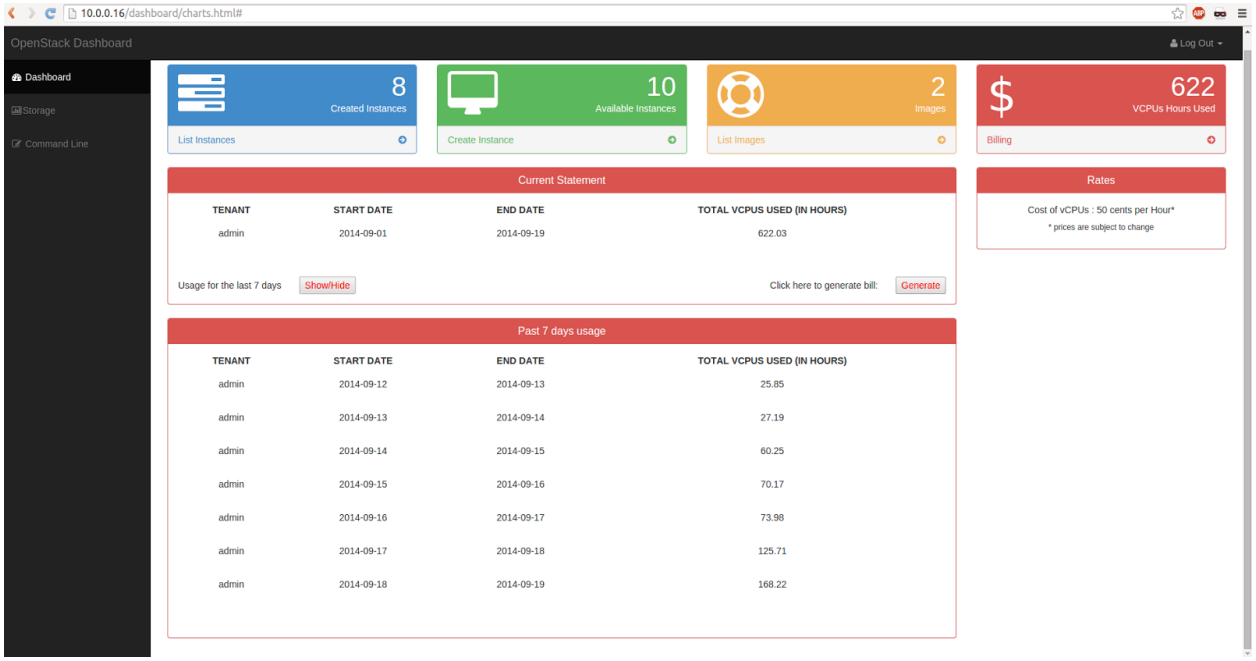
```
{
  "tenant_usages": [
        {
```

```
        “start": "2014-09-01T00:00:00.000000",
        "stop": "2014-09-18T00:00:00.000000",
        "tenant_id": "f1d53a137a404740869ce62b18bf1fe2",
        “total_hours": 453.81444444444463,
        “total_local_gb_usage": 3249.960000000001,
        "total_memory_mb_usage": 458399.2888888888,
        "total_vcpus_usage": 453.81444444444463
        }
    ]
}
```


Using the same api calls, we were able to get previous months / days usage summary for a given tenant id or project id.

In order to implement a simple bill generation feature, we got the VCPUs used per hour information from the same api and apply the rates that are shown in the dashboard.

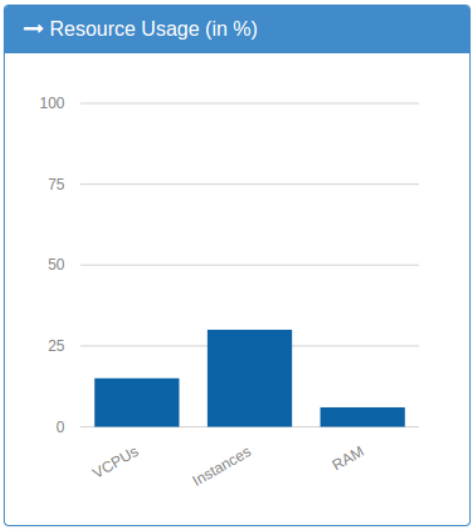
User has a facility to view the previous months/days usage summary by clicking on the Show/Hide button.



- **Resource Usage:**

To prevent system capacities from being exhausted without notification there are limits set for each tenant. For example with the default limit setting, a user cannot create more than 10 instances under a given project id.

Resource usage section provides information about the number of instances used out of allocated, number of VCPUs used out of allocated and RAM used out of allocated in percentages. All the usage information is dynamic and to represent this information we used Morris charts. In order to get the information about used and total allocated, we used the below Openstack api provided.



Method	**URI**
GET	**/v2/{tenant_id}/limits**

The response object is as follows:

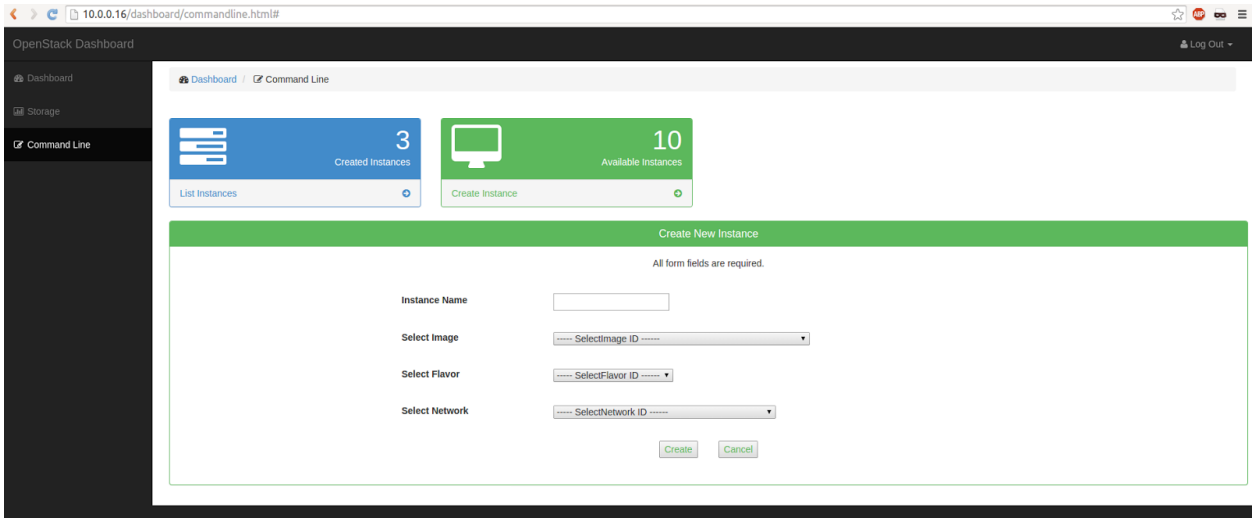
```
{
 "limits": {
        "absolute": {
                "maxImageMeta": 128,
                "maxPersonality": 5,
                "maxPersonalitySize": 10240,
                "maxSecurityGroupRules": 20,
                 "maxSecurityGroups": 10,
                "maxServerMeta": 128,
                "maxTotalCores": 20,
                "maxTotalFloatingIps": 10,
```

```
"maxTotalInstances": 10,
 "maxTotalKeypairs": 100,
 "maxTotalRAMSize": 51200,
"totalCoresUsed": 7,
 "totalFloatingIpsUsed": 0,
 "totalInstancesUsed": 7,
"totalRAMUsed": 6656,
"totalSecurityGroupsUsed": 0
        },
"rate": []
        }
}
```


Web UI using Command Line :

- **Create Instance Using Command Line**:

The create instance feature allows the user to create an instance in the openstack by providing different images that are already available in glance, choosing between different flavors/configurations and the networks which are available in neutron. But instead of making a rest call for creation, we implemented this feature through a python script which can actually login to the openstack host and run “nova boot” commands. Installing “pxssh” module for python is the prerequisite for the script to execute. The excerpt of the script is shown below.

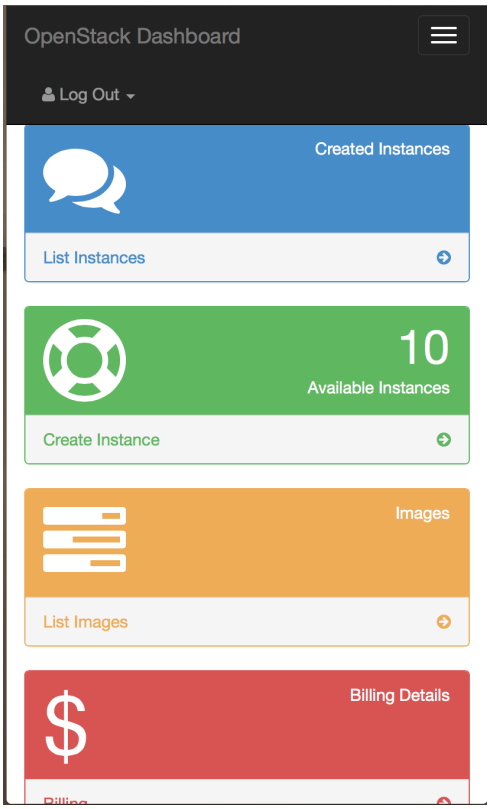



```
s = pxssh.pxssh();
hostname = get_hostip()
s.login(hostname, username, password, port="2022")
cmd = "nova boot --image %s --flavor %s --nic net-id=%s %s"     %(imageId,
flavorId, networkId, instanceName)
```

s.sendline(cmd)

Mobile UI using REST

We have designed our mobile application using Phone Gap. Phone Gap is an open source mobile application framework designed to develop mobile apps using HTML, CSS and JavaScript. These applications can be deployed on wide range of mobiles without losing features of a native app. We made it simple by having the same base code for both the Web App and Mobile App.



Features Include:

- List Instances :
  It displays list of all instances created. This includes VM Name, VM Size, and VM IP Address.
- Create Instance:
  Details of the VM such as VM name, VM flavor, Image name should be entered and on submitting the details an Instance is created.
- List Images:
  It displays list of all the images available .It consists of Image Name and Image ID.

- Container Creation:

Name of the container to be created is entered and upon clicking the create container button, a container is created in the OpenStack dashboard .To implement the Container creation, we used OpenStack Swift API as shown below.

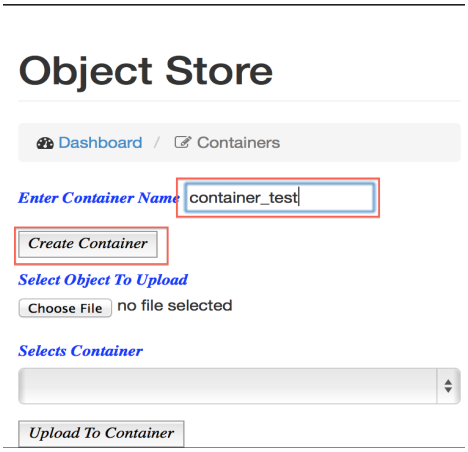
Method **URI**

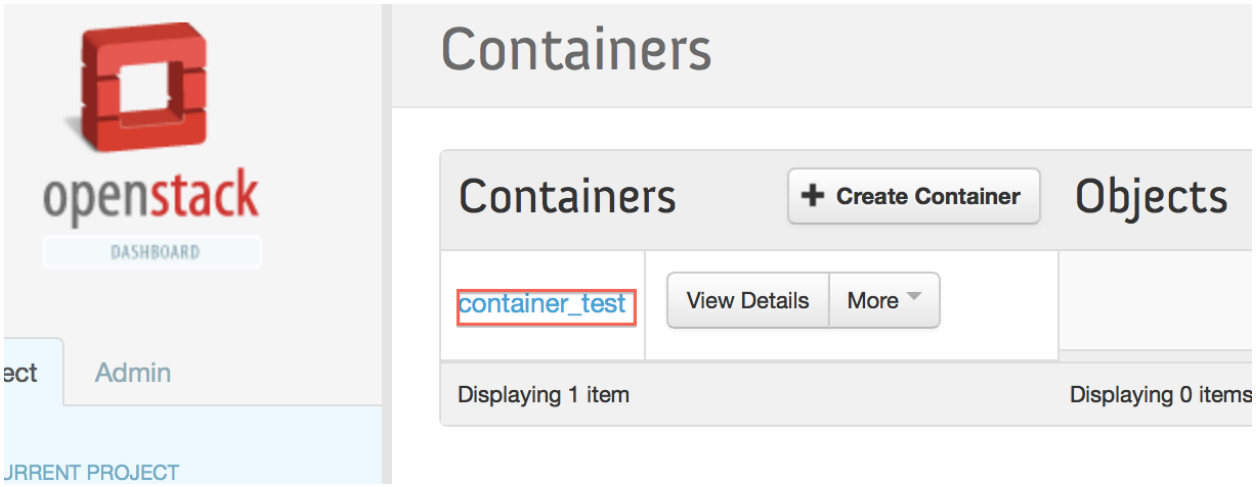
GET **/compute/virtual-machines/:Zone Id/Containers/Container Name**

SWIFT API for Container Creation:

```
ContainerApi containerApi = swiftApi.getContainerApiForRegion(zoneId);
containerApi.create(containerName, null);
```

Example:





- Upload File to Container:

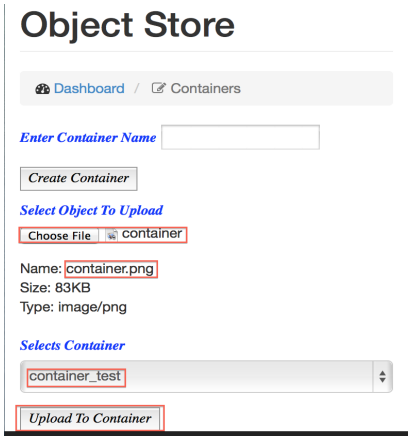
File to be uploaded in the container must be chosen through the Choose File button. The Container in which the file needs to be uploaded must be chosen from the Select Container Drop down. Upon clicking the Upload to Container button, the file is uploaded to the selected container and can be viewed in the Openstack Dashboard.

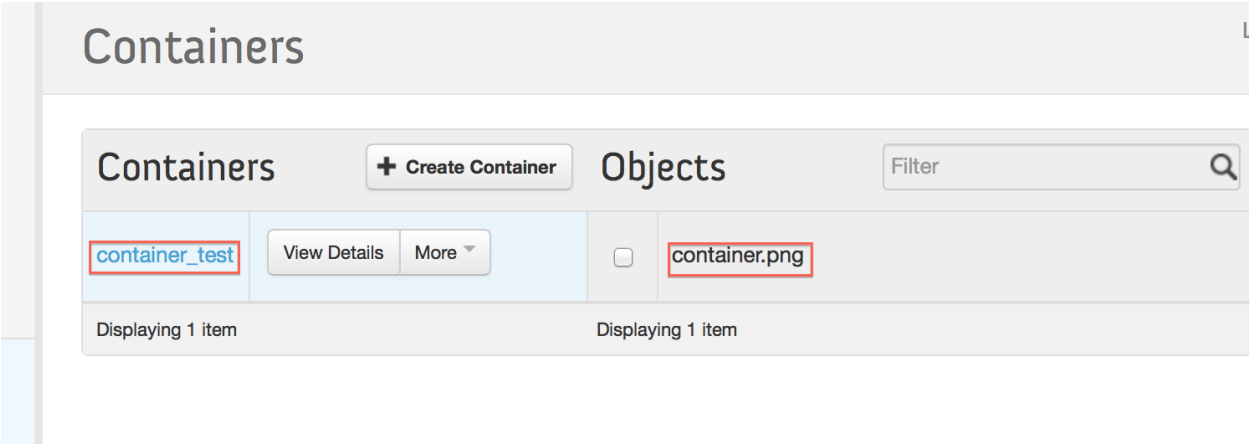
Method	URI
GET	**/compute/virtual-machines/ upload/: zoneId/:containerId/:fileName**

SWIFT API to Upload an Object:

```
ObjectApi objectApi = swiftApi.getObjectApiForRegionAndContainer(zoneId, containerId);
Payload payload = newByteSourcePayload(wrap(bytes));
String OBJECT_NAME = filename;
objectApi.put(OBJECT_NAME, payload, PutOptions.Builder.
 metadata (ImmutableMap.    of("key1", "value1")));
```

Example:





Authors:

Sravya Yannamreddy	**User Authentication, Instance Creation using REST**
Sowmya Vuddaraju	**List VM's , List Images using REST**
Ramyakrishna Yarram	**Instance Creation using CLI, Billing**
Sravya Dara	**Uploading image from Mobile using REST**
SriVidhya Kuppam	**Graphs demonstrating % of resource usage**

References :

http://jclouds.apache.org/guides/openstack/#nova

http://getbootstrap.com/

https://www.playframework.com/

http://phonegap.com/