281-01: Cloud Technologies

Integrated System Design

Submitted To

Dr. Jerry Gao

Date of Submission

28th April 2015

Submitted By

Team # 17

Name	SJSU ID
Chungsik Song	005298034
Xiao Wei	010128742
Shubham Gupta	010006568
Jayshri Bhausaheb Nikam	010038821

Table of Contents

Introduction

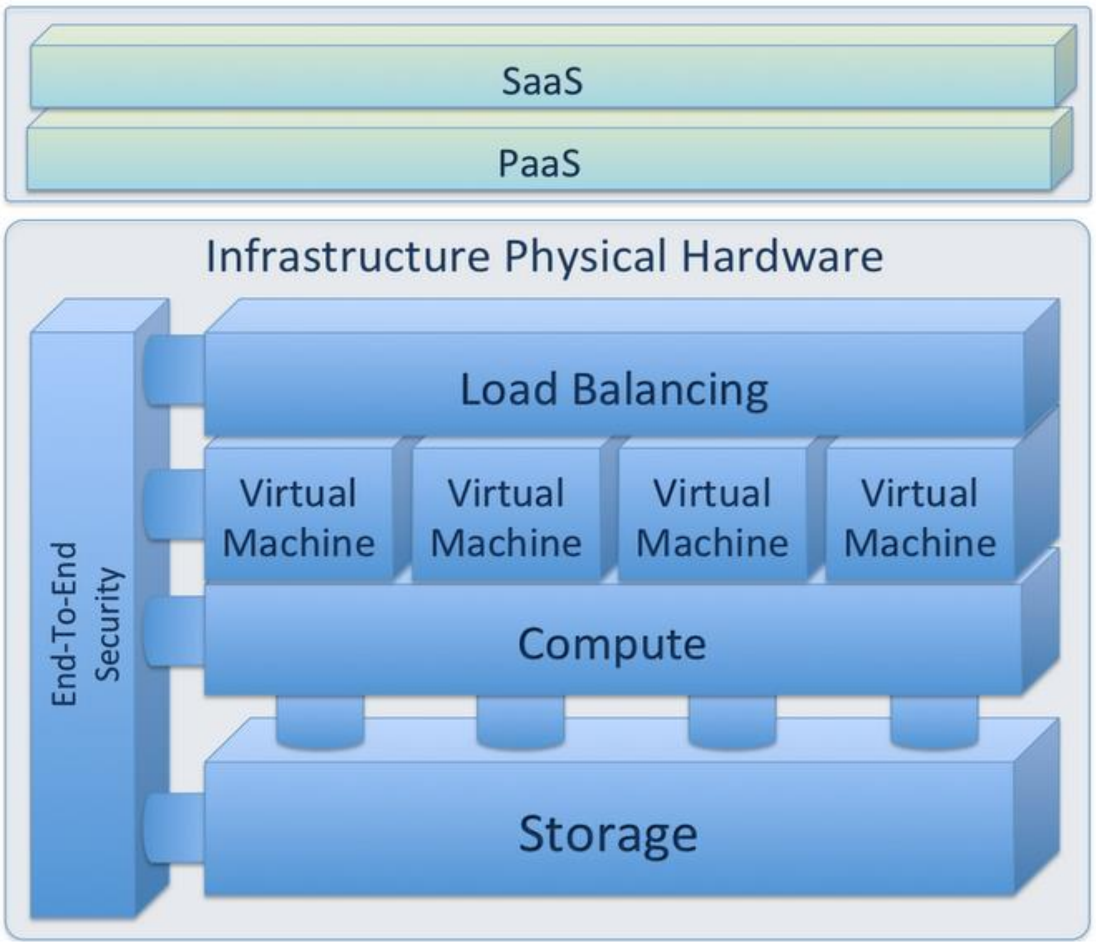
Testing as a service is an on demand testing where the testing activities of an organization is outsourced to a service provider.

Mobile application testing –

We can test Mobile applications using TaaS as it provides a realistic platform to test mobile applications, and leverages the key advantages of cloud. Taas is being considered a viable testing model by organizations to save costs and improved service for their test requirements.

Benefits of Mobile Testing on Cloud:

MTaas has significant advantages over traditional testing environments as it is highly scalable, cost effective, provides faster deployment and reduces the load of maintenance at organizations end.



Overview and System Design

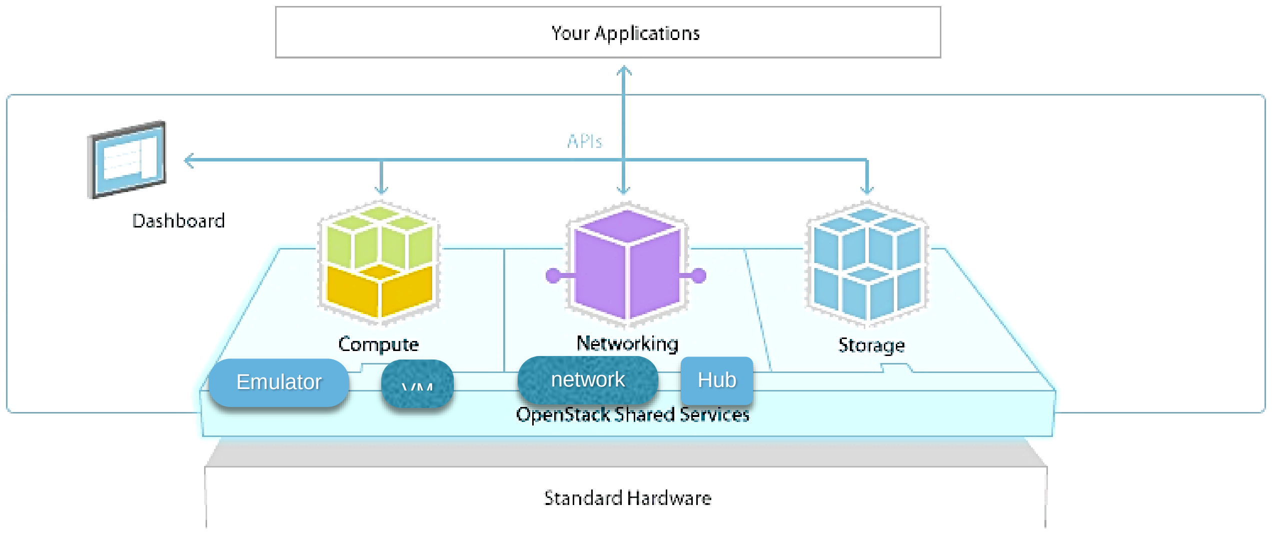
The resources for mobile testing consists of three general types of components — test server, mobile device and hub.

To make the Iaas for mobile testing, the three types of resources should be built on cloud. The system will provide the user with these resources, and manage them with proper algorithms to achieve good performance and balanced loads.

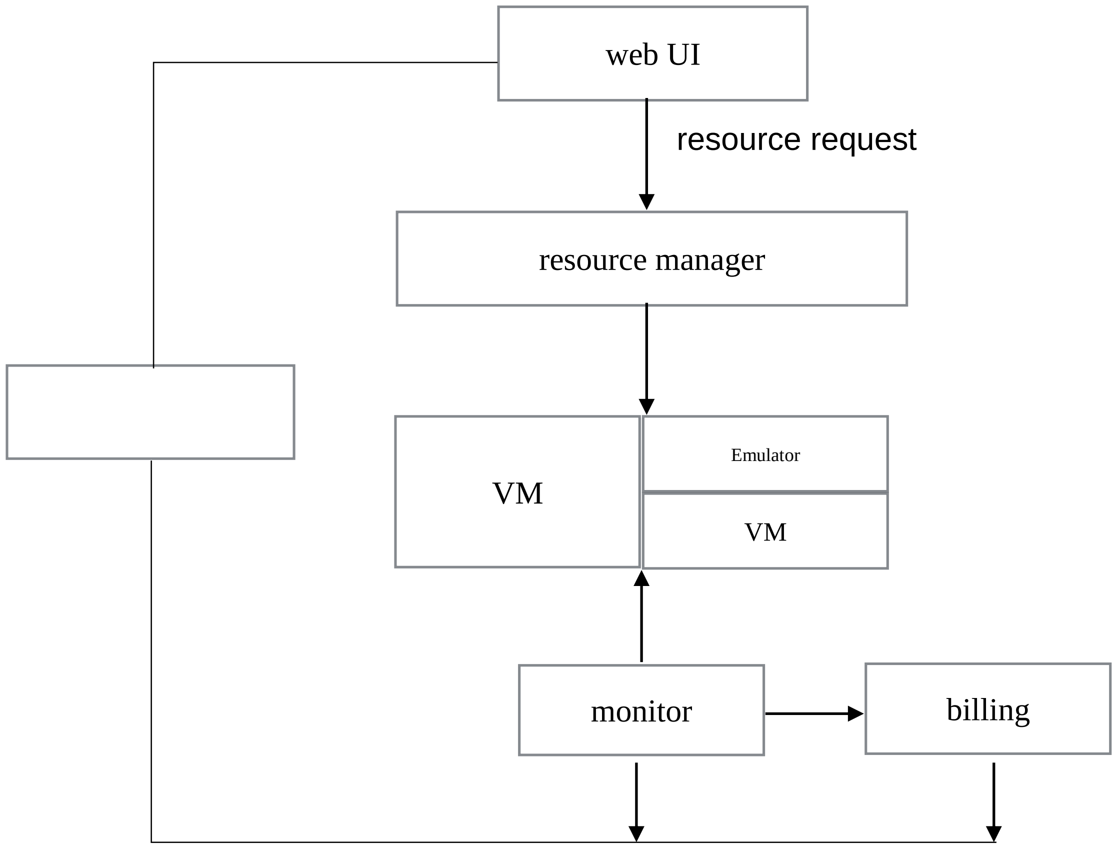
All the resources usage are monitored, computers, mobile devices and hubs. The usage statistics are crucial for resource management and billing nodes.

A billing module will be built based on the monitoring function, with proper policies applied to the resources allocated and used by particular user.

Users of the system will communicate through a web UI, requesting for resources, configuring resources, viewing the status of resources are the most important functionalities of this web UI.



System Architecture



The architecture of this system consists of these components:

Resource Management
Resource provision and allocation

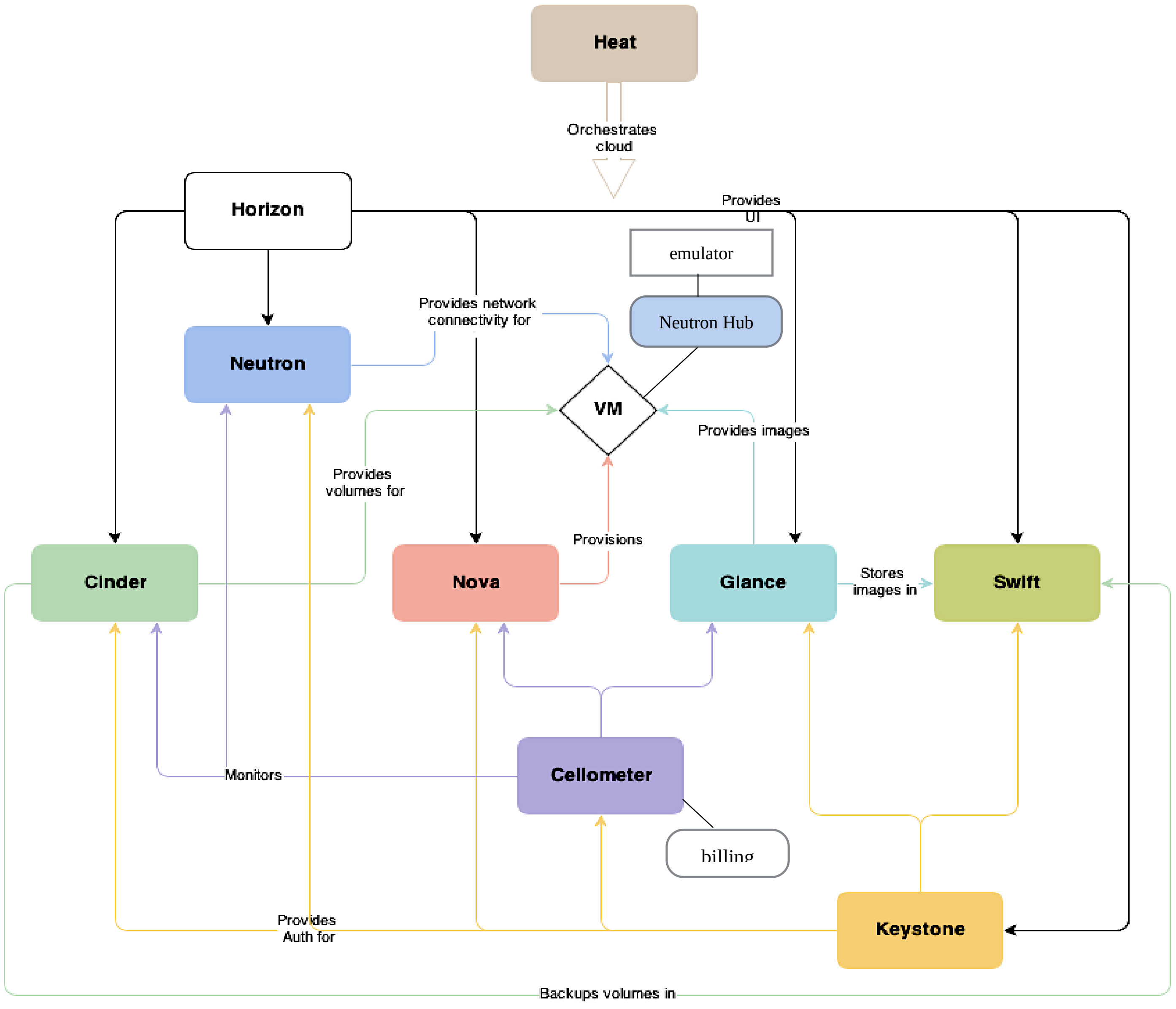
Emulator Controller
Host the emulators on VMs

Resource Monitor
Keep track of the usage of resources

Billing Component
Calculate bill from monitoring statistics

Web Server
User interface for requesting and viewing

Based on the conceptual architecture of Openstack, we are going to add the new resources for our system. The test servers can be view as VM here. Similar to VM, emulators are virtual mobile evices, they should also be provided by Nova-the computing node of Openstack, with storage from Cinder, network offered by Neutron and images from Glance.



Openstack offers three types of resources in their IaaS – compute, networking and storage. For our Mobile Testing IaaS, test server can be seen as a typical compute resource. In addition, the mobile device emulator is another type of computing resource. As to networking, hubs or virtual hubs are needed to connect mobile devices or emulators to test servers.

Mobile Hubs is analogous to Neutron. Neutron provide network for VMs, which can let the VMs communicate with each other. A new Neutron-like node is needed to realize connection between test servers(VMs) and Mobile Devices(Emulators).

The images of emulators should be provided by Glance, and stored in Swift.

New features will be added to Nova to manage life-cycles of VMs and Emulators, including spawning, scheduling and decommissioning of VMs and emulators on demand

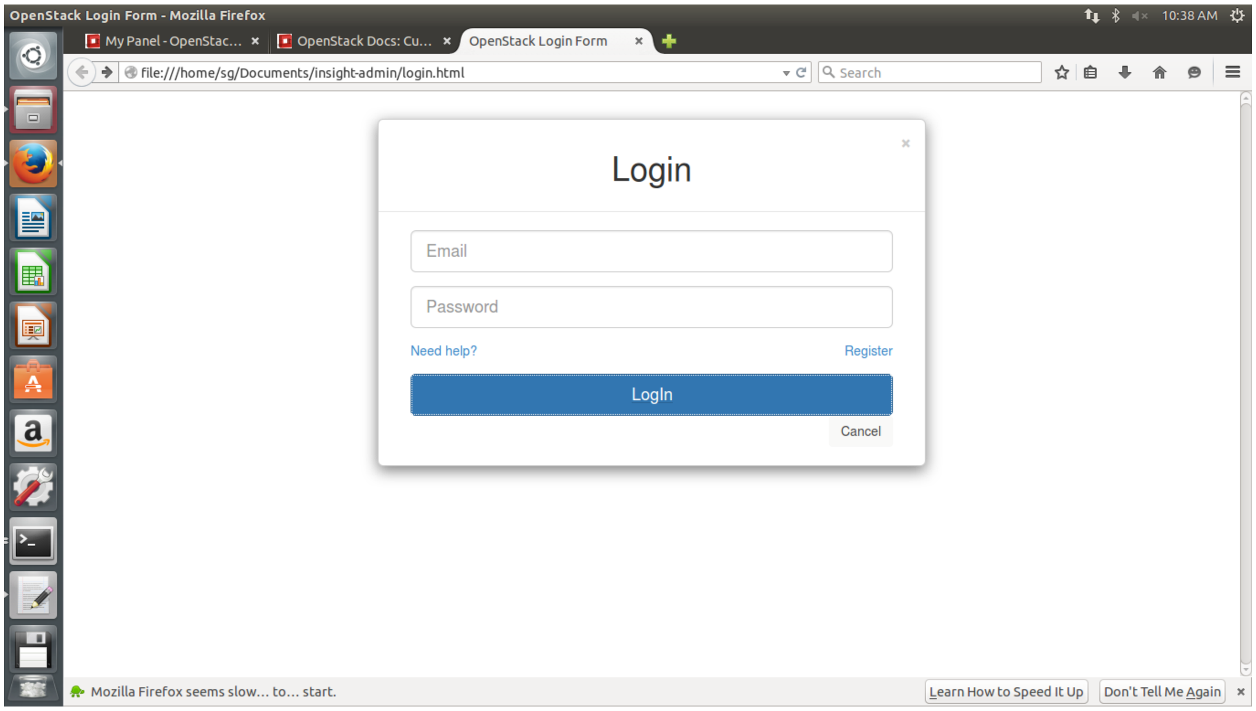
Ceilometer will be monitoring the resources used by the VMs and Emulators on Nova, and the network resources used at Neutron and our 'Neutron Hub'.

Horizon as the dashboard, will provide UI for test servers(VMs), Mobile Devices(Emulators) and Hubs.

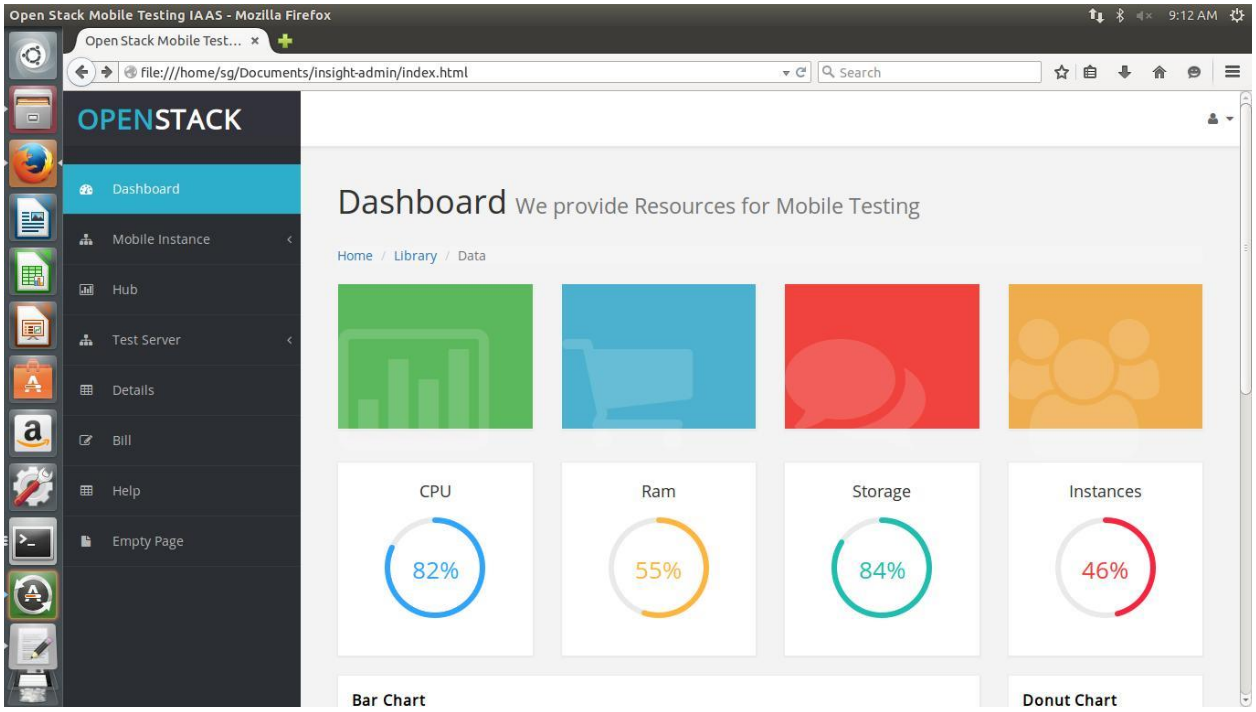
Dashboard

The dashboard component of the project provides the user an interface to communicate with the features of OpenStack. User can launch various mobile instances as well as test server instances on OpenStack. The Hub tab allows the user to communicate their test server with the mobile instances. User can also view the details of their instances like CPU usage, Ram, Storage, number of instances. User can also generate the bill of the instances created. OpenStack horizon code is modified to display details about the instances launched and cost of the resources utilized.

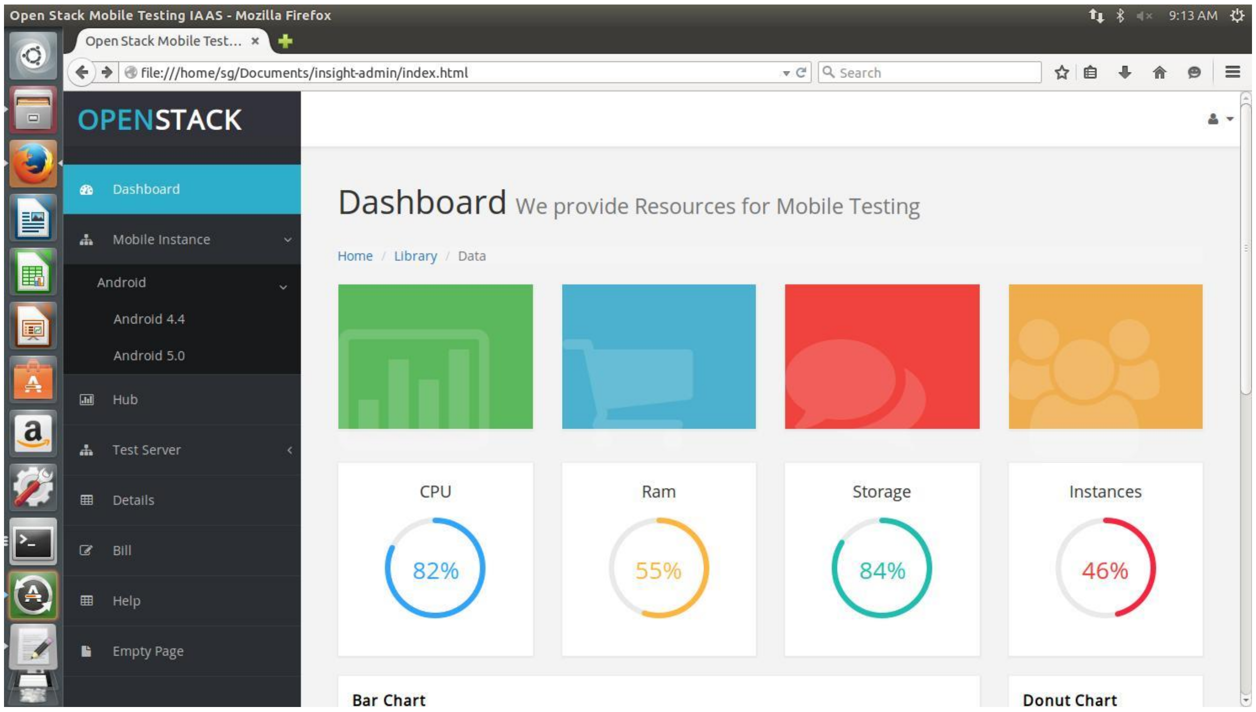
1. Login page authenticates the user by verifying the username and password.



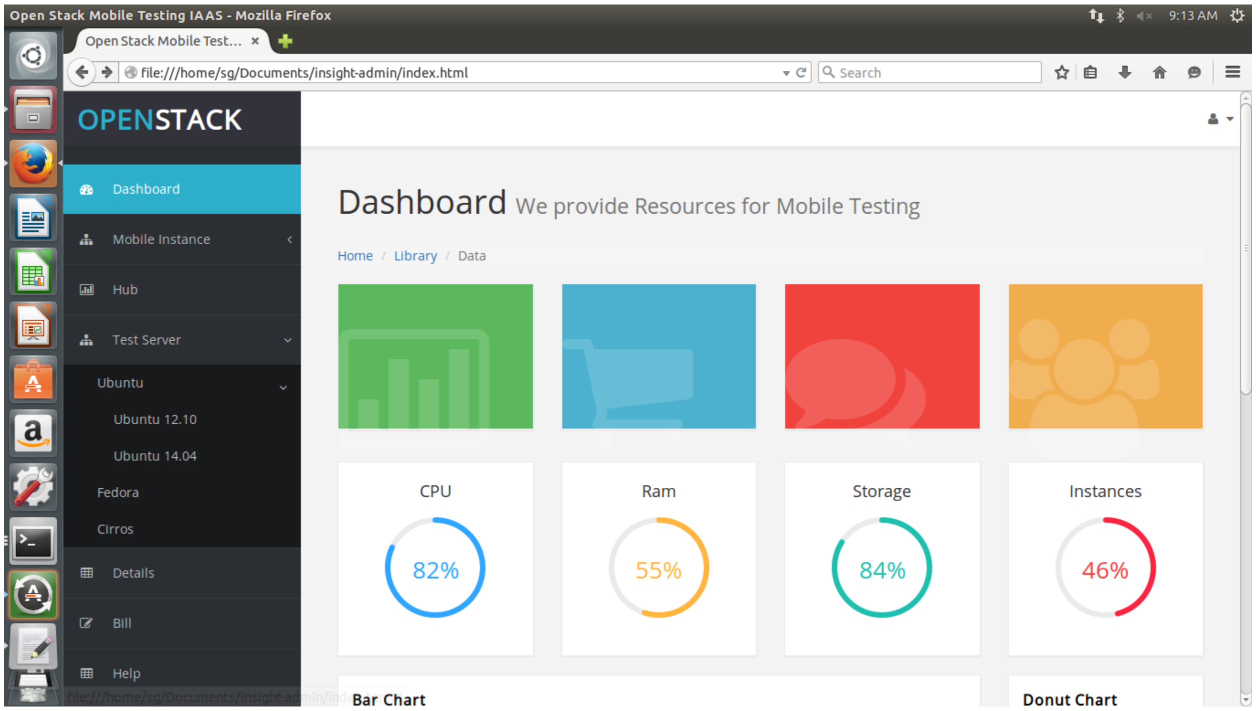
2. Various different tabs define various features of the project.



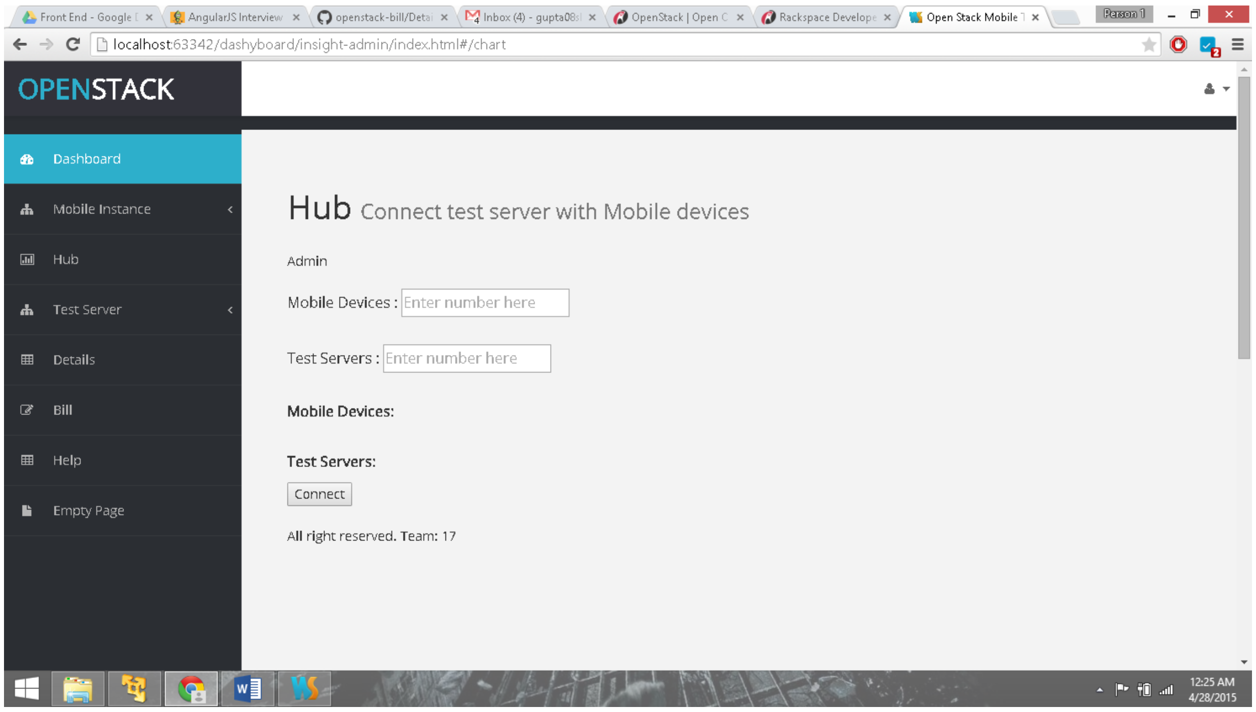
3. User can choose between various different versions of mobile instances of instances.



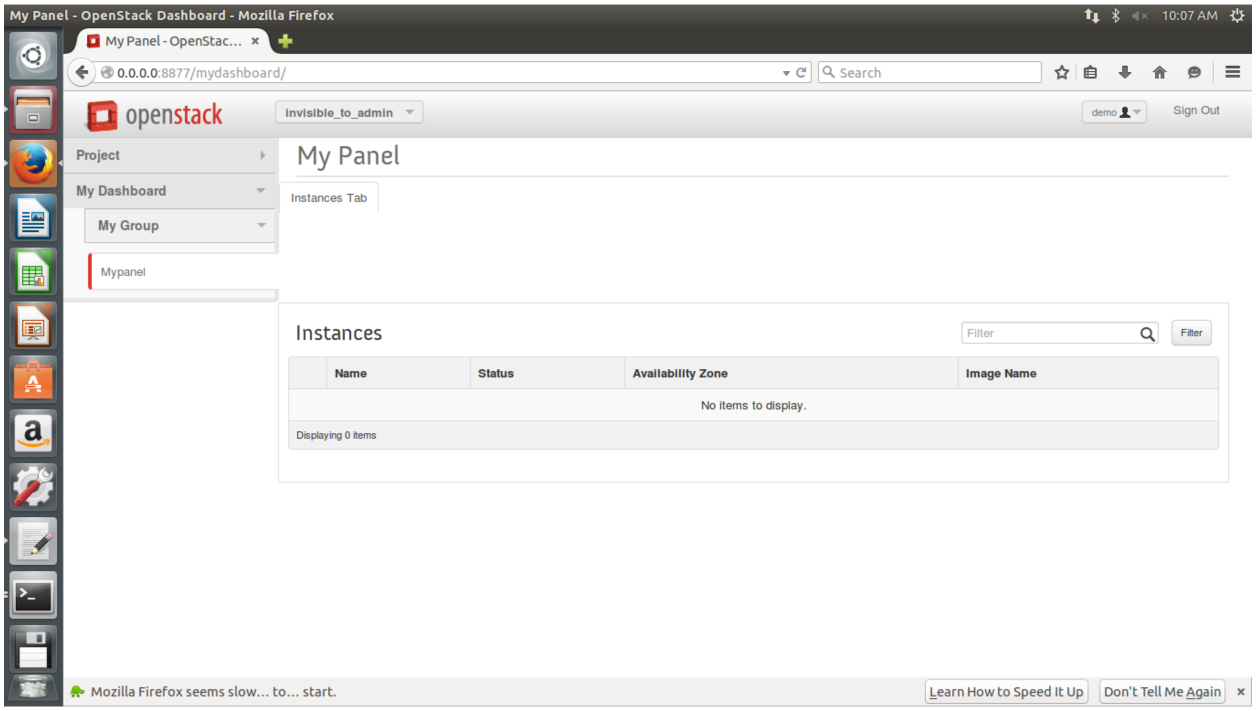
4. To launch test servers user can choose from different operating systems.



5. Hub allows user to connect mobile instances with the test servers.


localhost:63342/dashyboard/insight-admin/index.html#/chart
OPENSTACK
Dashboard
Mobile Instance
Hub
Test Server
Details
Bill
Help
Empty Page
Hub Connect test server with Mobile devices
Admin
Mobile Devices : Enter number here
Test Servers : Enter number here
Mobile Devices:
Test Servers:
Connect
All right reserved. Team: 17

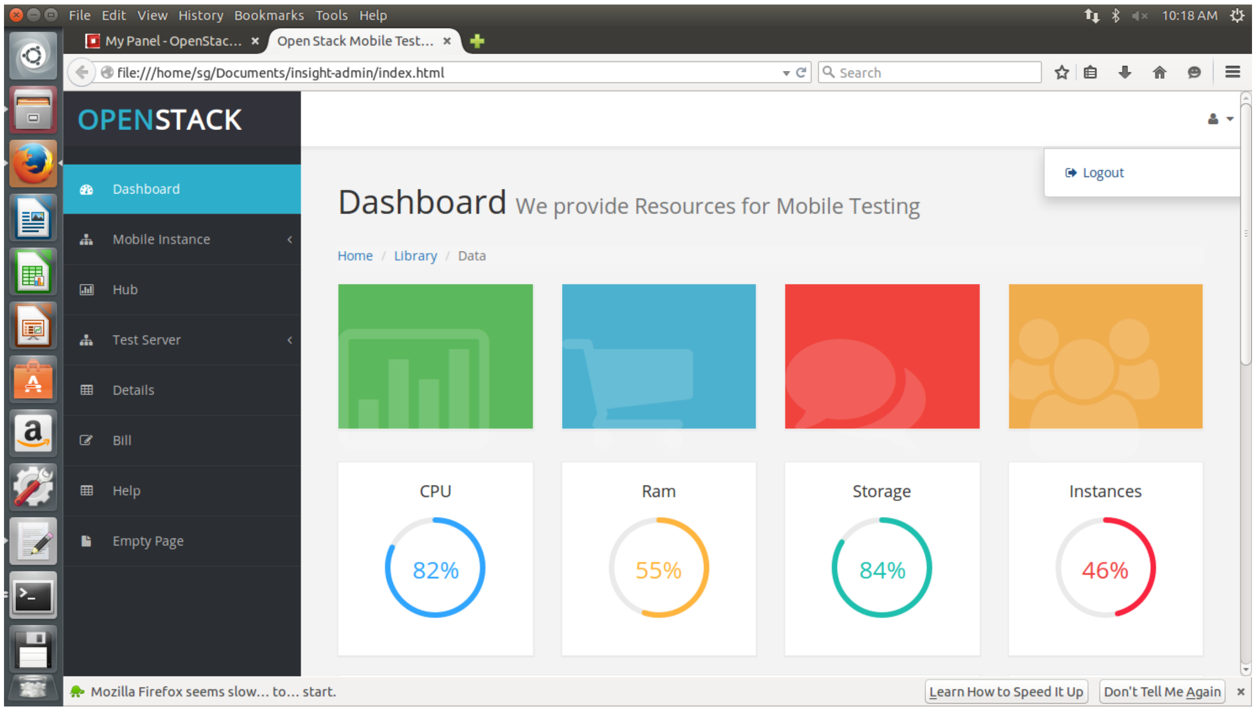
6. When user clicks the detail tab, he is directed to other webpage displaying the specifications of the instances.



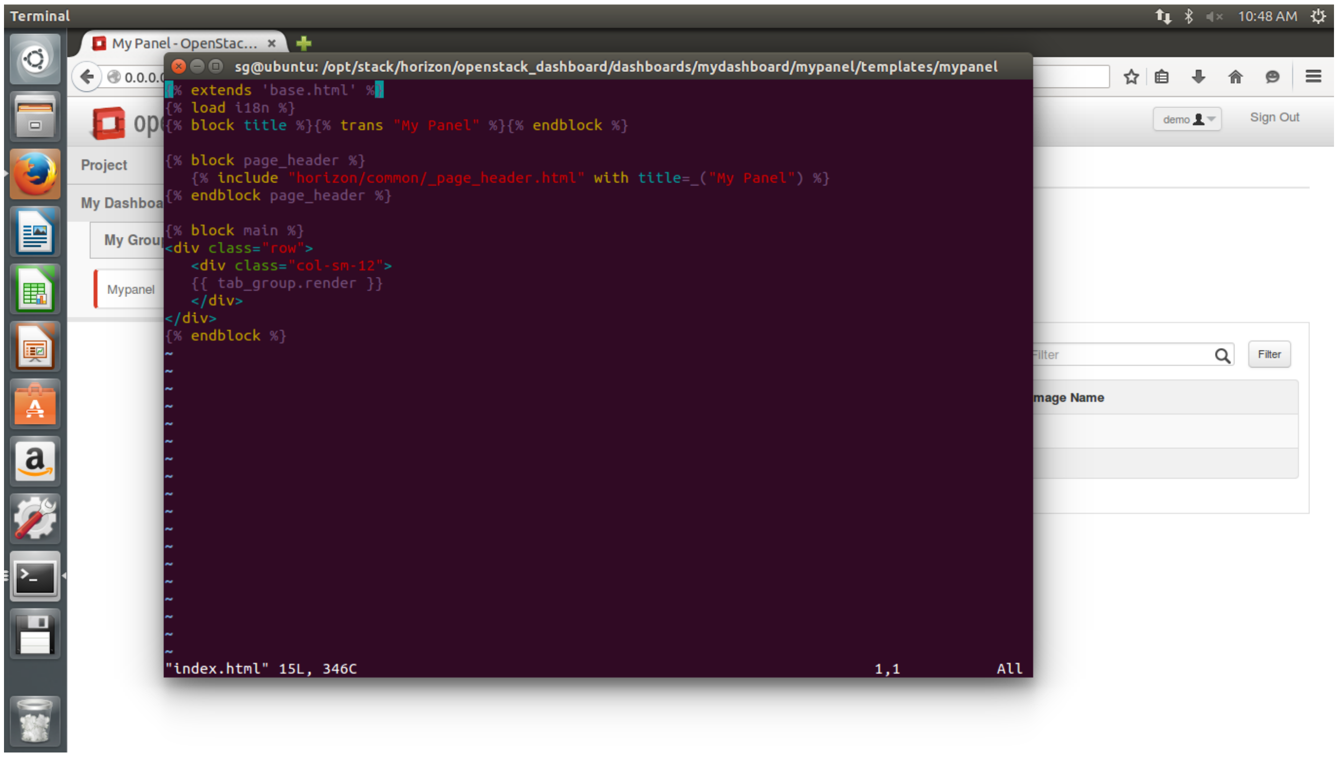
	Name	Status	Availability Zone	Image Name
No items to display.				

Displaying 0 items

7. Dashboard allows different user to Login and Logout and view their account details.


File Edit View History Bookmarks Tools Help
My Panel - OpenStac...
Open Stack Mobile Test...
file:///home/sg/Documents/insight-admin/index.html
OPENSTACK
Dashboard
Mobile Instance
Hub
Test Server
Details
Bill
Help
Empty Page
Logout
Dashboard We provide Resources for Mobile Testing
Home / Library / Data
CPU
82%
Ram
55%
Storage
84%
Instances
46%
Mozilla Firefox seems slow... to... start.
Learn How to Speed It Up
Don't Tell Me Again

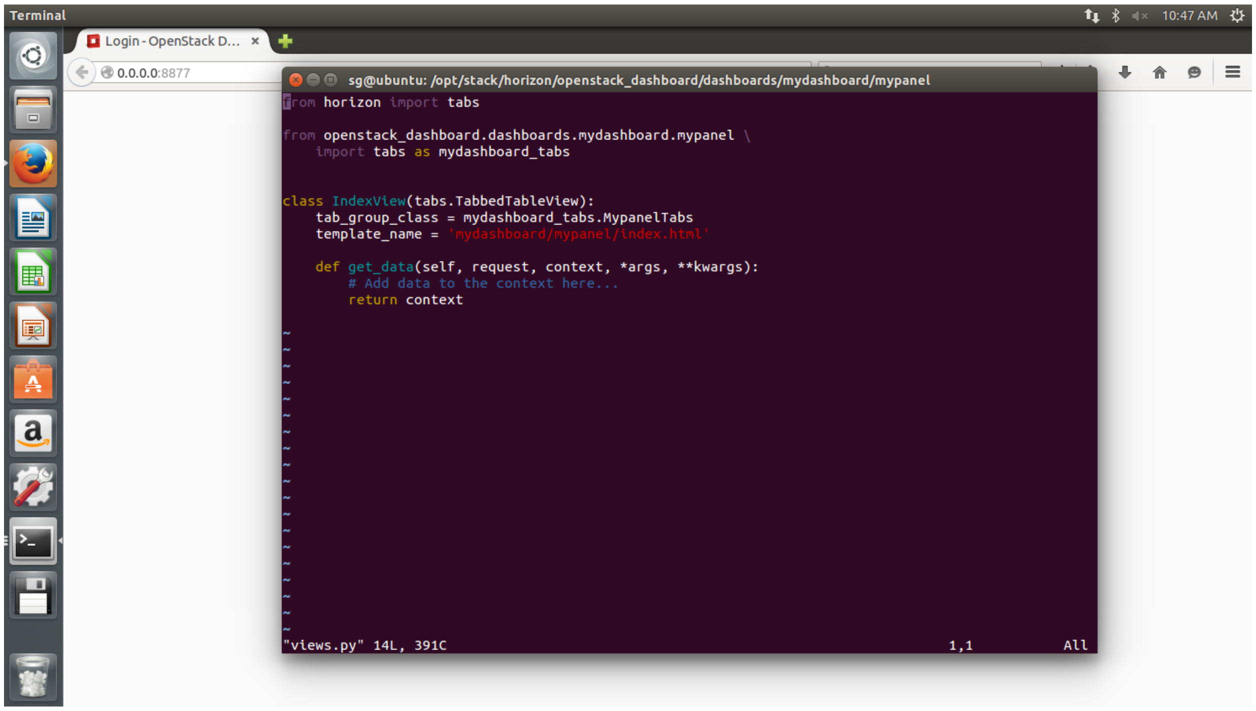
8. This is the index.html file, responsible for displaying different views on the dashboard.



sg@ubuntu: /opt/stack/horizon/openstack_dashboard/dashboards/mydashboard/mypanel/templates/mypanel

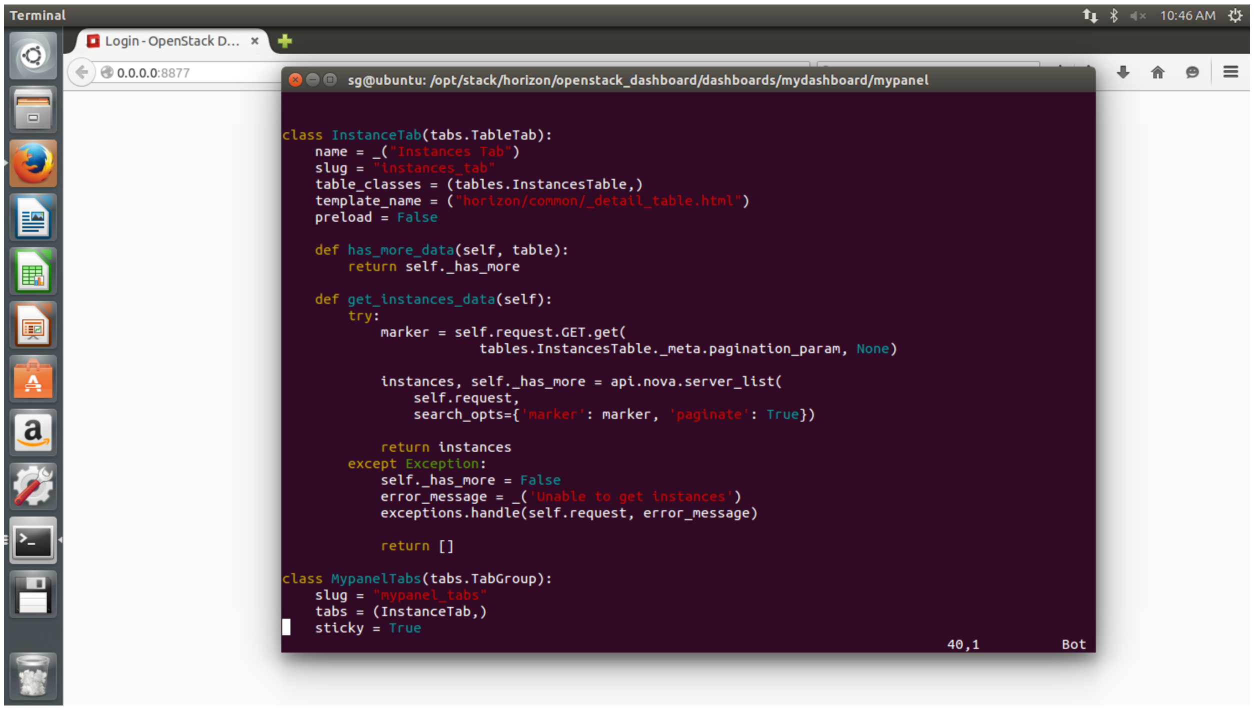
```
{% extends 'base.html' %}
{% load i18n %}
{% block title %}{% trans "My Panel" %}{% endblock %}

{% block page_header %}
  {% include "horizon/common/_page_header.html" with title=_("My Panel") %}
{% endblock page_header %}

{% block main %}
<div class="row">
  <div class="col-sm-12">
  {{ tab_group.render }}
  </div>
</div>
{% endblock %}
~
"index.html" 15L, 346C                                   1,1           All
```

9. The views.py returns the required data in the form of views to the index.html and hence displays the request to the user on the dashboard.



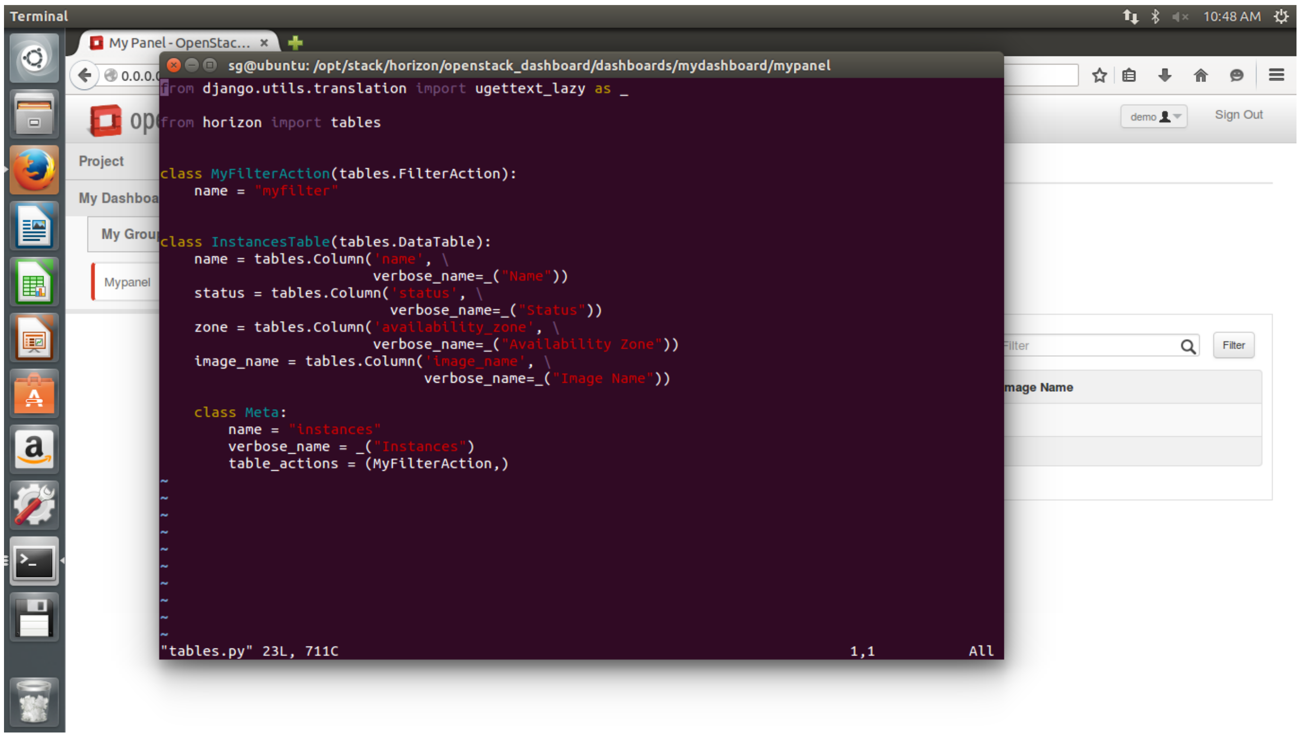
```
from horizon import tabs

from openstack_dashboard.dashboards.mydashboard.mypanel \
    import tabs as mydashboard_tabs


class IndexView(tabs.TabbedTableView):
    tab_group_class = mydashboard_tabs.MypanelTabs
    template_name = 'mydashboard/mypanel/index.html'

    def get_data(self, request, context, *args, **kwargs):
        # Add data to the context here...
        return context
"views.py" 14L, 391C                                          1,1           All
```

10. Tabs.py file manages the tabs in the navigation bar and communicates with different files, so that when a tab is clicked respective view is displayed on the dashboard.



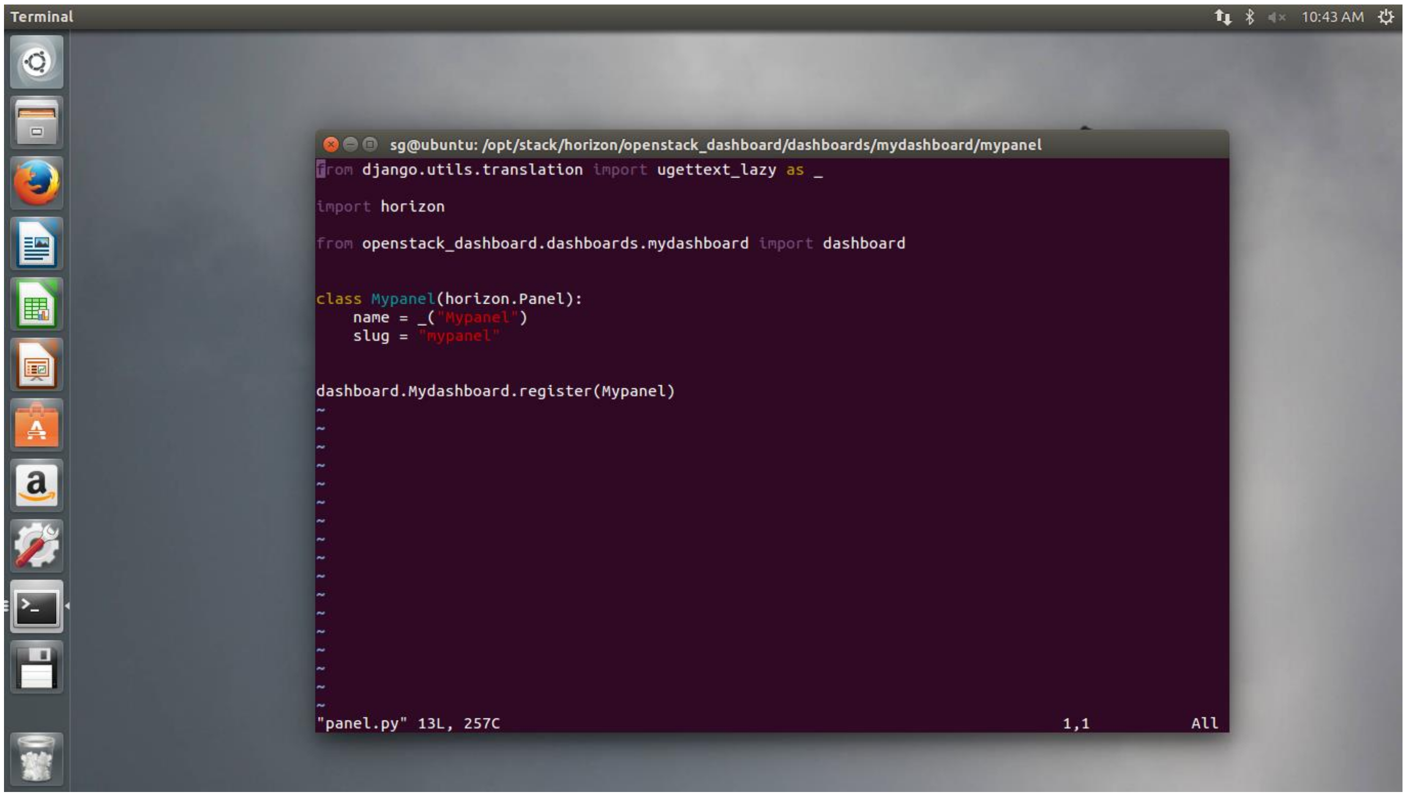
```
class InstanceTab(tabs.TableTab):
    name = _("Instances Tab")
    slug = "instances_tab"
    table_classes = (tables.InstancesTable,)
    template_name = ("horizon/common/_detail_table.html")
    preload = False

    def has_more_data(self, table):
        return self._has_more

    def get_instances_data(self):
        try:
            marker = self.request.GET.get(
                        tables.InstancesTable._meta.pagination_param, None)

            instances, self._has_more = api.nova.server_list(
                self.request,
                search_opts={'marker': marker, 'paginate': True})

            return instances
        except Exception:
            self._has_more = False
            error_message = _('Unable to get instances')
            exceptions.handle(self.request, error_message)

            return []

class MypanelTabs(tabs.TabGroup):
    slug = "mypanel_tabs"
    tabs = (InstanceTab,)
    sticky = True
                                                              40,1          Bot
```

11. This is tables.py file. This file handles the tabular view of the instances. It manages the details of instance and displays them on the dashboard.



```
from django.utils.translation import ugettext_lazy as _
from horizon import tables


class MyFilterAction(tables.FilterAction):
    name = "myfilter"

class InstancesTable(tables.DataTable):
    name = tables.Column('name', \
                         verbose_name=_("Name"))
    status = tables.Column('status', \
                           verbose_name=_("Status"))
    zone = tables.Column('availability_zone', \
                         verbose_name=_("Availability Zone"))
    image_name = tables.Column('image_name', \
                               verbose_name=_("Image Name"))

    class Meta:
        name = "instances"
        verbose_name = _("Instances")
        table_actions = (MyFilterAction,)
~
"tables.py" 23L, 711C                                         1,1           All
```

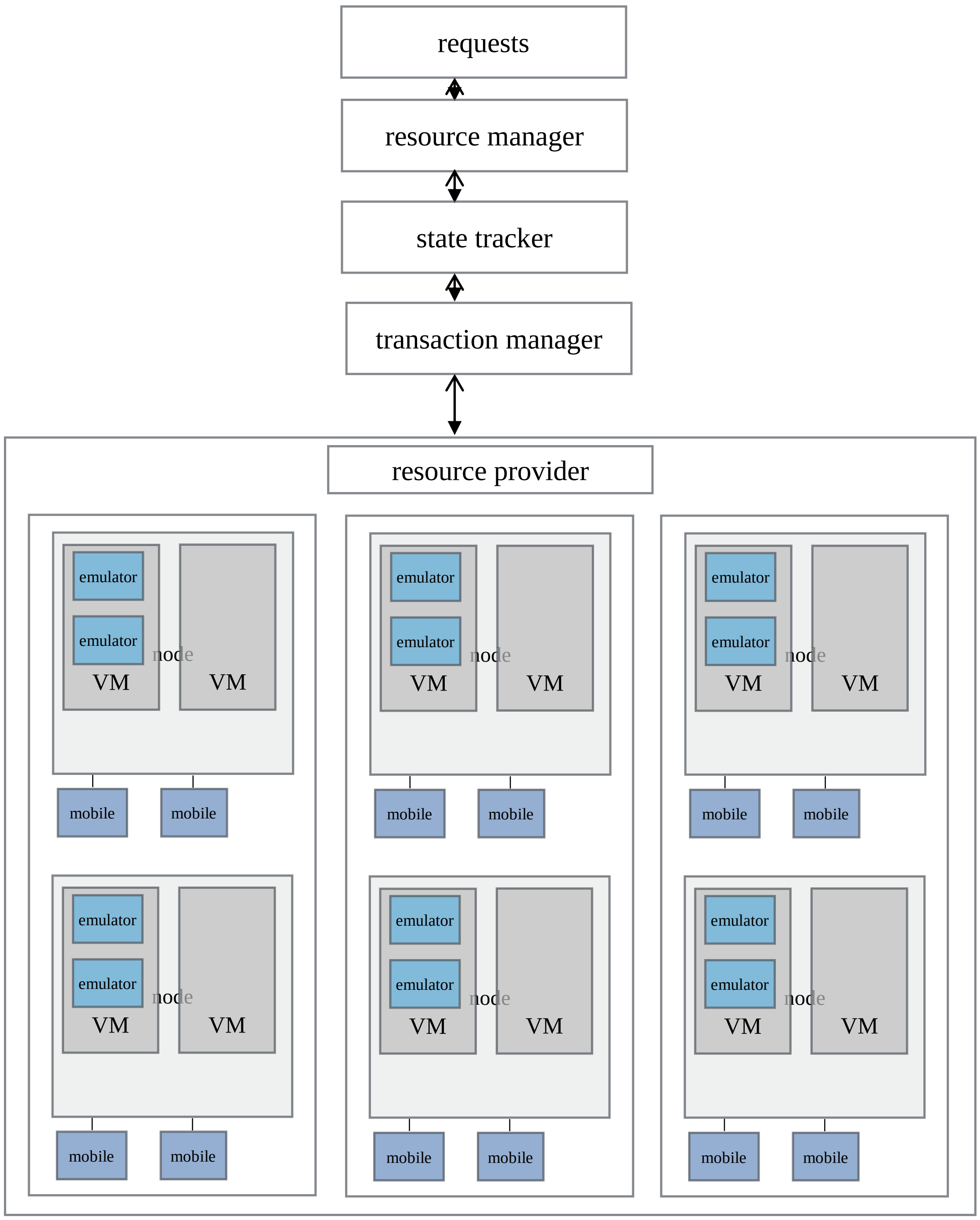
12. This code is written in panel.py file, which adds a new panel to the dashboard and is responsible for the user input to display the instance details.



```
from django.utils.translation import ugettext_lazy as _

import horizon

from openstack_dashboard.dashboards.mydashboard import dashboard

class Mypanel(horizon.Panel):
    name = _("Mypanel")
    slug = "mypanel"


dashboard.Mydashboard.register(Mypanel)
~
"panel.py" 13L, 257C                                          1,1           All
```

Resource Provision and Management

Resource management engine load requests from users. The resource manager of the engine keeps track of the status of all resources, and will make arrangement for the request. The arrangement is then executed by the transaction manger, to allocate/deallocate resources. Then



the records in state tracker will be updated. The architecture is shown in figure below.

Types of Resources

Four types of resources are considered in this system.

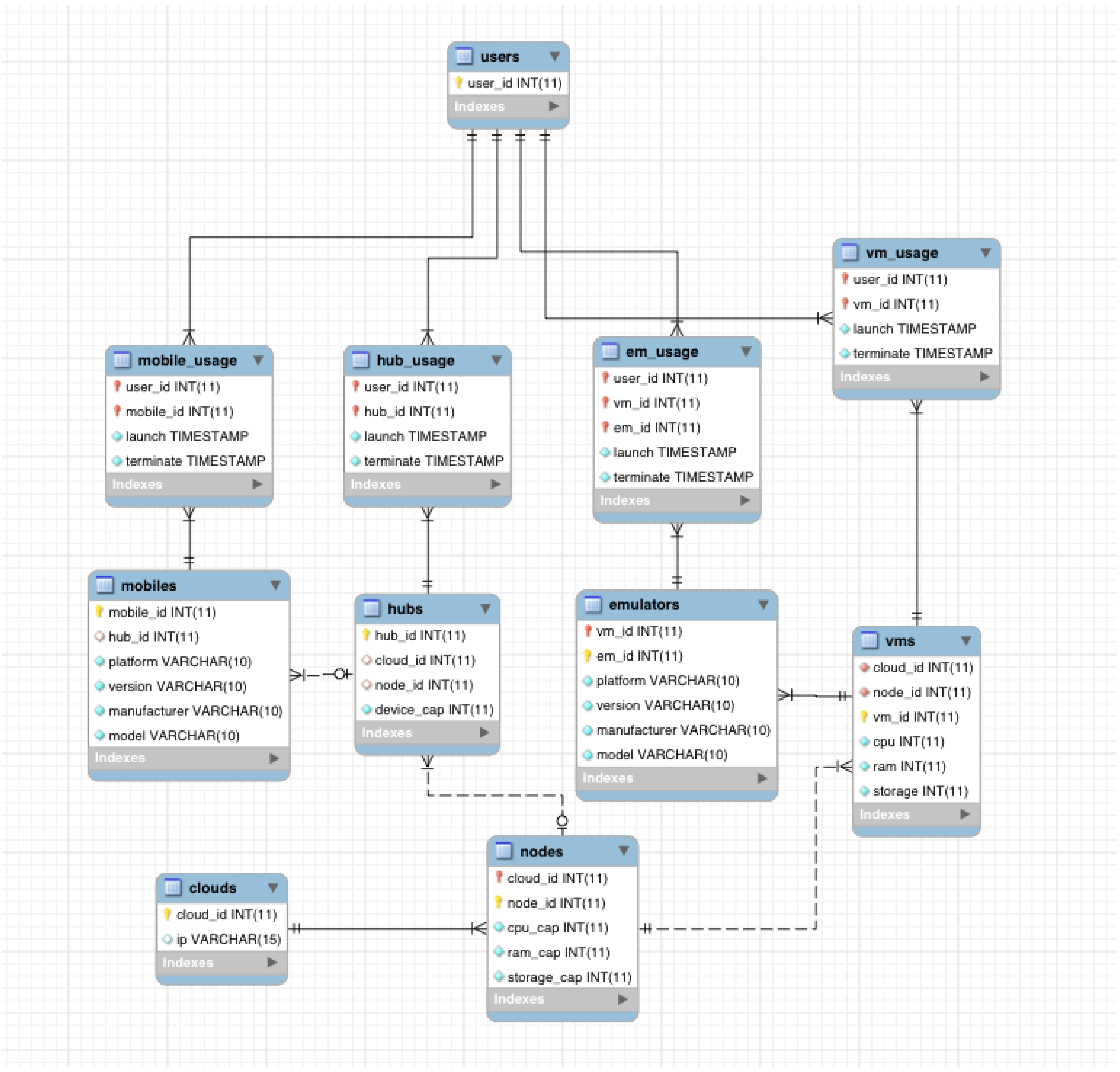
- VMs
  - VMs are hosted by computing nodes in clusters
  - Two types of VMs
    - system VMs - host emulators, managed by system
    - user VMs - serve as users' test servers, managed by users themselves
- Emulators
  - hosted in system VMs
  - the host system VM controls the accessibility of emulators to users
- mobiles
  - real mobile devices connected to nodes
  - once connected, not supposed to move frequently
  - managed by the controller program in host node
- hubs
  - real mobile hubs connecting nodes and mobile devices
  - each has a maximum capacity

The states are recorded by a database in the state tracker.

Layers of Relations

There are basically four layers of relations:

- Infrastructure Layer
  - Cloud - cluster of nodes, e.g Openstack
  - Node - a computing node, can host VMs
- Resource Layer
  - VM
  - Emulator
  - mobile device
  - hub - though devices and hubs are physical resources, kind of infrastructure, we consider them more as resources
- Usage Layer - the user behavior of occupying or releasing resources
  - VM usage
  - Emulator usage
  - mobile usage
  - hub usage
- User Layer
  - record active users, connected to the behavior of users


users
user_id INT(11)
Indexes
mobile_usage
user_id INT(11)
mobile_id INT(11)
launch TIMESTAMP
terminate TIMESTAMP
Indexes
hub_usage
user_id INT(11)
hub_id INT(11)
launch TIMESTAMP
terminate TIMESTAMP
Indexes
em_usage
user_id INT(11)
vm_id INT(11)
em_id INT(11)
launch TIMESTAMP
terminate TIMESTAMP
Indexes
vm_usage
user_id INT(11)
vm_id INT(11)
launch TIMESTAMP
terminate TIMESTAMP
Indexes
mobiles
mobile_id INT(11)
hub_id INT(11)
platform VARCHAR(10)
version VARCHAR(10)
manufacturer VARCHAR(10)
model VARCHAR(10)
Indexes
hubs
hub_id INT(11)
cloud_id INT(11)
node_id INT(11)
device_cap INT(11)
Indexes
emulators
vm_id INT(11)
em_id INT(11)
platform VARCHAR(10)
version VARCHAR(10)
manufacturer VARCHAR(10)
model VARCHAR(10)
Indexes
vms
cloud_id INT(11)
node_id INT(11)
vm_id INT(11)
cpu INT(11)
ram INT(11)
storage INT(11)
Indexes
clouds
cloud_id INT(11)
ip VARCHAR(15)
Indexes
nodes
cloud_id INT(11)
node_id INT(11)
cpu_cap INT(11)
ram_cap INT(11)
storage_cap INT(11)
Indexes

Lifecycle of a resource request

The typical lifecycle of a resource request in our system has these phases:

1. Generation
    vm request should include {os,cpu,ram,storage}
    emulator request should include {platform,version,manufacturer,model}
    mobile device request should include {platform,version,manufacturer,model}
    since hubs are used only to connect mobile devices and computing nodes, the request is automatically generated by default when user requires mobile devices.
2. Submission
    requests are kept in a queue stored in the resource manager
3. Processing
    The allocator will then search the resource states for available nodes.
    Different kinds of algorithms like least connection, simple match or ant colony
4. Allocation
    The chosen node will be allocated the desired resource for user.
    The usage will be recorded in the database, with the time of launching.
5. Deallocation
    The resource will be freed, while the usage in database will not be deleted, but set status to terminated, the time of termination will be recorded.

The python function to start a ubuntu test server virtual machine is like this:

```
def start_ubuntu():
        credentials = get_nova_credentials_v2()
        nova_client = nvclient.Client(**credentials)

        user_keystone = ksclient.Client(auth_url=<auth_url>,\
                username = "demo",password = "stack",tenant_name="demo")
        glance_endpoint = keystone.service_catalog.url_for(service_type="image")
        glance = glclient.Client(glance_endpoint,token = keystone.auth_token)
        imgs = glance.images.list()
        has_ubuntu=False
        for image in imgs:
                print(image.name)
                if name == "ubuntu"
                        has_ubuntu=True
        if has_ubuntu ==False
                upload_ubuntu("~/Downloads/trusty-server-cloudimg-amd64-disk1.img")

        print(nova_client.servers.list())
        image = nova_client.images.find(name="ubuntu")
        flavor = nova_client.flavors.find(name="m1.tiny")
        net = nova_client.networks.find(label="private")
        nics = [{'net-id': net.id}]
```

```
instance = nova_client.servers.create(name="vm2", \
            image=image, flavor=flavor, nics=nics)
print("List of VMs")
print(nova_client.servers.list())
```

The python function to start an emulator is like this:

```
home = expanduser("~")
sudoPassword = <pwd>
avd_name ='cloud_avd'

target_id='android-8'
abi_name='default/armeabi-v7a'

print home

android_sdk_dir = home+"/android"

os.chdir(android_sdk_dir)

android_sdk_source = android_sdk_dir+"/android-sdk-linux"

command = "echo no | " + android_sdk_source+"/tools/android create avd -n "+avd_name+" -t
"+target_id
print command
p = os.system(command)

command = android_sdk_source+"/tools/emulator -avd "+avd_name

print command
p = os.system(command)
```

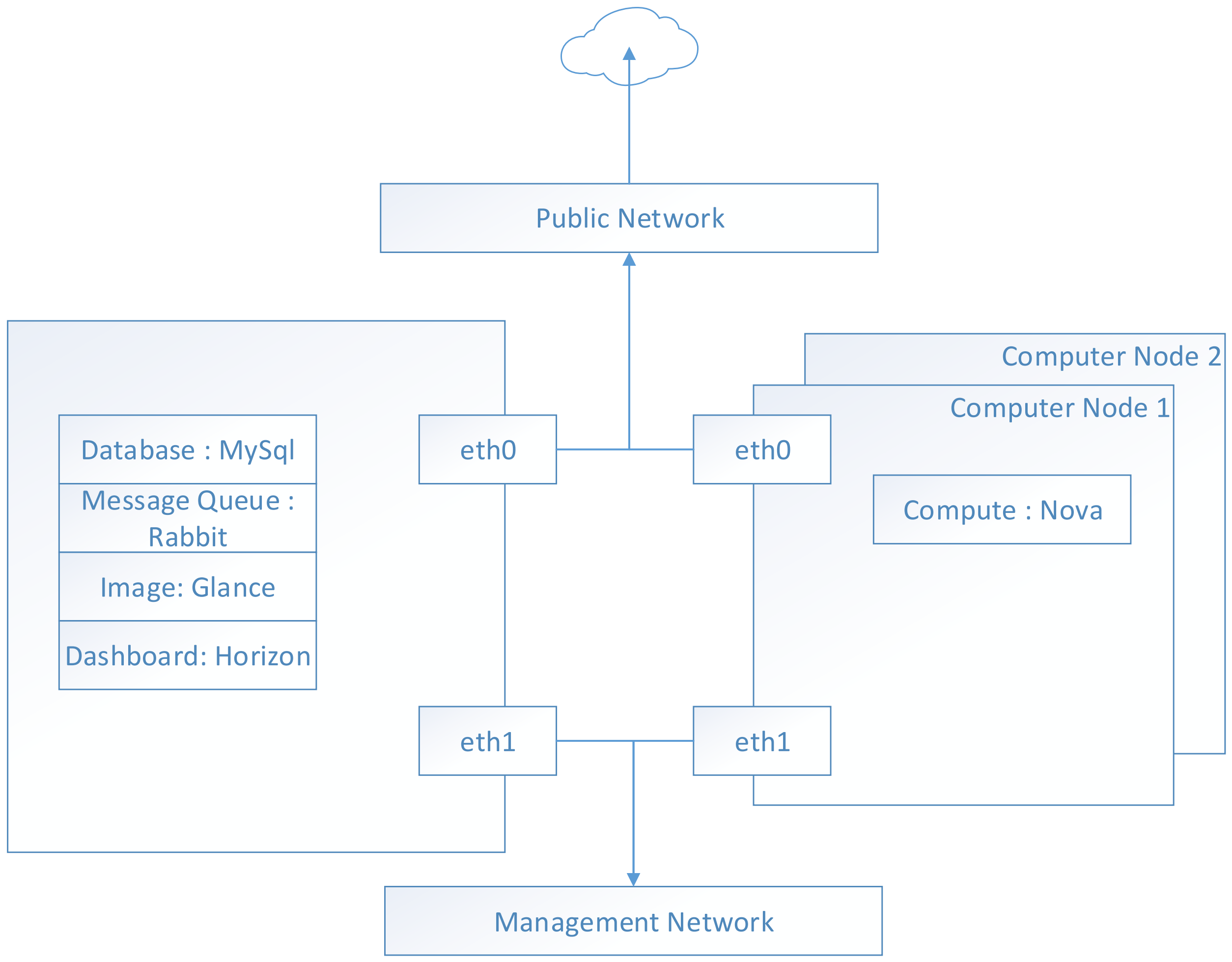
The transaction manager will call these functions to allocate resources, and corresponding deallocation functions to deallocate them.

The mobile devices and emulators are never moved in our cloud system, they are assigned to different users through virtual hubs, which connect them to the user's test server.
Take Android for instance, the user will send adb commands through the hubs to the emulators.

OpenStack Components on Multi Nodes

We install openstack on multinode with an architecture which has a single controller node and multiple compute nodes.

Control Node	Compute Node
Keystone Glance Cinder Horizon Neutron • Neutron Server Nova • Nova novncproxy • Novnc • Nova api • Nova scheduler • Nova conductor	Nova • Nova Compute Neutron • Neutron Openvswitch Agent



Configure multiple Compute nodes

To distribute your VM load across more than one server, you can connect an additional nova-compute node to a cloud controller node. You can reproduce this cd ../configuration on multiple compute servers to build a true multi-node OpenStack Compute cluster.

To build and scale the Compute platform, you distribute services across many servers. For a multi-node installation, you make changes to only the nova.conf file and copy it to additional compute nodes. Ensure that each nova.conf file points to the correct IP addresses for the respective services.

Control Node Configurations

Network Configuration

The control node has two Network Interfaces: One with network connectivity (eth0), and another internal network for Management use.

```
# vi /etc/network/interfaces

#External Network
auto eth0
 iface eth0 inet static
 address 192.168.47.200
 netmask 255.255.255.0
 gateway 192.168. 47.2

#Management Network
auto eth1
 iface eth1 inet static
 address 10.0.0.10
 netmask 255.255.255.0
gateway 10.0.0.2
```

For VM in VMPlayer we use

```
$ sudo vim /etc/network/interfaces

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5)
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
  address 192.168.47.200
  netmask 255.255.255.0
  broadcast 192.168.47.255
  gateway 192.168.47.2

dns-nameservers 192.168.47.2
```

Cluster Controller Configuration

The cluster controller runs all OpenStack services. Configure the cluster controller's DevStack in local.conf:

```
[[local|localrc]]
HOST_IP=192.168.47.200
FLAT_INTERFACE=eth0
FIXED_RANGE=10.4.128.0/20
FIXED_NETWORK_SIZE=4096
FLOATING_RANGE=192.168.47.128/25
MULTI_HOST=1
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=labstack
MYSQL_PASSWORD=supersecret
RABBIT_PASSWORD=supersecrete
SERVICE_PASSWORD=supersecrete
```

```
SERVICE_TOKEN=xyzpdqlazydog
```

Install MySQL server

OpenStack services require a database to store information. We will use MySQL as database back-end.

```
# apt-get install python-mysqldb mysql-server
```

When we need to add additional nodes, such as compute nodes or storage nodes, MySQL should start on all the interfaces, as the default is only for localhost.

Install Messaging server

OpenStack requires a messaging/broker service to communicate between its services. We will use RabbitMQ.

```
# apt-get install rabbitmq-server
```

Control Node

Network Configuration

```
# vi /etc/network/interfaces

#Management Network
auto eth0
 iface eth0 inet static
 address 10.0.0.20
 netmask 255.255.255.0
gateway 10.0.0.1

#Data Network
auto eth1
 iface eth1 inet static
 address 11.0.0.20
 netmask 255.255.255.0
gateway 11.0.0.1
```

Compute Nodes Configuration

The compute nodes only run the OpenStack worker services. For additional machines, create a local.conf with:

```
[[local|localrc]]
HOST_IP=192.168.47.201 # change this per compute node
FLAT_INTERFACE=eth0
FIXED_RANGE=10.4.128.0/20
FIXED_NETWORK_SIZE=4096
FLOATING_RANGE=192.168.47.128/25
MULTI_HOST=1
LOGFILE=/opt/stack/logs/stack.sh.log
```

```
ADMIN_PASSWORD=labstack
MYSQL_PASSWORD=supersecret
RABBIT_PASSWORD=supersecrete
SERVICE_PASSWORD=supersecrete
SERVICE_TOKEN=xyzpdqlazydog
DATABASE_TYPE=mysql
SERVICE_HOST=192.168.47.201
MYSQL_HOST=192.168.47.201
RABBIT_HOST=192.168.47.201
GLANCE_HOSTPORT=192.168.47.201:9292
ENABLED_SERVICES=n-cpu,n-net,n-api,c-sch,c-api,c-vol
NOVA_VNC_ENABLED=True
NOVNCPROXY_URL="http://192.168.47.201:6080/vnc_auto.html"
VNCSERVER_LISTEN=$HOST_IP
VNCSERVER_PROXYCLIENT_ADDRESS=$VNCSERVER_LISTEN
```

Install MySQL python library

On compute nodes we only need to install MySQL client and MySQL Python library.

```
# apt-get install python-mysqldb
```

Billing

Overview

In the project billing is added in two main components of OpenStack- Keystone and Horizon which are responsible for identity and dashboard service. The details are extracted from Nova which is the compute service of OpenStack.

As Horizon is the dashboard for OpenStack, billing is added to it. Horizon communicates with keystone and nova to extract the billing.

We will be storing the billing information for a tenant in the keystone database.

Bill of Tenants

Tenant is a group of user belonging to same company or project. Tenant's bill for each tenant will be stored for every month. After the first instance is launched by the user registered under a tenant an entry is made to database. Tenant can also view the usage and bill of each user under him.

Bill creation for Tenant

After the first instance is launched by the user registered under a tenant this bill is created.

This bill is regularly calculated and updated to provide a more accurate estimate to the admin.

At the start of each month, Horizon will trigger an operation by sending request to nova-api for previous month's usage of a tenant. After this Keystone will be requested to get unit cost of resources. Finally totally bill is calculated and updated for each tenant in DB.

The below sequence diagram explains this operation.

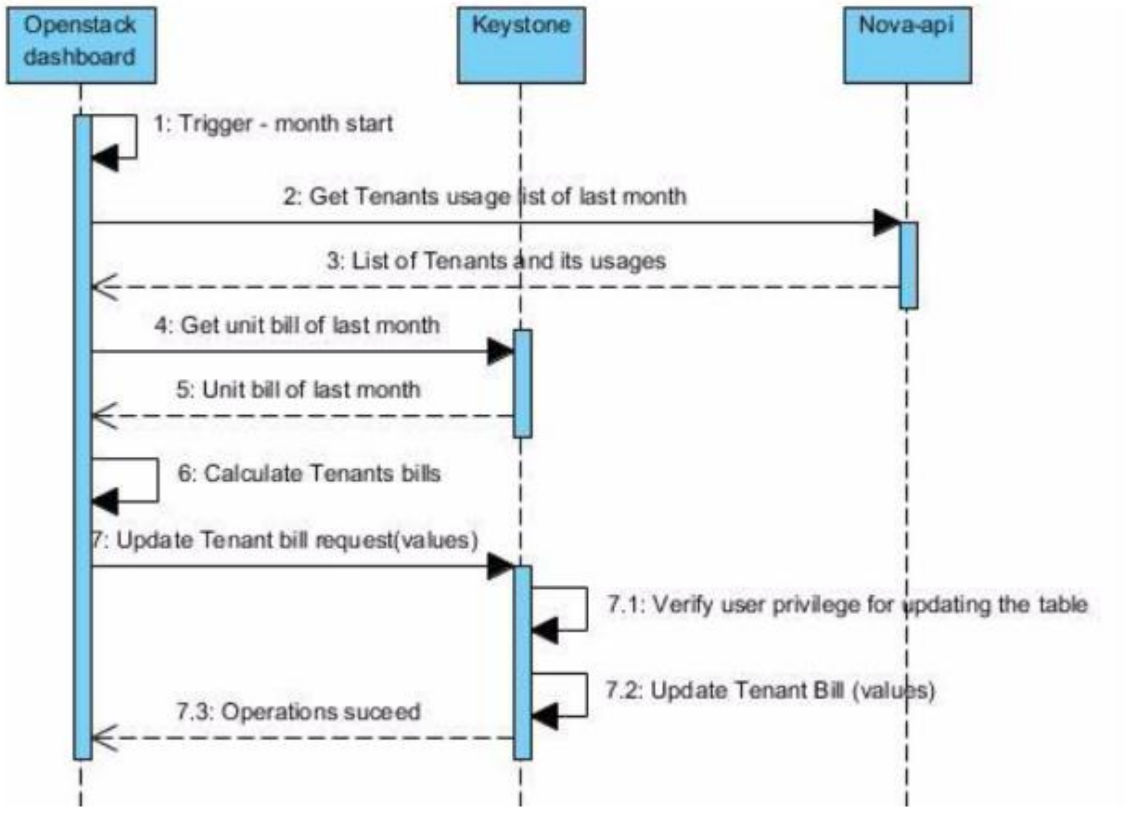

Openstack dashboard
Keystone
Nova-api
1: Trigger - month start
2: Get Tenants usage list of last month
3: List of Tenants and its usages
4: Get unit bill of last month
5: Unit bill of last month
6: Calculate Tenants bills
7: Update Tenant bill request(values)
7.1: Verify user privilege for updating the table
7.2: Update Tenant Bill (values)
7.3: Operations suceed

Fig. Tenant Bill creation sequence diagram