Local Cloud Computing Environment using Openstack and BDAS

Fall 2014

|  |
| --- |
| Technical Report  Service Oriented Computing (18-655) |

***Team***

Rao Li, Zilong Huang, Mohan Yang, Tongtong Bao, Yuwei Yang

***Advisor:***

Jia Zhang, Wei Wang

Table of Contents

1. Introduction
2. Motivation
3. Related Work
   1. XenServer
   2. Berkeley Data Analytic Stack (BDAS)
   3. Openstack
4. Original Design (using XenServer as hypervisor)
   1. System Design
   2. Setup methodology
   3. Challenges and possible solution
   4. Why we abandon this approach
5. New Design
   1. System Design
   2. Setup methodology
      1. General Openstack setting
      2. Sahara setting in Openstack
         1. Hadoop Setup
         2. Spark Setup
   3. Challenges and possible solution
6. Project Result, Conclusion and Future work
7. Contribution of each team member
8. ***Introduction***

This project aims at setting up a local cloud computing using ***Openstack*** and ***BDAS*** (Berkeley Data Analytic Stack).

With ***Openstack***, a local cloud computing environment could be deployed, so that web services like creating virtual machine on and creating cloud storage become possible in our local server.

Berkeley Data Analytic Stack is a software stack that provides an outline about what are the essential components for setting up a big data analysis cluster, components like Hadoop and Spark are necessary for Big Data analysis job.

So, we attempt to build a local cloud computing environment not only targeting at providing simple computational service, but also able to launch a big data analysis cluster that is with ***Hadoop*** and ***Spark*** installed and configured.

First, we start with ***XenServer*** as the hypervisor, which proved to be a very unclever choice. Though ***XenServer*** does very good in virtualization technology, it does not provide any Web Interface so that a pure ***XenServer*** could not be configured as a web service. ***Openstack*** could be installed together with ***XenServer***, so that a web service interface could be used to launch virtual machine on a ***XenServer***. However, there are many incompatibility between ***Openstack*** and ***XenServer***, so after putting much effort into this, we decide we should not continue investing into ***XenServer*** based ***Openstack***, which seems to be too costly.

Then, we switch to set up ***Openstack*** on ***Ubuntu Server***. Openstack has a relatively completed operational manual for using Ubuntu as the underlying hypervisor. But still it needs quite a lot extra configuration to have the cloud computing platform built with Hadoop, Spark, and also works with the existing network.

We have found out way to make Openstack works, but still there are problems that need more knowledge in Openstack to solve. In this project, we did learn a lot about how a cloud computing platform works, and how to set up one, it is a very precious experience for all the teammates. Setting up local cloud computing is not an easy job, but it really worth the effort, especially when Cloud computing becomes a trend of the new development in Computer Industry.

***2. Motivation***

There are several advantages of using a local cloud computing environment. It provides a limited accessibility like desktop computing, while also provides a combination of computing resource like cloud computing. Its security and convenience is what a public cloud service cannot match. There are three particularly important reasons why we want to build a local cloud computing environment for Carnegie Mellon University - SV campus.

**Local Resource can be conveniently and safely used**

There are many kinds of cloud computing services all around us these days, like Amazon Web Service (AWS) and Google Cloud Computing Services. They all provide highly reliable computing resources and user friendly UIs, but they all face one problem: If you want to do any kinds of data analyzing, you have to upload your original data up to the cloud. This is considered time consuming and inconvenient, especially when the amount of data needs to be processed is extremely large. As we are in the age of big data, situations like this are inevitable. On the other hand, although the public cloud services all provides storage services like Amazon Simple Storage Service (S3), we should also consider the security issue. These storage systems are considered reliable, but not totally trustworthy, especially when you are storing data with great importance. With a local cloud environment, however, these issue no longer exist. All CMU faculties and students can use the local cloud just like using desktop computer. The data can be placed safely and conveniently on the cloud side. Users can access their data conveniently and no longer need to waste time waiting for the data uploading to the cloud. Besides, security is no longer a problem. As the local cloud only grants accessibility to specific users, in this case, CMU students and faculties, the data stored in the cloud is highly secure.

**Pricing**

Local cloud is a low pricing cloud environment. After the universities have purchased the servers, everything else is free except for the charge of electricity. This is why a local cloud is so necessary, it is much more cheaper than using a public cloud service. For example, the AWS charges users in an interesting way, because it has so many charging areas. Many users may just notice the price of the instances, like the on demand price of m1.large is $0.175 and m1.small is $0.077, but they failed to realize the pricing of other services. For example, when you are using the AWS cluster to do a mapreduce job, several more charges besides the price of the instance should be considered, including the price of the AWS EMR (Elastic Mapreduce) cluster, the price of the I/O data, the price of the Auto Scaling Group, and the users may even have to pay for the S3. The combination of all these prices can be considerably large. However, when using the resource of the local cloud computing environment, everything is totally free. All we have to do is to purchase several servers and the cloud environment is ready to service for several years. Besides, as the universities already got these servers, there’s no reason to not make use of them

**Local Cloud Service can provide customized service**

Public cloud services are convenient and fast, but compared to local cloud computing environment, it cannot provide so much customized service. Using a local cloud service, we can adjust almost everything from the operating system to the software installed in it. The public cloud service, like AWS, do provide many kind of different customized AMIs (Amazon Machine Image). But to satisfy the need of the most, most of the AMIs only installs the most basic tools and software, it is hard for a user to customize the image. If a user had to make a customized version of AMI, he or she had to create a snapshot, which would be charged quite a high price. With local cloud computing environment, however, users are allowed to modified the image or upload his own customized image in order to acquire a customized service. This is considered to be a marked advantage of setting up a local cloud environment.

1. ***Related Work***

This project has utilized many different software components available to build a local cloud computing system. Key components related are Citrix XenServer, Openstack and Berkeley Data Analytic Stack. The project could not be done without these related works (components).

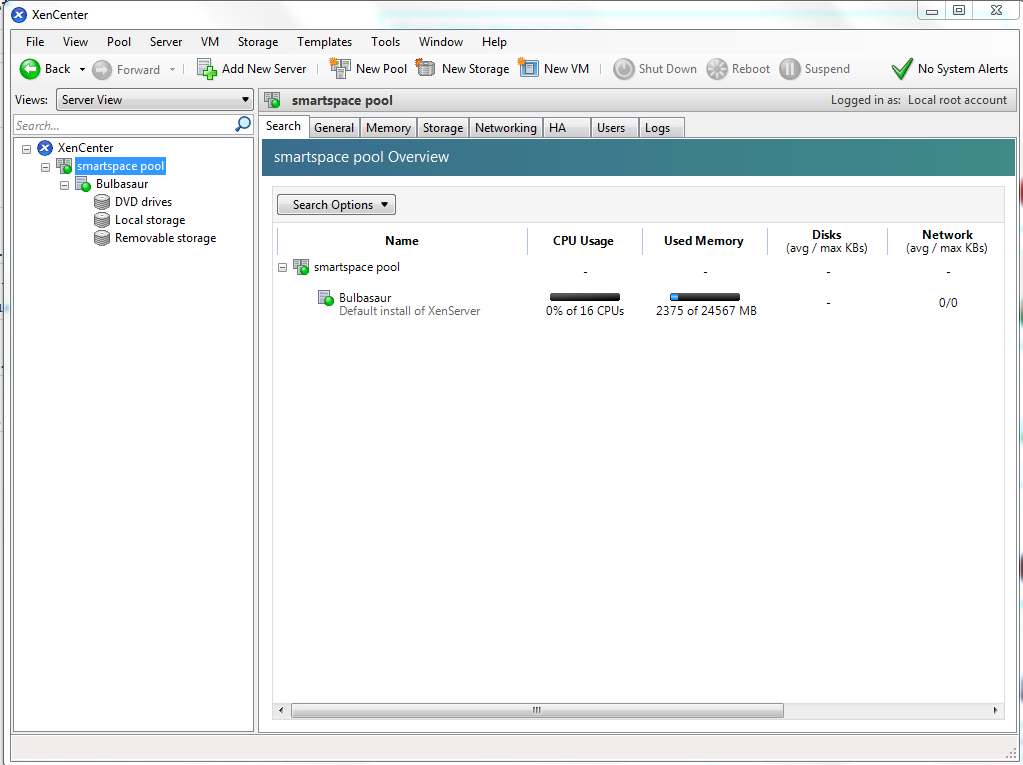
***3.1 XenServer***

Xen is a hypervisor using a microkernel design, providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently.  
It is capable of spawning a new virtual machine on the server, creating new storage repository on the disk, providing virtual network bridging service and all other services for a virtual machine to work normally as a virtual server on a real physical server.

XenServer is a virtualization platform based on the Xen hypervisor, with virtualization management tools like XenCenter, a centralized XenServer administration interface, performance alerting and reporting, host failure protection, virtual disk management, and resource pooling.

***3.1.2 XenCenter***

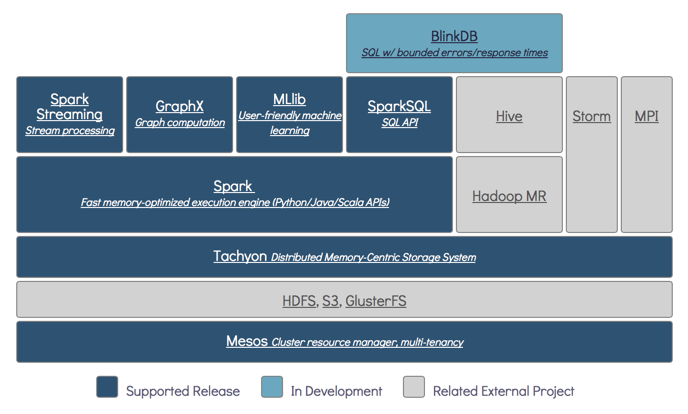
A very useful, although Windows-only, GUI that provides easy access to the most commonly used XenServer functionalities. However, more complicated operations still need to be done on the command line console, like IP allocation, and Openstack integration.

  
***Figure 3.1 The current view of the XenCenter cluster***

We had a pool of virtual machine with Pokemon theme naming managed by XenServer in the SmartSpaces pool. The entire pool has been reset by accident, and currently only Bulbasaur server is managed by XenServer. Pikachu and Charmander have been converted to pure Openstack on Ubuntu servers.

***3.2 Berkeley Data Analytic Stack***

Berkeley Data Analytic Stack (BDAS) is a core part of the whole construction of the local cloud computing infrastructure, it is an open source software stack that integrates software components being built by the AMPLab to make sense of Big Data.



***Figure 3.2 the architecture of Berkeley Data Analytic Stack,   
Source: The AMPLab Website*** [***https://amplab.cs.berkeley.edu/software/***](https://amplab.cs.berkeley.edu/software/)

It is more a concept than actual ‘software’. Any machine installed with components described in the architecture could claim itself a Berkeley Data Analytic Stack. The key components are Hadoop and Spark. Other components like Mesos could be substituted any cluster resource manager that provides similar functionality the Mesos does.

***3.2.1 HDFS***

The hadoop distributed file system(HDFS) is a distributed file system designed to run on commodity hardware. It is highly fault-tolerable and designed to be deployed on low-cost hardware. It can provide high throughput to application data and it is especially suitable for very huge dataset. The HDFS lays on the lower part of the BDAS structure, providing services to upper layers including hadoop and spart. Both the mapreduce job and streaming job performed by hadoop and spark depend on the HDF.

***3.2.2 Apache Hadoop***

Apache Hadoop is an open-source software framework for distributed storage and distributed processing of Big Data. It is especially suitable for data processing job of cloud clusters. It’s data processing job can be divided as two parts: hadoop streaming job and mapreduce job. Hadoop streaming jobs uses a jar file provided by hadoop. The users need to provide the mapper and reducer to run the job, all the data processing job should be contained in the mapper and reducer. The command is like below:

|  |
| --- |
| $ Hadoop jar hadoop-streaming-2.5.0.jar -input /xxx -output /xxx -mapper “java -jar xxx.jar” -reducer “java -jar xxx.jar” |

The hadoop mapreduce job, however, does not need the hadoop streaming jar file. It highly relies on the hadoop distributed file system. All input should come from HDFS and all output should be stored in HDFS. The user has to create their own mapreduce jar file and use the following command:

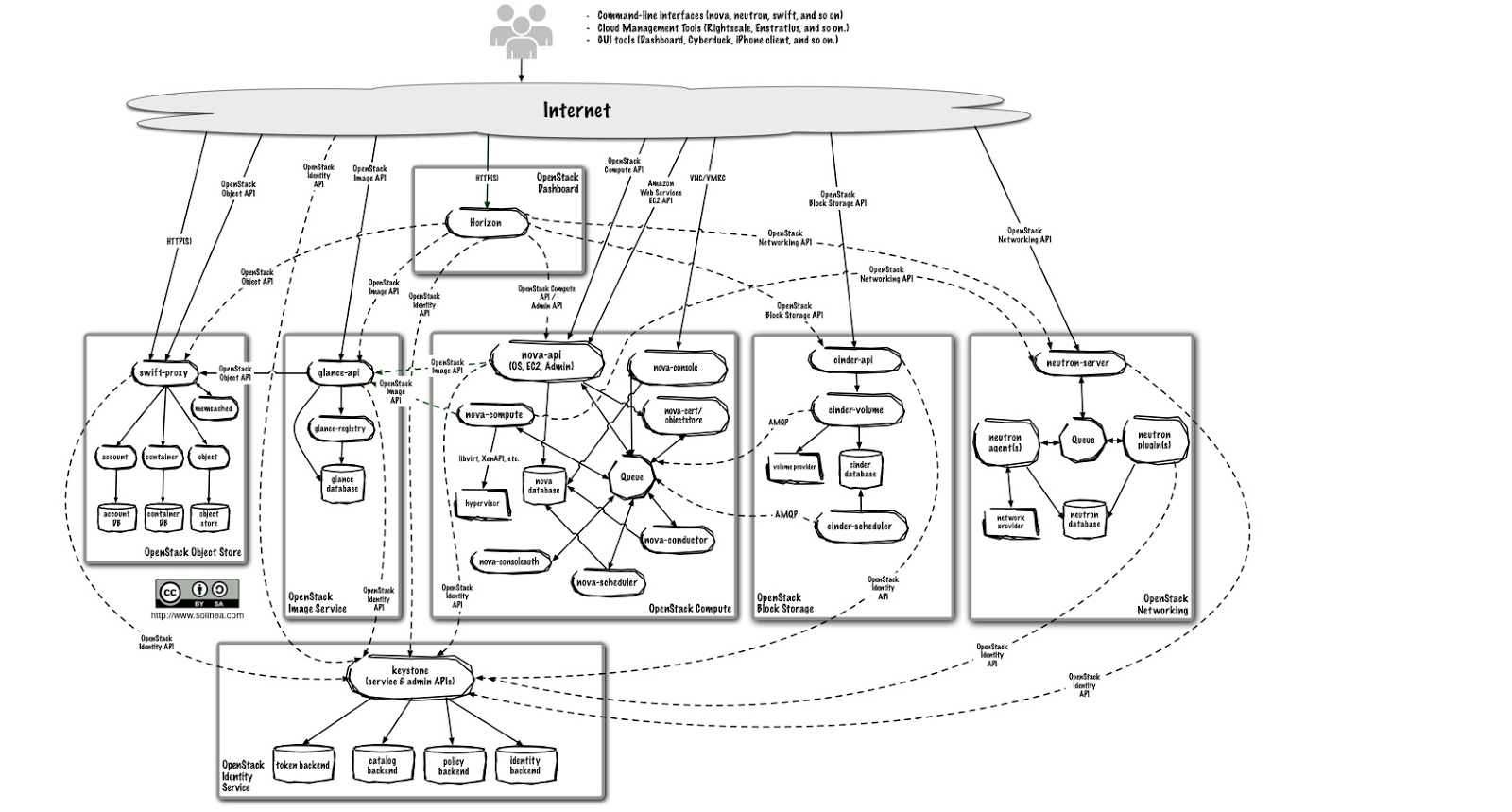
|  |
| --- |
| $ Hadoop jar xxx.jar |

***3.2.3 Apache Spark***

Apache Spark is an framework very similar to hadoop. The difference is that it uses in-memory computing technique so that it can run much more faster than hadoop, nearly 100 times faster according to the official data. However, limited by the in-memory space, spark cannot deal with very large amount of data. It is usually use to do the data processing for the relatively smaller amount of data.

***3.3 Openstack***

Openstack is a free and open-source software cloud computing software platform. In our final setup, we deployed it on Ubuntu as our infrastructure as a service (IaaS) solution. As can be seen from the below diagram, openstack has many services, and each is responsible for a different component of an IaaS solution.[1] Openstack can be either installed with XenServer, or standalone on Linux directly. Either way, we used Devstack to manage and handle all these different components in our OpenStack installation.

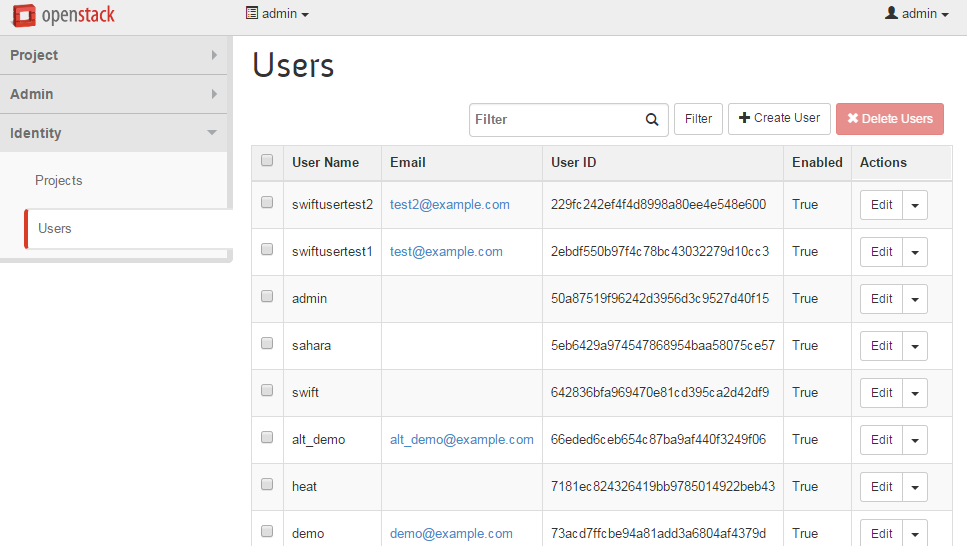
 ***Figure 3.3 Openstack logical architecture.  
 Source: http://docs.openstack.org/admin-guide-cloud/content/logical-architecture.html***

***Horizon - Dashboard Web UI***

Horizon is a web application that provides an easy to use web interface to administer on the Openstack server. Features visible on the dashboard depends on the components installed - for example, if Sahara is not installed, there will be no “Data Processing” section displayed on the dashboard. In addition, the dashboard has implements user access control (using Keystone), and Administration and Identity management tabs are not visible to a normal user. It’s also extensible by developers.

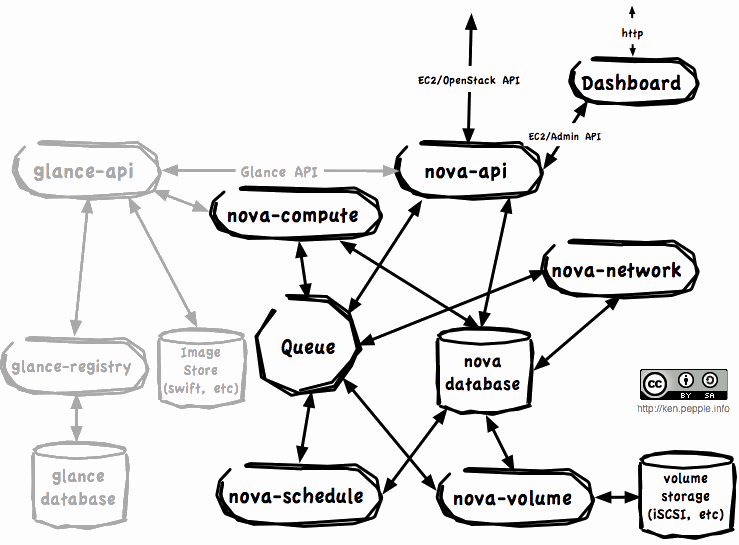
***Keystone - Identity***

Keystone is Openstack’s default security service. It handles all authentication and authorization in Openstack, and communicates with all components to make sure that there is proper authentication for actions and the user has the proper roles for the action. It also manages private and public keys in the system.

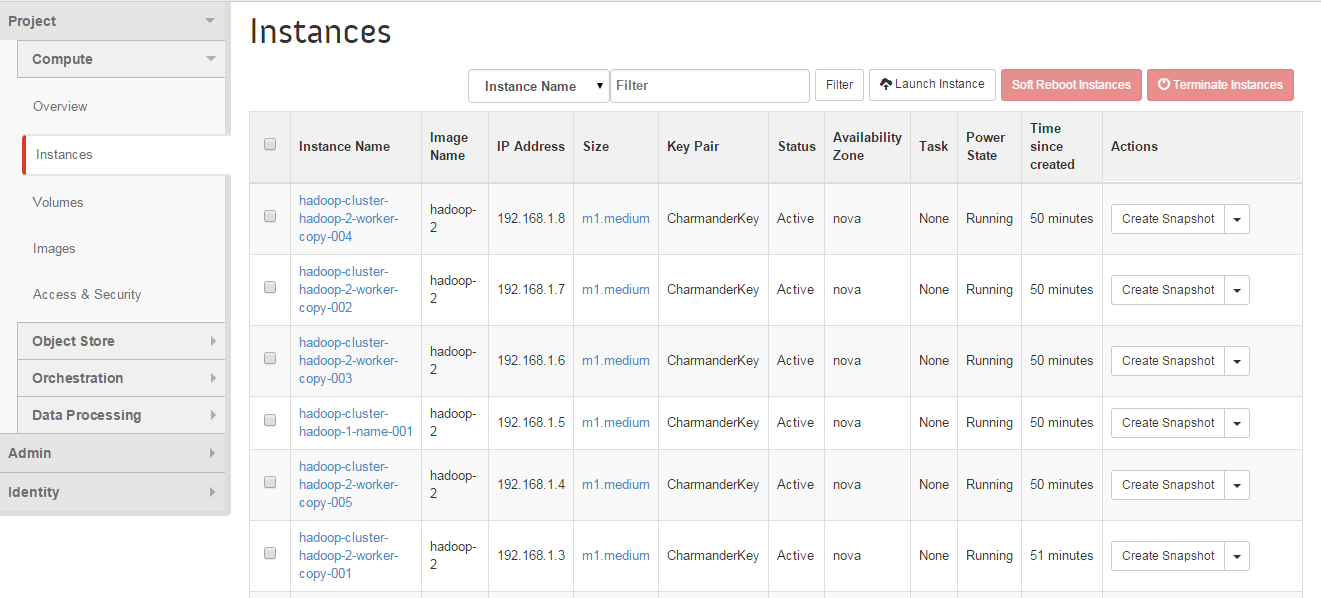
  
***Figure 3.4 User management in the dashboard***

***Nova - Compute***

Nova is the component that manages the lifecycle of computing instances, i.e. virtual machines. The API service responds to end user service calls, and it supports the OpenStack Compute API, AWS EC2 API, and Admin API, and the API is how the dashboard operates on Nova. The Nova Compute service creates instances through the hypervisor that OpenStack is installed on. If it’s installed on XenServer, it’ll use the Xen API, and on our (final) setup, it uses QEMU. It also has scheduling and limited networking capabilities (nova-network).



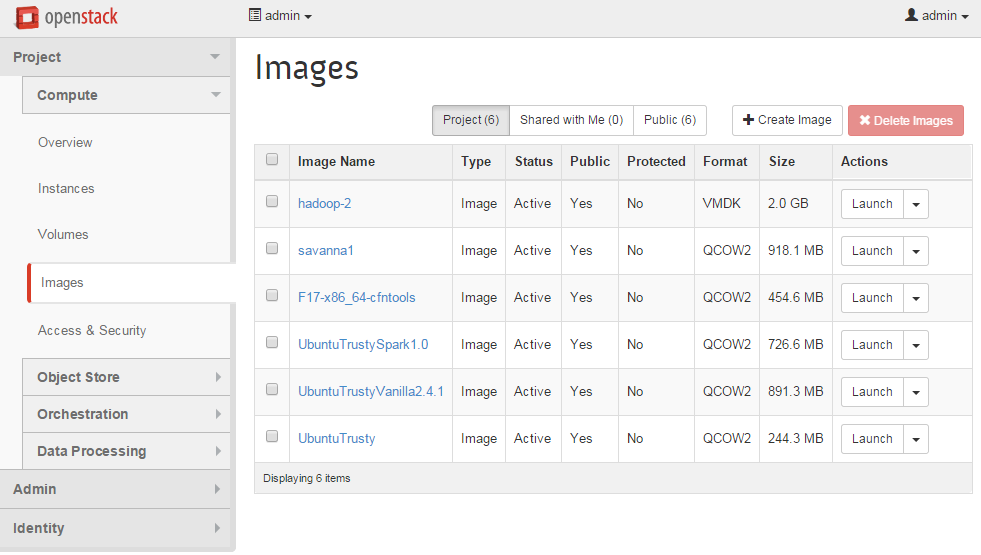
***Figure 3.5 Subcomponents of Nova   
Source:*** [***http://ken.pepple.info/openstack/2011/04/22/openstack-nova-architecture/***](http://ken.pepple.info/openstack/2011/04/22/openstack-nova-architecture/)



***Figure 3.6 Nova managed instances***

***Glance - Image Registry***

An image is used to boot a virtual machine, and every images available for Openstack to boot a virtual machine needs to be registered in Glance. The image itself is stored in Cinder Volume Services and Swift Object storage. It has periodic processes like replication services to ensure consistency and availability.

  
***Figure 3.7 Image Registry***

***Cinder - Block Storage***

Every virtual machine needs a volume as its virtual hard disk to operate normally, and Cinder is the component that provides such virtual hard disk service. It provides an infrastructure for pooling and managing volumes, snapshots, types, and interacts with Nova to provide instances with volumes.

***Swift - Object Storage***

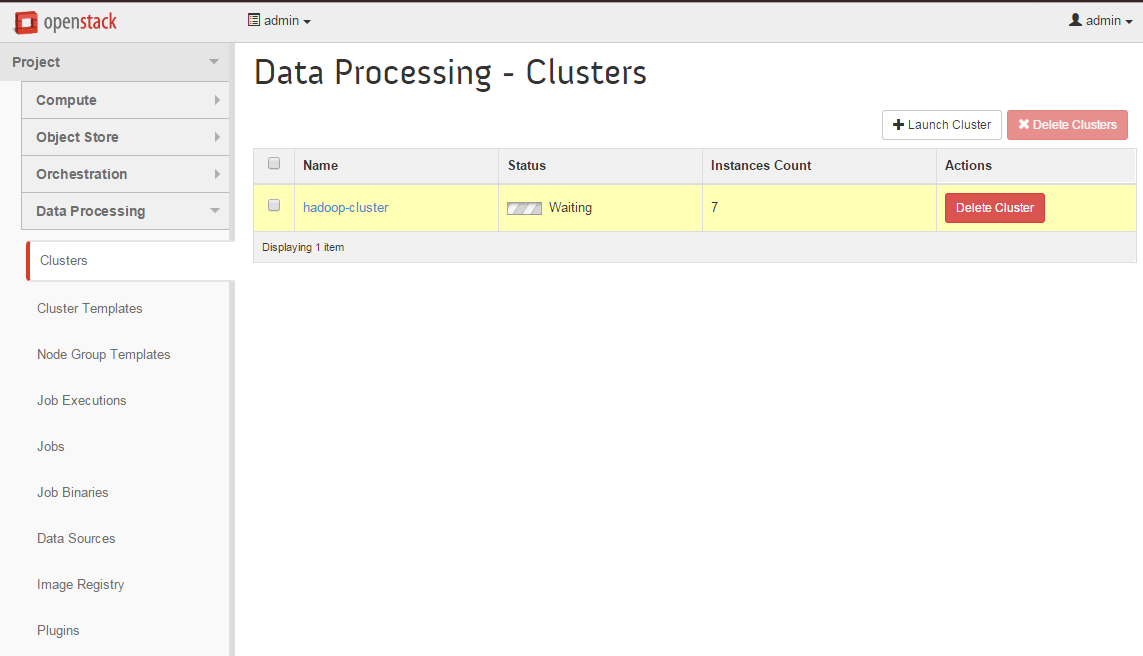
Swift provides object storage service. You can create a container in Swift, and then upload files into the container. Therefore when you launch a new instance, the instance can access the files stored in Swift easily. It is especially convenient when you are going to do data processing, the input file could be uploaded into the container as well as the executable binary file. Thus the cluster could load the input from the Swift and produce output into the Swift service. It’s highly scalable and you can store large amounts of unstructured data.

***Neutron - Advanced Networking***

It is a component for a more flexible and customized network configuration. Nova’s built-in network service is usable, but Neutron provides much more advanced network topology control. If it’s used, Neutron will communicate with Nova with its API, instead of Nova using nova-network. However, because it’s so highly customizable, it’s a lot harder to work with than nova-network.

***Heat - Instances Orchestration***

When you are going to launch multiple instances, you are going to need this. It provides a way to launch multiple instances according to a template where the configurations of the instances are recorded. This is used by Sahara to launch clusters.



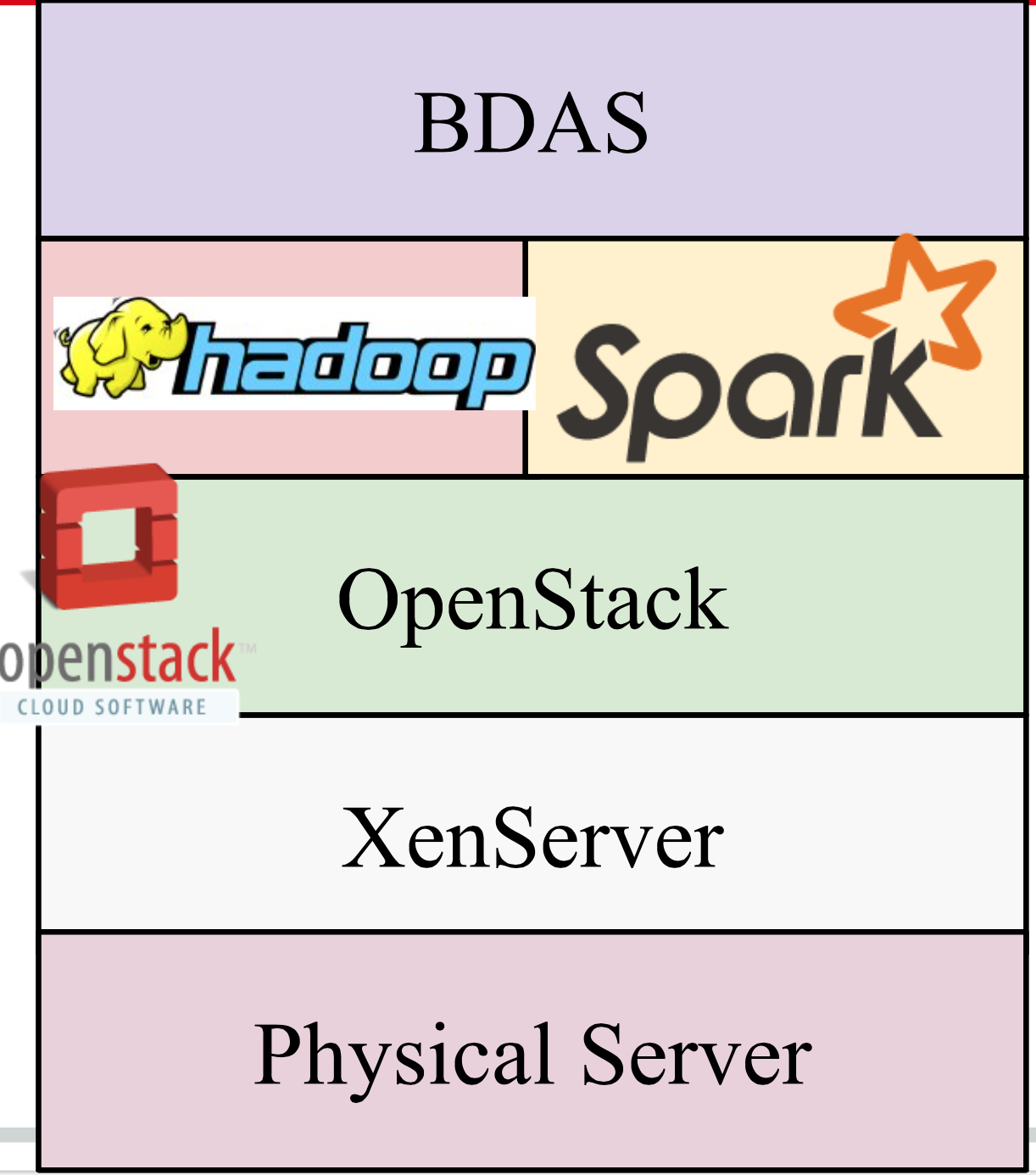
***Figure 3.8 A data cluster launched by Sahara, using a template.***

1. ***Original Design***

At the very beginning, because some of the faculty members are using XenServer as their simple virtualization platform, so we start to look for the possibility to integrate current XenServer and Openstack to provide a local cloud computing service as a local web services. However, this proves to be very difficult because it will need a lot of manual configuration. Nevertheless we did learn a lot about how Openstack in this early stage.

***4.1 System Design***

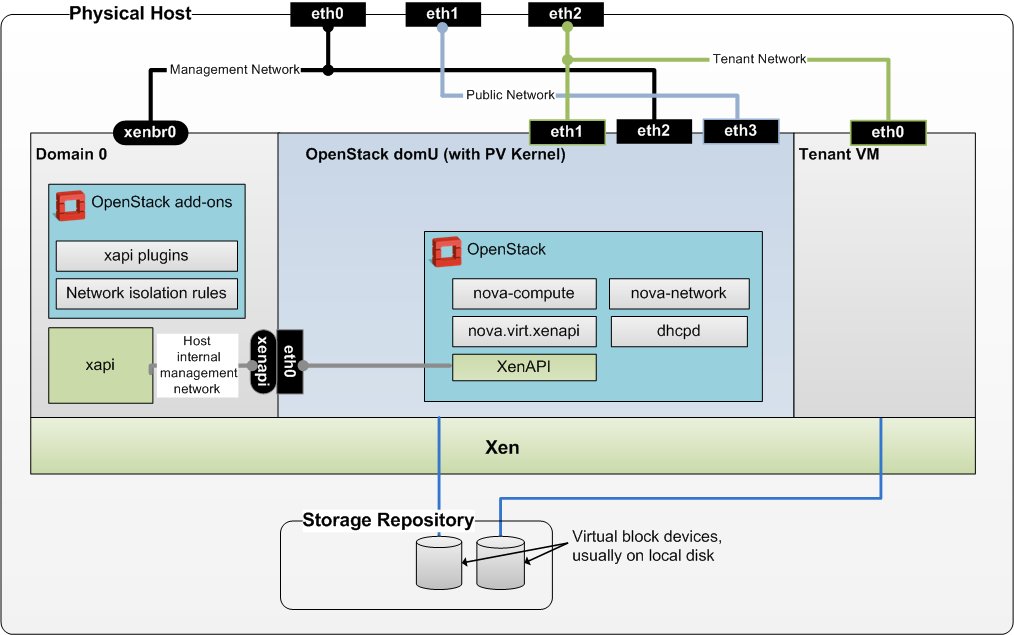
The original stack designed is as figure 4.1.



***Figure 4.1 The original design of architecture of our local computing Service***

In this original design, we plan to install Openstack based on XenServer, and then setup the hadoop platform together with Spark on top of Openstack, then Berkeley Data Analytic Stack could work with the support from hadoop and Spark.

Let us look at the following diagram showing how Openstack and XenServer would work together.



***Figure 4.2 Deployment Architecture, Source: Openstack Wikipage*** [***https://wiki.openstack.org/wiki/XenServer/XenAndXenServer***](https://wiki.openstack.org/wiki/XenServer/XenAndXenServer)

Figure 4.2 actually has explained how the Openstack interacts with the existing XenServer as the underlying hypervisor.

When you installed a XenServer on a physical machine, a control domain (i.e. virtual machine) call Domain 0 (denoted and Dom0 in the following) is created as the default dominating domain, which interacts with user input and do the tasks accordingly.

Openstack on XenServer will be installed like a new virtual machine alongside with the Dom0. It will provide the key services for Openstack, except the task for running a virtual machine. It is responsible for hosting a Website interface for the user to interact with, and then forward the user’s request to the Dom0 on the XenServer. Also, some key services of the Openstack like Cinder also needs to interacts with XenServer storage repository service to ensure the working of Openstack.

***4.2 Setup methodology***

XenServer already has the ability to spawn a virtual machine, so what we need to do is to first prepare the XenServer with some extra plugins and setting so that Openstack could easily invoke the service provided by the Xen hypervisor.

Because Openstack does not support using LVM as the underlying storage repository service, you will need to destroy the default storage repository and rebuild one with the ext3 format.

A short script is written to accomplish the job.

|  |
| --- |
| xe pbd-unplug uuid=$PBD\_UUID;  xe pbd-destroy uuid=$PBD\_UUID;  xe sr-forget uuid=$SR\_UUID;  xe sr-create host-uuid=$HOST\_UUID shared=false type=ext content-type=user device-config:device=$DEV\_PATH name-label="RAID1"; |

Then, you will need to install all necessary small application for a Openstack to work properly. More details of the preparation scripts could be found at the github page (link in the Appendix).

After you get a clone from the devstack[2], you will need to run the ***install\_OS\_domU.sh*** script inside the devstack to have a the DomU installed with Openstack Service running on it.

Then you should have installed the Openstack successfully, and thus can launch a new instance (it is actually a virtual machine running on the XenServer) to get a cloud computing service.

***4.3 Challenges and possible solution***

We did come across a lot of problems during our setup of Openstack together with XenServer.

Firstly, when the ***install\_OS\_domU.sh*** is executed, it expects that there are only one physical host that it can reach. However, those prepared XenServer are connected via a Server Pool, so the script will see multiple hosts available and thereafter it will stop to execute, requiring only one host should be accessible by the script. Then you will probably need to detach the server from the server pool and then reinstall XenServer (because XenServer detached from server pool will have the storage damaged) and then try to install Openstack.

The second major problem is with the networking. The Openstack expect the physical server that is running XenServer to have three physical network interface card which our server has only two. Thus the network is not properly configured, and the newly launched instance lack the ability to connect to the outside network. It is possible to tackle the problem with virtual NIC well configured on the XenServer.

The third major problem is with the storage repository. The Openstack has a component specified to serve for the volumn service, which requires a special storage repository as the backend from XenServer. With proper configuration, a newly created partition of the hard disk could serve as the storage repository for the Cinder service of Openstack.

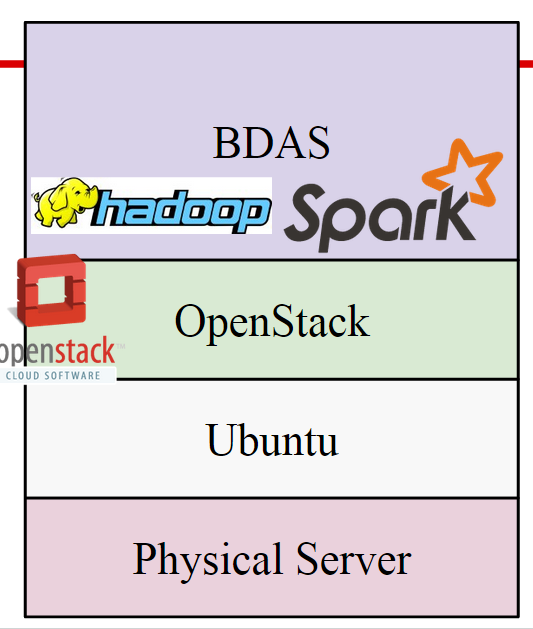
***4.4 Why we abandoned this approach***

When we are trying to configure a new repository for the Cinder service from Openstack, we wrongly messed up the data on the disk, and thus the XenServer we are operating on cannot work properly anymore. It’s predictable that more problems will come up even though we manage to overcome current existing problem. Also, there are really few documentation and materials related to setting up Openstack on XenServer, so supports are very limited. Thus we decide to let go of XenServer and switch to Ubuntu Server, expecting we would not need to do so many works to configure the hypervisor itself for it to work with Openstack, but instead configure Openstack to provide proper service.

1. ***New Design***

**5.1 System Design**

Because of the tough problems we met in setting up XenServer, we decided to install Openstack based on the Ubuntu instead. Based on the openstack, we then build hadoop and spark. But before configure hadoop and spark, we should set up Sahara which provides users with simple means to provision a Hadoop and a Spark cluster.



***Figure 5.1 The Design using Ubuntu Server as the underlying hypervisor***

Observe that the difference between figure 5.1 and figure 4.1, in the new design hadoop and Spark are considered to be part of BDAS.

**5.2 Setup Methodology**

Openstack is a very complex cloud computing platform. It does provide some ‘plug-and-play’ setup method, but in order to have the big data processing functionality, special configurations are needed.

**5.2.1 General Openstack Setting**

Generally Openstack could be configured via devstack[2]. devstack provides a bunch of shell script and utilities to help you build a local cloud computing platform, as long as you write the configuration file correctly.

Part of the configuration file is presented here.

|  |
| --- |
| **#----NETWORK CONFIGURATION----#**  **#The IP address of the current host of openstack**  **HOST\_IP=10.0.21.5**  **FIXED\_RANGE=192.168.1.0/24**  **FIXED\_NETWORK\_SIZE=256**  **FLOATING\_RANGE=10.0.20.56/29**  **#-- possible setting for sahara --#**  **enable\_service s-proxy s-object s-container s-account**  **SWIFT\_REPLICAS=1**  **#-------AUTO ASSIGN FLOATING IP ENABLE, POSSIBLE PROBLEM HERE-------#**  **EXTRA\_OPTS=(auto\_assign\_floating\_ip=True)**  **LOGFILE=/opt/stack/logs/stack.sh.log**  **VERBOSE=True**  **LOG\_COLOR=False**  **SCREEN\_LOGDIR=/opt/stack/logs**  **ENABLED\_SERVICES+=,sahara** |

It is important to properly set the fixed ip range so that it will not collide with the outside network, otherwise a newly launched instance will be lack of connectivity to the world wide web.

**5.2.2 Sahara Setting in Openstack**

After the general Openstack setting, the question of “how to perform Big Data analysis with Openstack?” came to mind. “Sahara” is Openstack’s answer to big data question. It is an integrated project in the Juno release, and actually started out as something called Savanna.

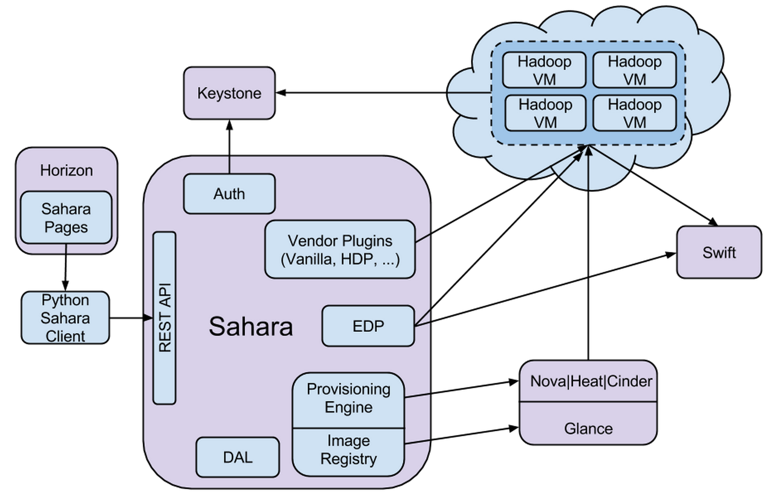
Sahara aims to provide users with simple means to provision a Hadoop cluster at OpenStack by specifying several parameters like Hadoop version, cluster topology, nodes hardware details and a few more.

The key features of Sahara are:

1. Fast Hadoop cluster deployment.
2. Supports different types of jobs: MapReduce, Hive, Pig and Oozie workflows.
3. Data source can be taken from various sources: Swift, HDFS, NoSQL and SQL databases.

**5.2.2.1 OpenStack Configuration**

The architecture of Saharais shown in figure 5.2.



***figure 5.2 The Architecture of Sahara***

There are 6 key components in Sahara. I will introduce components that related to our project.

* Auth -- responsible for client authentication and authorization, communicates with Keystone.
* Vendor Plugins -- mechanisms responsible for configuring and launching Hadoop on provisioned VMs.
* Provisioning Engine and Image Registry -- work with Nova, Heat, Cinder and Glance, which are the key components in OpenStack to provision engine and register images for Sahara.
* EDP(Elastic Data Processing) -- responsible for scheduling and managing Hadoop jobs on clusters provisioned by Sahara. EDP supports Hive, Pig, MapReduce, MapReduce Streaming and Java job types on Hadoop clusters. EDP have access with Swift and HDFS and can work with different types of databases.
* Rest API -- exposes Sahara functionality via REST and supply GUI for the Sahara is located on Horizon.

To setup Sahara upon openstack, there are three basic steps:

1. Put command ‘$echo “enable\_service sahara” ’ into OpenStack configuration file -- “localrc”
2. After enable the Sahara Service, put command ‘$echo “plugins=vanilla,hdp,spark” ’ into configuration file in Sahara -- ‘/etc/sahara/sahara.conf ’
3. Install new special images for data processing with command lines:

|  |
| --- |
| glance image-create --name UbuntuTrustyVanilla2.4.1 \  --disk-format qcow2 --container-format bare --is-public True \  --copy-from http://sahara-files.mirantis.com/sahara-juno-vanilla-2.4.1-ubuntu-14.04.qcow2 |

**5.2.2.2 Hadoop Setup**

After the installation of Openstack and Sahara, we are able to further configure the hadoop and spark cluster.

First, make sure that the hadoop and spark are set to “enable” in the “sahara.conf” file. Then restart sahara, and the “data processing” tag should appear in the console page, like below:



***figure 5.3 The sub-options of Openstack Project***

With the data processing tag appeared, we can configure to launch clusters. Download the hadoop cluster image from the openstack website, make sure the image is the newest version of “vanilla hadoop” image. This image is designed for the hadoop clusters used for cloud computing. After downloading the image, log into the openstack as the admin, and upload the image with the “image -> create image” button. As the image is the qcow2 format, we did a transformation. Although the openstack console page supports QCOW2 format, the NOVA only support the VMDK format image. We converted the image file from QCOW2 to VMDK using the following command:

$ qemu-img convert -f qcow2 -O VMDK xxx.qcow2 xxx.vmdk

After the transformation, we uploaded the .vmdk file to the openstack and created a new image.

However, only by registering the image on openstack is not enough. We still need to register it on the sahara. So using the “register image” button under the “data processing” tag, we registered the image to sahara. Make sure to add tags to the image registered, there are at least two tags, indicating the version of the hadoop and the version of the Ubuntu. We once suffered a lot trying to figure out the reason why our clusters cannot be successfully launched and finally realized we have to tag the image before launching virtual machines.

After configuring the image, we still need to configure the template of launching. We used the “Node Group template” tag to create two group templates, one worker node group and one name node group. The “Availability Zone” “OpenStack Flavor” can be set according to the user’s need, the “Storage Location” however, has to be set as “Ephemeral Drive”. Besides, based on our network configuration, the “Floating IP Pool” need to be configured as “Do not assign floating IPs”. The other configures like “HDFS Parameters” or “ MapReduce Parameters” can be set based on the reality. Another thing that we studied a lot is that in a hadoop cluster, the number of the name node should be exactly one, and the number of the data node should be at least three. If the number of the nodes are configured incorrectly, the cluster cannot be launched.

**5.2.2.3 Spark Setup**

Spark is a fast and general engine for large-scale data processing. It has a lot of advantages as the followings:

(1)Speed: Run programs up to 100x faster than Hadoop.

(2)Ease of Use: write applications quickly in Java, Scala or Python.

(3)Generality: combine SQL, streaming and complex analytics.

(4)Runs Everywhere: Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, HBase, S3, etc.

When compared to Hadoop, Spark is a in-memory version of Hadoop. Hadoop use HDFS while Spark use streaming.

When setting up Spark, we should first download the Spark cluster image from the openstack website. We install new special images with command lines as follows:

|  |
| --- |
| glance image-create --name UbuntuTrustySpark1.0 \  --disk-format qcow2 --container-format bare --is-public True \  --copy-from \  http://sahara-files.mirantis.com/sahara-juno-spark-1.0.0-ubuntu-14.04.qcow2 |

Then the steps are similar with setting up Hadoop. We need to register it both on the Spark and Sahara. Also make sure to add tags to the image registered. After configuring the image, it’s time to configure the template of launching. We used the “Node Group template” tag to create two group templates, one worker node group and one name node group. Then used the “Cluster Template” to create two cluster templates. And then we can create cluster. Later, we use the command ‘ssh -i xxx.key ubuntu@xxx’ to connect to Spark. At first, We cannot ssh to ubuntu successfully. Then we found that we should wait for a long time because creating cluster took much time. It was necessary for us to download the scala language first before running spark shell. So we use the command to download and configure Scala:

|  |
| --- |
| $ wget http://www.scala-lang.org/files/archive/scala-2.10.4tgz  $ tar xvf scala-2.10.4.tgz  $ ln -s scala-2.10.4 scala  $ export SCALA\_HOME=/home/ubuntu/scala  $ export PATH=$SCALA\_HOME/bin:$PATH |

We then use a ‘Hello World’ program to test whether the Scala is there.

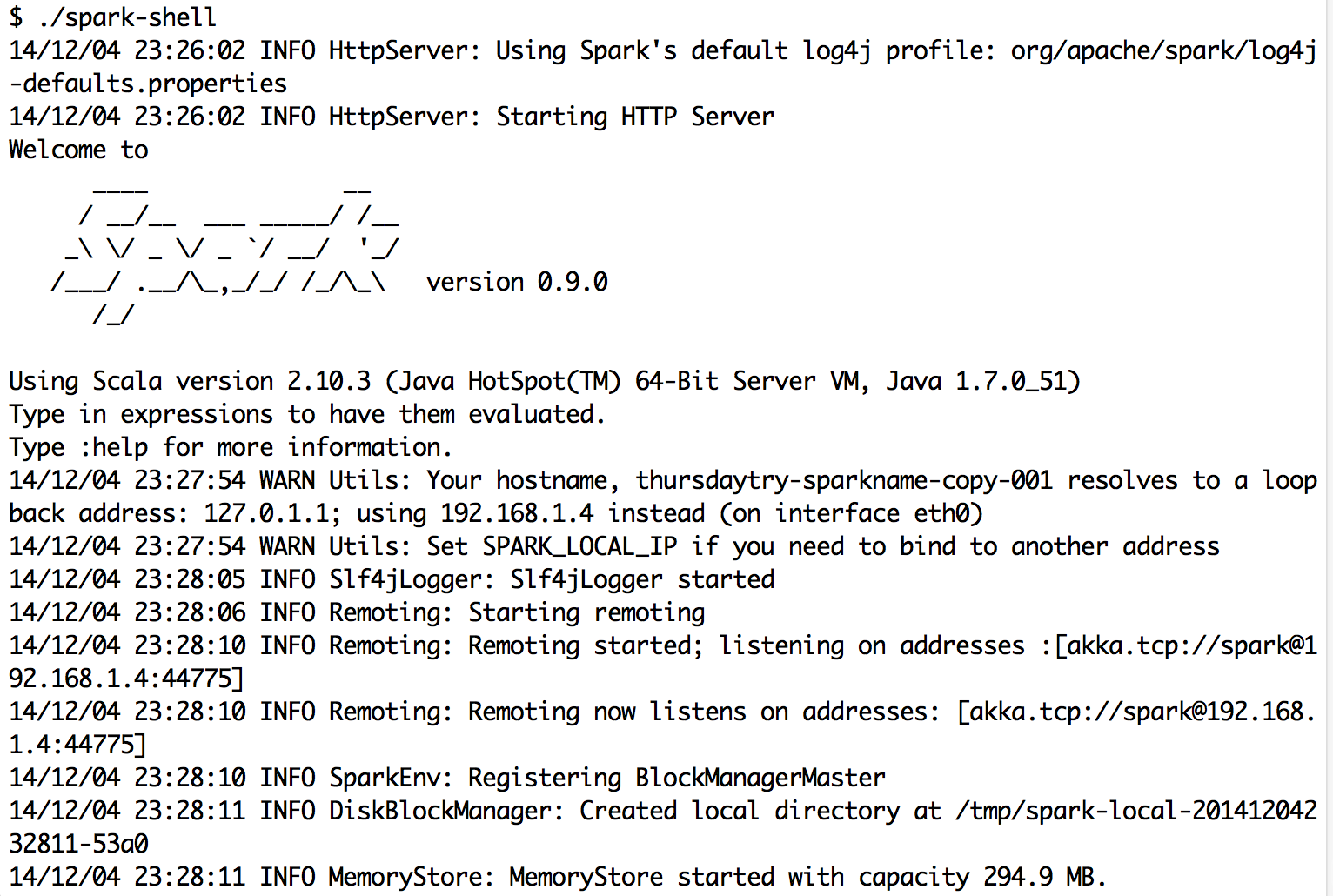
|  |
| --- |
| scala>object HelloWorld{  | def main(args: Array[String]) {  | println(“Hello, world!”)  | }  | }  defined module HelloWorld  scala >HelloWorld.main(null)  Hello, world! |

This result confirms that we install scala successfully.

Then it is time to install and launch Spark.

|  |
| --- |
| $ wget http://d3kbcqa49mib13.cloudfront.net/spark-0.9.0-incubating-bin-cdh4.tgz  $ tar xvf spark-0.9.0-incubating-bin-cdh4.tgz  $ ln -s spark-0.9.0-incubating-bin-cdh4.tgz spark  $ cd bin  $ ./spark-shell |

Figure 5.4 shows launching Spark successfully.



***figure. 5.4 Spark launched successfully***

Apache Spark is a ***cluster computing engine***. It abstracts away the underlying distributed storage and cluster management aspects. The main abstraction for computations in Spark is [***Resilient Distributed Dataset(RDD)***](https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf). Due to its simplified programming interface, it unifies computational styles which were spread out in otherwise traditional Hadoop stack. MapReduce model is composed of following stages: Map-->shuffle and sort -->Reduce. But RDD does not restrict us in traditional MR way. This is a MapReduce example in Spark:

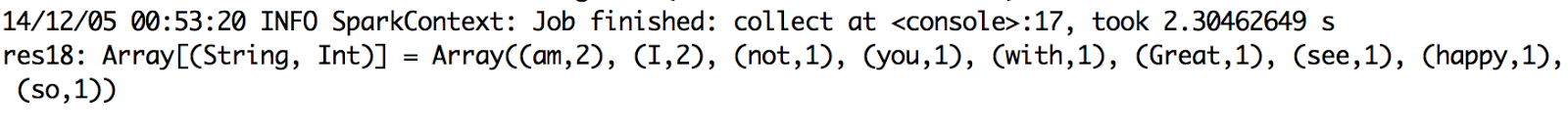
(1) Input a text as followings:

|  |
| --- |
| I am so happy with you  Great I am not see |

(2)Use the following command:

|  |
| --- |
| scala> val textFile = sc.textFile(“home/ubuntu/spark/input.txt”)  scala> val wordCounts = textFile.flatMap(line => line.split(“ ”)).map(word => (word, 1)).reduceByKey((a,b) => a + b)  scala> wordCounts.collect() |

Figure 5.5 shows Result.



***figure. 5.5 Result of the program***

From the result, we can find that running MapReduce in Spark is easier and faster.

**5.3 Challenges and Possible Solutions**

Though devstack is a very convenient way to set up Openstack on our local server. Many details are hidden behind the screen, so some parts are using the default setting does cause us some problems. Also, because Openstack is open source solution, some bugs does exist and we may come into them.

1. We fail to launch cluster after we configure all the node template and cluster template. After emailing to the mailing list and a bunch of googling work, we found that we will need to specifically add plug-in tags to those images which will serve for plug-in like Vanilla (for hadoop) and Spark.

2. When launching hadoop cluster using Sahara in Openstack, there is a security group error. It is a very strange error, with the following error message.

|  |
| --- |
| Resource CREATE failed: BadRequest: Security group 1 not found for project |

However using the following command, we can successfully locate the security group.

|  |
| --- |
| $ nova secgroup-list |

And we work-around this problem by setting the security group name the same as security group id, e.g. , name ‘3’ and id ‘3’. It seems that when trying to verify the accessibility of security group for a tenant, there is a bug in the code that looks into the security group name instead of checking the security id.

3. Master node and worker node does not work collaboratively. And this error haunts around for the last few weeks, and we are unable to locate what exactly is the cause for this problem. We did try to set up Neutron as the networking service for Openstack, expecting the problem comes from the floating ip setting of the newly created instance, but the problem was not resolved. And it remains unsolved till now.

1. ***Project Result, Conclusion and Future work***

In the end of this project, we did set up a prototype of local cloud computing platform, with the basic functionality to launch instance to provide basic computational service, and with the basic data processing functionality. However, those cluster does not work in an expected way, this problem still needs to be fixed up. Much effort are put into dealing with XenServer, and this does slow us down quite a bit in the early state.

In this project, we learn that it is not a trivial job to set up a cloud computing platform. Many things need to be taken into consideration, like networking, disk partition. And we needs to understand the whole picture of how Openstack works, as well as some minor details about how different components interacts. Open source solution is typically difficult to work with, but you can learn a lot of things from it.

In the future, it is possible to configure multiple physical servers together as a whole Openstack system, each of them running one or some components in the Openstack. It is a more efficient way of deploying Openstack cloud computing platform, but it does require extra effort to configure each server with specified functionality. Canonical release a solution recently, named as MaaS [4], this may help to deploy Openstack on multiple servers easily.

Also, Neutron component of Openstack could be further configured, so that the networking could acts more normally with external network. And thus instances within the Openstack subnet could be accessed ‘directly’ from the external network.

1. ***Contribution of each team member***

Zilong Huang, responsible for OpenStack and XenServer’s cooperative, and Openstack Ubuntu

Mohan Yang, responsible for Openstack Ubuntu and Openstack Sahara Hadoop, Spark

Tongtong Bao, responsible for Openstack Ubuntu and Openstack Sahara Hadoop

Rao Li, responsible for Openstack Sahara and Testing Sahara

Yuwei Yang, responsible for Openstack Sahara Spark and Testing Sahara

**Reference:**

**[1] Openstack Administration Guide** [**http://docs.openstack.org/admin-guide-cloud/content/ch\_preface.html**](http://docs.openstack.org/admin-guide-cloud/content/ch_preface.html)

**[2] Devstack github page** [**https://github.com/openstack-dev/devstack**](https://github.com/openstack-dev/devstack)

**[3] XenServer and Openstack, source:** [**https://wiki.openstack.org/wiki/XenServer**](https://wiki.openstack.org/wiki/XenServer)

**[4] Introduction to using Xen and Citrix XenServer with Openstack, source:** [**https://wiki.openstack.org/wiki/XenServer/XenAndXenServer**](https://wiki.openstack.org/wiki/XenServer/XenAndXenServer)

**[5] Metal as a service, source :** [**https://maas.ubuntu.com/**](https://maas.ubuntu.com/)

**Appendix  
Documentation used:**

Openstack Installation Guide for Ubuntu 14.04, <http://docs.openstack.org/juno/install-guide/install/apt/content/>

Openstack Architecture Design Guide,

<http://docs.openstack.org/arch-design/content/>

Openstack Sahara,

<https://wiki.openstack.org/wiki/Sahara>