A study on OpenStack and secure multi-cloud license sharing

Winter Internship 2016-17
Distributed and Object Systems Lab
IIT Madras

Parth Thakkar
Ganesh Kulkarni (Supervisor)
Dr. D. Janakiram (Supervisor)

28th November - 3rd January

Contents

1 Abstract

This report describes the work I did as a part of my winter research internship at Distributed and Object Systems Lab, IIT Madras. My work consisted of two parts: setting up a private cloud using OpenStack on BOSS MOOL, and investigating a secure mechanism for sharing software licenses across multiple private clouds. While setting up a basic deployment of OpenStack is an easy process thanks to the well-written documentation and availability of pre-built packages for popular Linux distributions, deploying OpenStack on BOSS MOOL is a challenging task and requires some hands-on experience with Linux. In this report, I describe the challenges faced during the installation, followed by a description of the final setup of the successful installation. I then proceed to describe the problem of license sharing between private clouds, my proposed solution, and an analysis of the same.

2 Introduction

OpenStack is a popular open source ecosystem for deploying private, hybrid or even public clouds. It consists of various loosely coupled services that interact with each other to provide a massively scalable cloud operating system. A typical OpenStack consists of various machines (nodes) running certain services, and the whole setup involves multiple networks to separate different network traffic types.

BOSS MOOL is an Indian GNU/Linux distribution developed jointly by CDAC and IIT Madras. It aims to redesign the Kernel to reduce coupling, and also adds certain custom security features. It is customized to suit India's digital environment. It supports most of the Indian languages.

Installing OpenStack on BOSS MOOL is challenging because OpenStack does not support it. While BOSS MOOL is Debian based, its packages aren't up-to-date and hence it is not possible to install a recent release of OpenStack using apt. The same problem exists with Ubuntu, however, OpenStack provides Ubuntu Cloud Archive - which is a repository containing the latest releases. Unfortunately, this cannot be added to BOSS MOOL, and only the BOSS MOOL package maintainers can fix this. Thus, the only resort is to install OpenStack from source. While it's usually simple to install software from source, it becomes slightly involved when there are several components in the picture, including numerous daemons that need to be manually setup. The first part of the report describes the deployment process of OpenStack on BOSS MOOL along with the various challenges involved. It concludes with a description of the final setup.

The next part of the internship was to investigate the problem of secure software license sharing between multiple private clouds. The problem has a practical relevance: consider the case of a large organization housing several private clouds for each department. Due to the sensitivity of data, the clouds must have minimal interaction, but they also need to share licensed software amongst them. Solving this would reduce cost as well as some IT work. The second part of this report describes the problem in more detail, proposes a solution, and also analyzes it.

3 OpenStack Architecture

OpenStack is a large ecosystem of various services that interact with each other to provide a massively scalable cloud operating system. It is designed to deploy private, public and hybrid clouds. Like any other cloud infrastructure, OpenStack is deployed on multiple *nodes*, each running certain *services*. These nodes are connected by multiple *networks* for reasons described later. The networks being talked about here are of two types: physical and virtual. It is the virtual networking aspect that is taken care of by OpenStack. The following sections describe the three components of OpenStack: services, nodes, and networks.

To the user of the cloud, the dirty details of cloud infrastructure are invisible, and can easily spawn instances, create networks between them, scale up or down the instances all using simply an API or a UI. But behind the scenes, a lot is going on.

3.1 Services

OpenStack consists of many different services tailored for different uses, however, some of them are key to its functioning. Any OpenStack setup involves at least the following services:

- Keystone - The Identity Service
- Nova - The Compute Service
- Neutron - The Networking Service
- Glance - The Image Service
- Horizon - The OpenStack Dashboard. This is not a hard requirement, but most installations would have this.

Keystone

Central to all the services is Keystone - all other services depend on it for restricting access to only authorized users. To understand the role of Keystone, I find it useful to compare Linux (or any OS) with OpenStack. Thinking along those lines, Keystone is the piece that handles the creation of users and groups, allocating permissions to them, letting users login etc. Keystone integrates with several login mechanisms including but not limited to OpenID, SAML, OAuth etc. It also supports Keystone-to-Keystone federation which is key to the operation of my proposed solution for secure license-sharing.

Nova
Nova is again a crucial service, and its main functionalities are to schedule, spawn, scale and destroy VMs. This service is similar to AWS EC2. It works with various hypervisors including libvirt (qemu and kvm-qemu), Xen and VMWare. Plugins for each of these are available and so switching between them is not a lot of work. Nova provides a native Compute API and also an EC2 compatible API.
Neutron
Neutron handles all networking work in OpenStack. Neutron provides platform independent support for managing every aspect of the Virtual Network Infrastructure (VNI) in the OpenStack deployment: creating/deleting virtual networks, routing, firewalling, load-balancing, VPN, and others. Neutron allows usage of Linux Bridge, OpenVSwitch, ML2 and many other virtual switches for these purposes.
Glance
This service takes care of static disk images which contain the executable code as well as the operating environment. It lets OpenStack users create image templates (flavours) that specify the number of virtual CPUs, memory, hard disks etc. This greatly simplifies the process of creation of new VMs. This service is often coupled with Cinder where the disk images are typically stored.
Horizon
Horizon provides an easy to use, and intuitive web-based UI for managing the cloud, as well as for the users of the cloud. Users can manage VMs, virtual networks, and many other services through the UI. This is useful for various simple tasks which might be overkill for scripting.
How the services interact
OpenStack services mainly consist of one or more daemons, one or more REST API, a MySQL database, and an AMPQ message queue (RabbitMQ currently). All the services interact with each other via the REST APIs and the message queue. The state gets stored in MySQL database. Most services also have a command line client that provides a CLI for the REST API.

3.2 Nodes

In any cloud infrastructure, there are several physical machines dedicated to running virtual machines of the users. These are known as compute nodes in OpenStack terminology. The OpenStack service known as Nova handles spawning of these virtual machines. This is taken care of with the help of hypervisors, and Nova supports most popular ones - libvirt, Xen, VMWare etc. Besides these nodes, there are several other nodes running the core OpenStack services required for its functioning, and other auxiliary services such as block storage, object storage etc.

3.3 Networks

As mentioned in the previous section, an OpenStack deployment consists of several nodes running different services which need to intercommunicate in order to function. This intercommunication is provided by the Physical Network Infrastructure (PNI). In a typical OpenStack deployment there are multiple physical networks connecting these nodes in order to separate the traffic flowing through them. For example, there can be a separate network between the nodes for management traffic. This network, for example, can be behind a NAT and hence public internet cannot access it. Similarly, there can be a provider network that allows public internet to access the VMs. Different networks have different security policies and different content flowing through them. This isolation improves performance, maintenance, and security of these networks.

However, that part is not provided by OpenStack. What OpenStack provides is tools to manage the VNI - the Virtual Network Infrastructure. This means it enables the user to set up networks between the VMs which may be in different physical subnets, set up firewalls, add load balancers, floating IPs, security groups etc. Neutron provides APIs as well as a CLI for performing these operations.

Neutron provides the above facilities in a vendor-neutral manner (to a great extent). It achieves this through a plugin architecture and it supports several popular virtual networking tools including OpenVSwitch, Linux Bridge, VMWare etc. Thus, Neutron does not actually set up the VNI. It leaves that jobs to the above-mentioned tools. What it provides is a uniform API for handling the tools, and also integrating the VNI information with other OpenStack services.

Although Neutron by itself does not provide much for handling the PNI, a knowledge of the same is important for deploying an OpenStack cloud. The simple reason being that the VNI needs to be built on top of PNI.

4 Setup

My entire OpenStack setup was done on a single machine, but to simulate multiple nodes and multiple physical networks, I used two VirtualBox VMs on the host: one for the Controller node and another for the Compute node. Both the nodes had three network interfaces:

1. One interface for a bridged network that connects both VMs, the host and the physical network in the DOS Lab.
2. The other two - a host-network interface and a NAT-interface combined provided a *NAT-network* between the VMs and the host (which had to be done because VirtualBox's built-in NAT-Network wasn't functioning correctly).

The other aspect is the networking type of the setup. The self-service networking option was deployed as compared to the provider networking option. Self-service networking provides more customizability by providing overlay segmentation methods such as VXLAN. Essentially, it routes virtual networks to physical networks using NAT. Additionally, this option provides the foundation for advanced services such as LBaaS and FWaaS.

The Mitaka release was to be used for the setup. Unfortunately, BOSS MOOL package repository had very old releases of OpenStack packages, and hence I had to install from source. Normally to make a basic OpenStack deployment on supported systems, it's sufficient to follow the documentation on the website. However since in this case, the installation had to be done on BOSS MOOL, documentation was not clear. Besides, installing OpenStack from source is as such challenging due to the vast number of separate components that need to be set up. For example, for setting up Nova on the compute node, at least 4 different daemons need to be configured. Besides this, the binaries need to be installed and must be properly configured. Similarly, there are 5 different neutron daemon unit-files to be written. Same holds for other services to be installed. This is a tedious, limited and highly error-prone process. The lack of documentation regarding installation from source aggravates the issue.

After initially struggling with the installation, I came up with a middle-ground solution. I installed older versions of OpenStack packages from BOSS MOOL package repository, and let that process take care of setting up the daemons and basic configuration. Since those scripts don't change greatly between releases (at least for a basic setup), this worked out pretty well. Once the daemons and configuration templates were setup, I then built the OpenStack packages from source resulting in latest binaries. These binaries then replaced the older ones and thus a working OpenStack setup of the Mitaka release was ready. After figuring out the installation mechanism for the latest packages,

Figure 4.1: The structure of the setup

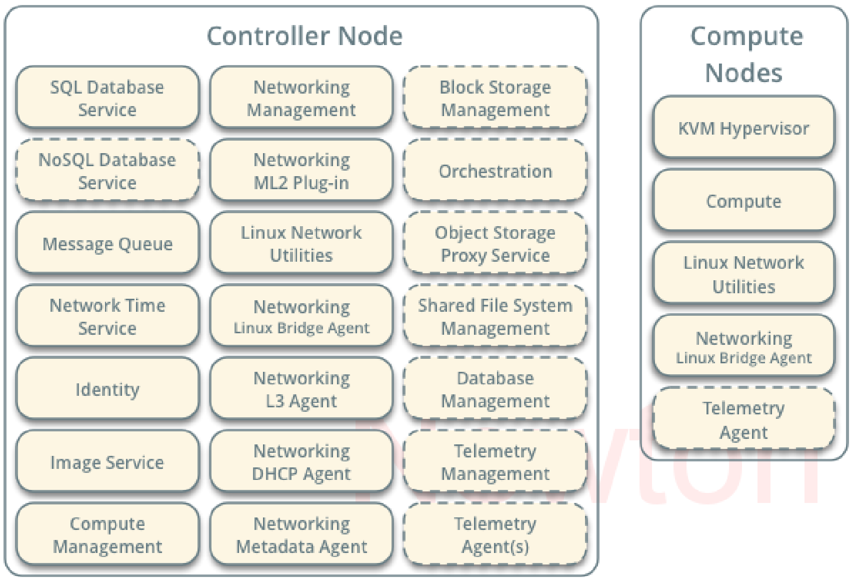


Figure 4.2: The networking of the setup (Only Controller and Compute nodes were included in the setup)

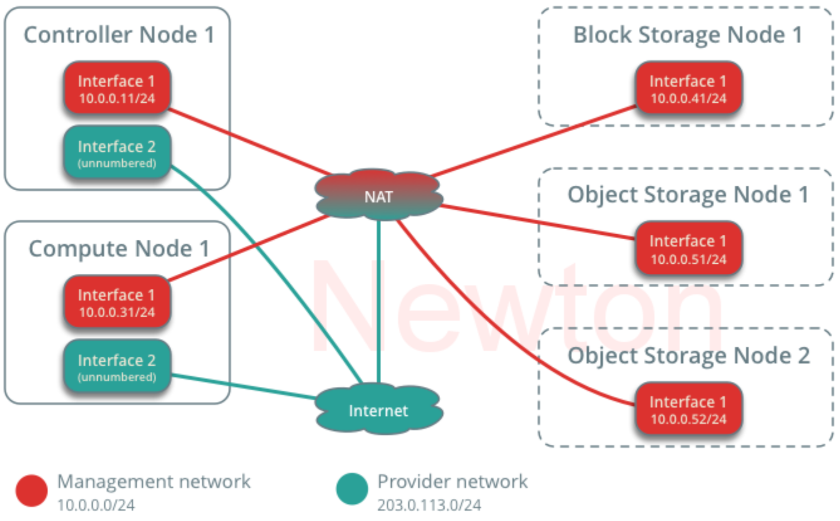
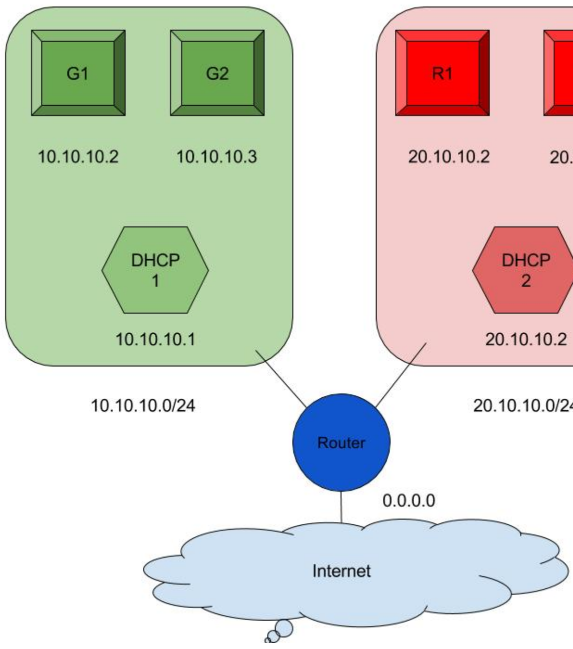


Figure 4.3: Structure of the test setup on the cloud



having the right configuration was simply a matter of following the OpenStack documentation. Of course, this wasn't pain-free as it required me to modify several unit-files (for services) and configuration files, but this was the best approach I could find.

As shown in Figure 4.1, Controller node houses several services. One of the important services is the Networking L3 Agent which provides the self-service option in the cloud. It allows the creation of overlay networks between the VMs running on hosts in different subnets. This is important when the cloud scales to several compute nodes. But not just that, even for a small cloud it allows different tenants to create their own private networks.

For compute node, I used the Qemu hypervisor. VirtualBox does not support hardware acceleration *within* its VMs, and hence KVM-Qemu was not an option. Deploying KVM-Qemu on physical machines is not very different from the setup being described.

Debugging required an understanding of Linux Network Namespaces. For testing the cloud setup, I created two virtual networks on it, having 2 instances and a DHCP server in each, connected them with a virtual router and gave them internet access to them through the router. Doing this required a lot of debugging. However, an understanding of this simple fact makes a lot of things clear: OpenStack creates a network namespace for each network, for the router, and uses virtual bridges and virtual ethernet pairs to interconnect these namespaces. The namespaces and the virtual bridges can be inspected easily on the command line, and things become clear very easily using that information.

5 License Sharing

The following sections describe the second part of my internship. The problem is to devise a secure mechanism of sharing software-license between multiple private OpenStack clouds. Consider the scenario where 2 private clouds A and B need to share a licensed software. The software is available within cloud A, and cloud B needs to access it. However, this needs to be done in a secure manner as both clouds house sensitive data that should not leak.

The following section describes how licensing works in software used by organizations. After which the assumptions made by the solution are described, followed by the proposed solution itself. Finally, an analysis of the solution is presented to conclude the work.

5.1 Licensing - Status quo

Software licenses can be broadly classified into two types:

1 Node locked: A software can run on a fixed set of machines. No other machine can run the software.

2 Floating: At a time the software can run on a fixed *number* of machines, but any machine can run the software.

Organizations typically use floating licenses and hence this work is focused on licenses of these types. Floating license setups involve one or more License manager server(s). Whenever the software under consideration is run, it contacts the license manager server and will function only if allowed to. The software may have an open connection with the server for as long as it runs, or it may periodically ping it. There is no restriction assumed on the connectivity between the software and the license server.

A lot of software often come with a third party License Manager software. Popular ones are the FluxLM, IBM LUM, Reprise License Manager etc. Any license manager of this kind comes with a set of (configurable) ports that it uses. The clients also communicate with the license manager using the hostname instead of IP address. All this is taken care of in the solution being proposed.

5.2 Assumptions

The proposed solution makes the assumption that not both parties are cooperative in the act of stealing data - that is, if one party wants to steal data, the other party doesn't want to be helpful. This is a reasonable assumption since if this is the case then there are various mechanisms in which data can be given to unauthorized personnel. This solution protects against *undesired* data access in one cloud from the other cloud.

5.3 Proposed Solution

Basic idea:

Use Keystone-Keystone Federation to limit access to the (docker) images of the software to be shared. Cloud A limits these permissions, and cloud B users can access the images if the permissions allow him. An on-demand VPN is set up between the clouds to allow the software in Cloud B to be able to talk to the license server in Cloud A, and nothing else. The VPN allows only those ports that the license manager needs. Once the usage is over, the VPN is destroyed.

The details:

1 ***What is Keystone-Keystone federation and how does it work?***

Keystone-Keystone Federation is an authentication mechanism that allows Keystone from one OpenStack cloud to act as an Identity Provider (IdP) for Keystone in another cloud. At a very high level what happens is when a user of Cloud B wants to login to Cloud A, he first logs in to Cloud B. Cloud B's Keystone then sends the authentication information to Cloud A, and thus cloud A now knows the identity of the user. The authentication information being sent along with Cloud B's digital signature, so Cloud A can verify that it was indeed Cloud B that sent the information. Since Cloud A has been configured to trust Cloud B as an IdP, the problem of authentication is solved. Cloud A then maps the provided identity to a user of its cloud and returns a token to the requester. This token is used for further interaction with Cloud A.

2. ***How are the images of the software downloaded?***

The software images can be stored in Cloud A's block storage service, or elsewhere. Since the user of Cloud B has now been authenticated by Cloud A, it can be verified if he has the permissions to download the image he is requesting. If so, the image is then transferred over the network between the clouds.

3. ***How is the VPN set up?***

Once the docker image has been installed on the target instance in Cloud B, a VPN is established between the license manager in Cloud A and the docker instance. This can be done using OpenStack's VPNaaS feature. Since the ports required by the license manager are known, the VPN can be set up to allow only those ports to communicate, so that the attack vector is reduced. Once the usage is over, the VPN is destroyed.

All of this can be streamlined into a single service that handles everything - maintenance of the image catalogue, authentication, permission handling, setting a VPN and finally destroying the same. The service can provide an API and a Horizon UI to simplify all steps involved.

The following section analyzes the above proposal in depth.

5.4 Analysis

In the whole scenario, 4 players are involved here, each of which can harm in different ways:

1. Software that Cloud A has
2. Cloud A - distributor of the software to be shared
3. Cloud B - consumer of the software to be shared
4. Network eavesdropper

For each of the above, the possible attack mechanisms and the corresponding fixes have been listed below.

1. **Software that cloud A has:**
   - ***Possible attack mechanism***
     1. Access the cloud's private network and possible leak info through the internet.
   - ***Fixes***
     1. Assume we have the digital signature of the software for verification
     2. The software will run inside a container and will be isolated from the network.
     3. If Cloud A uses this, it must have followed various security protocols and undergone server tests. Otherwise, even Cloud A would be at risk. Hence, this isn't an issue of this project.
2. **Cloud A - distributor of the software to be shared**
   - ***Possible attack mechanism***
     1. Upload modified malicious software for Cloud B
     2. Access the other cloud's network
   - ***Fixes***
     1. Cloud B can use the digital signature of the software for verification
     2. Cloud B can use a firewall to allow access only to the container that runs the software.
3. **Cloud B - consumer of the software to be shared**

- ***Possible attack mechanism***
  1. Access other cloud's network
- ***Fixes***
  1. Cloud A can use firewall to allow access only to the License Server

4. **Network eavesdropper**
   - ***Possible attack mechanism***
     1. Read/Modify the data on the network (VPN)
   - ***Fixes***
     1. Use IPSec ESP tunnel to authenticate and encrypt the data flowing between the VPN.

5.5 Advantages and Limitations

The proposed solution has several advantages:

1. **User-friendly** - the separate OpenStack service proposed can handle all the tasks required for securely sharing the license between the clouds. The users of this service do not need to take complicated actions.
2. **Simplicity** - The solution does not involve complicated protocols. It's simple to understand and hence a smaller attack vector.
3. **Built on reliable tools** - Uses existing security infrastructure (IPSec, Federated authentication etc) which has been field-tested for years.

There is one small limitation, however. The solution does not handle the case where both parties are willing to exchange private information. But there are various ways in which that can be done even if the clouds are completely isolated. Hence this is not a big concern for this particular project.

6 Conclusion

Both the tasks of my winter internship have been accomplished. The OpenStack cloud that was installed was very close in complexity and features to a legitimate private cloud of small size. It supports self-service networks and is horizontally scalable in compute power. It had multiple networks between the nodes to separate the traffic and improve security. The proposal to the license sharing problem was successfully devised and analyzed. There hasn't been any work on the implementation of the same. However, a good outline has been laid out to pursue that goal.

7 References

1. http://docs.openstack.org/
2. https://github.com/openstack
3. http://www.innervoice.in/blogs/2013/12/02/linux-bridge-virtual-networking/
4. https://developer.rackspace.com/blog/neutron-networking-l3-agent/
5. http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/
6. https://knowledge.autodesk.com/customer-service/network-license-administration