***CloudLab Documentation:***

***Creating, Managing and Operating OpenStack Instances***

Developed By: Hisham Kanaan, Khalid Amen, Brendan Makar, Melissa Nichols, Nathan Torrez, Tianhuan Tu

Advised By: Dr. Anyi Liu

CSE681: Information Security – Dr. Huirong Fu

School of Engineering and Computer Science, Oakland University

Table of Contents

[Table of Figures 4](#_Toc488151483)

[Introduction 6](#_Toc488151484)

[Problem Statement 7](#_Toc488151485)

[Proposition 7](#_Toc488151486)

[Individual Contributions 8](#_Toc488151487)

[Section I: Creating the OpenStack Instance on CloudLab 9](#_Toc488151488)

[Section II: Logging in to OpenStack and Profile Instructions overview. 13](#_Toc488151489)

[Section III: Creating a Network 15](#_Toc488151490)

[Section IV: Creating a Router 17](#_Toc488151491)

[Section V: Creating a new VM 20](#_Toc488151492)

[Section VI: Managing the created VM 23](#_Toc488151493)

[Option A: Managing through the default console of VM 23](#_Toc488151494)

[Option B: Managing through SSH over VM Public IP 23](#_Toc488151495)

[Section VII: Managing OpenStack Instances Using Python 26](#_Toc488151496)

[A. General Description 26](#_Toc488151497)

[B. Technical Challenges 28](#_Toc488151498)

[1) Reachability from Oakland University infrastructure 28](#_Toc488151499)

[2) Lack/inconsistency of documentation 28](#_Toc488151500)

[3) Inability to manipulate routing tables 29](#_Toc488151501)

[4) Router Interfaces 29](#_Toc488151502)

[C. Profiling the OpenStack Instance (ProfileScript.py) 30](#_Toc488151503)

[D. Re-Construct OpenStack Instance from XML (ReloadScript.py) 40](#_Toc488151504)

[E. Managing a VM instance from the internet (VMSSH.py): 43](#_Toc488151505)

[F. Library Program (MasterScript.py) 44](#_Toc488151506)

[G. Testing Program (TesterScript.py) 44](#_Toc488151507)

F[uture Works 56](#_Toc488151509)

[Conclusions 57](#_Toc488151510)

[Acknowledgements 57](#_Toc488151511)

[References 58](#_Toc488151512)

[Amendment from the REU 2017 Team 59](#_Toc488151513)

[Added Functionality as of July 17, 2017 60](#_Toc488151514)

[Using Virtual Laboratory to Run Attack Scenario Experiment 6](#_Toc488151515)1

[Creation of Laboratory Environment (For Instructors) 6](#_Toc488151516)2

[Running the Virtual Laboratory Environment Using Web Client (For Students) 7](#_Toc488151517)1

[Appendix A: ProfileScript.py 7](#_Toc488151518)4

[Appendix B: ReloadScript.py 7](#_Toc488151519)8

[Appendix C: MasterScript.py 8](#_Toc488151520)2

[Appendix D: TesterScript.py 8](#_Toc488151521)8

[Appendix E: VMSSH.py 9](#_Toc488151522)1

[Appendix F: MasterScript with Added Functionality 9](#_Toc488151522)2

[Appendix G: TesterScript with Added Functionality 11](#_Toc488151523)4

[Appendix H: Documentation for Instructors and Students 12](#_Toc488151523)0

# Table of Figures

Figure 1: CloudLab initial OpenStack profile creation 9

Figure 2: OpenStack parameters, highlighted are ones of particular interest 9

Figure 3: OpenStack parameters showing how to change number of IP addresses and VLANs 10

Figure 4: Cloudlab choosing the cluster desired for implementation 11

Figure 5: Cloudlab experiment homepage 12

Figure 6: OpenStack instance details in CloudLab 13

Figure 7: OpenStack Login page 14

Figure 8: Creating a new network in OpenStack 15

Figure 9: Input network parameters of name and nature in network creation 16

Figure 10: Input subnet details relevant to the created network 16

Figure 11: OpenStack creating a new router instance 17

Figure 12: OpenStack modifying new router parameters 18

Figure 13:OpenStack connecting routers to new networks 18

Figure 14: Choosing which new network router should connect to in OpenStack 19

Figure 15: OpenStack launching a new VM instance 20

Figure 16: Modifying image source for new VM 20

Figure 17: Modifying flavor type for new VM 21

Figure 18: Attaching new VM to desired network 22

Figure 19: OpenStack creating image snapshots 22

Figure 20: Viewing a VM console in OpenStack 23

Figure 21: Associating a Floating IP for a VM using OpenStack GUI 24

Figure 22: Creating a new Floating IP using OpenStack GUI 25

Figure 23: Assign the Floating IP to VM in OpenStack GUI 25

Figure 24: VM status if Floating IP association is successful. 25

Figure 25: Network Topology view using OpenStack GUI 26

Figure 26: General Network Diagram of our access from AWS to the CTL 27

Figure 27: Code snippet showing the profiling of interfaces 30

Figure 28: Code snippet of our prettify function and the XML file saving 31

Figure 29: Sample from the XML File produced by the Script 39

Figure 30: Code Snippet showing how Routers are parsed from the XML to create their dictionary 40

Figure 31: Code Snippet showing how networks and subnets are created on the OpenStack instance using attributes from their dictionary 41

Figure 32: Code Snippet showing the function to upload a new image 44

Figure 33: Code Snippet showing how hosts file is changed and how to create and authenticate an OpenStack connection 45

Figure 34: Location of the RC file to download from OpenStack GUI 46

Figure 35: Sample of how to run the Tester Script, and the list of provided functionalities. 47

Figure 36: Updated List of Options. 60

Figure 37: Showing Number of Public IP Addresses. 62

Figure 38: Where to find the “Images” page 63

Figure 39: Image Settings for Attack Image. 63

Figure 40: Decentralized networks created using Option 20 64

Figure 41: Creating a Security Group. 64

Figure 42: Adding a Rule to a Security Group. 65

Figure 43: adding a Security Group to a Virtual Machine. 65

Figure 44: Selecting option 7 and the output of the file that stores the IP addresses. 66

Figure 45: Selecting option 21 and The IP addresses stored in the database. 66

Figure 46: The Database Schema 68

Figure 47: Configuring the proxy server with the correct rules . 73

Figure 48: IP address to access the web client for one experiment 71

Figure 49: Attack and Victim IP's That the Student Will See. 71

Figure 50: Screenshot showing In Browser SSH Client. 73

# Introduction

In today’s world, there is a vital need for the automated execution and benchmarking of experiments. The ability to consistently reproduce experiment environments and their results is crucial. Research in the field of engineering and computer science is no different to this premise. Empirical research based on detailed benchmarking results has been essential in driving value to the research question in terms of proving or negating the proposal.

One particular topic of research has been the impact of the virtualization technology on changing the way everyone conducts business. It has been clear that the current market trend is headed towards a virtualized environment across the different layers of the technology. Since its original inception with the virtualization of end host instances and specialized servers, virtualization technology has come a long way in its maturity to be adopted in the networking field. The concept of an overlay network has been developed, with virtualized routers, switches, firewalls as well as a host of other cross layer virtualized security appliances. This has helped organizations cut down on the operation costs, increase their robustness and flexibility, decrease the mean time to recover from failures and build scalable topologies on the fly.

With the above being said, many initiatives were proposed as a collaboration between different organizations to introduce the virtualization technologies to the public. This would help drive forward the exploitation of the advantages of virtualization at a minimal cost to the users. In this document, we will discuss our efforts in building, and maintaining infrastructure instances built on OpenStack and Cloudlab. Cloudlab is the joint initiative between three universities: Wisconsin, Utah, and Clemson, and it provides the users a virtualized “slice” of their infrastructure for users to be able to experiment on. For more information on OpenStack and Cloudlab, please visit references [1], [2], [3] and [4].

# Problem Statement

In various fields of interest, such as research and education, dynamic and accurate recreation of research results is essential. For example, in the field of education, conducting studies on a topic can be more involved within participants in a “Create-once-produce-many” implementation. In this case, the primary collaborators in the topic are able to produce a single instance on which they build their material. However, they need to be able to pass copies of this material to students for local deployment and a direct hands-on experience. In the field of research, being able to swiftly backup and restore your topologies is vital. This can help in a more detailed benchmarking as well as the ability to provide manageable copies of the researcher’s environment. This can help others recreate, validate and further advance the topics of study based on the previous efforts.

# Proposition

In this effort, we build on the virtualization capabilities that are provided by Cloudlab and generally through OpenStack. We aim at creating a framework through which users can manage their virtualized environments that are deployed on OpenStack instances in an automated and script-able manner. We also propose the automated profiling and restoration of entire instances with minimal user overhead.

This document will proceed with sections on creating OpenStack instances deployed on Cloudlab resources, and the general time-consuming manner through which networks are created. Then, we offer our own deployed scripts that provide the users with capabilities of automating various functions as well as easily creating profiles of their environments, and later we show the ease with which a profile can be recreated. We also include information on how to create a specific attack scenario experiment, and how to use the web client created for use with this experiment. The illustrating screenshots and recommendations included in this document are based on primary experience in the operation of OpenStack instances. Please consider entire framework we proposed as a continued work in progress as we have provided now the necessary and sufficient conditions to profile and reproduce environments and results in OpenStack and Cloudlab.

# Individual Contributions

In this project, Hisham has tackled working with OpenStack and CloudLab. He studied the OpenStack software definitions and the SDK functionalities provided by OpenStack. Hisham was also responsible for finding solutions around the problems faced, as well as creating and validating the Python scripts that are produced herein.

Khalid tried out an introspection library, LibVMI that would help security researchers monitor their compute nodes and perform low level introspection to discover any compromise.

Brendan was operating with both focuses, and oversaw the website matters. In addition, Brendan was a co-author in this document, and a reviewer for its contents.

Melissa, a part of the second group who worked on this project, streamlined and added functionality to the previously created scripts, created images for use with attack scenario experiments, and wrote experiment documentation including the additions contributed by the second group to this document.

Nathan, also a part of the second group who worked on this project, streamlined and added functionality to the previously created scripts, added Snort rules to detect exploits with our attack scenario experiment, and helped to optimize the web client.

Tienhuan, also a part of the second group who worked on this project, developed a web client for usage with attack scenario experiments.

# Section I: Creating the OpenStack Instance on CloudLab

To start with, after logging in to Cloudlab, you are prompted to choose a profile for instantiation. For our purposes, we will choose the default OpenStack profile.

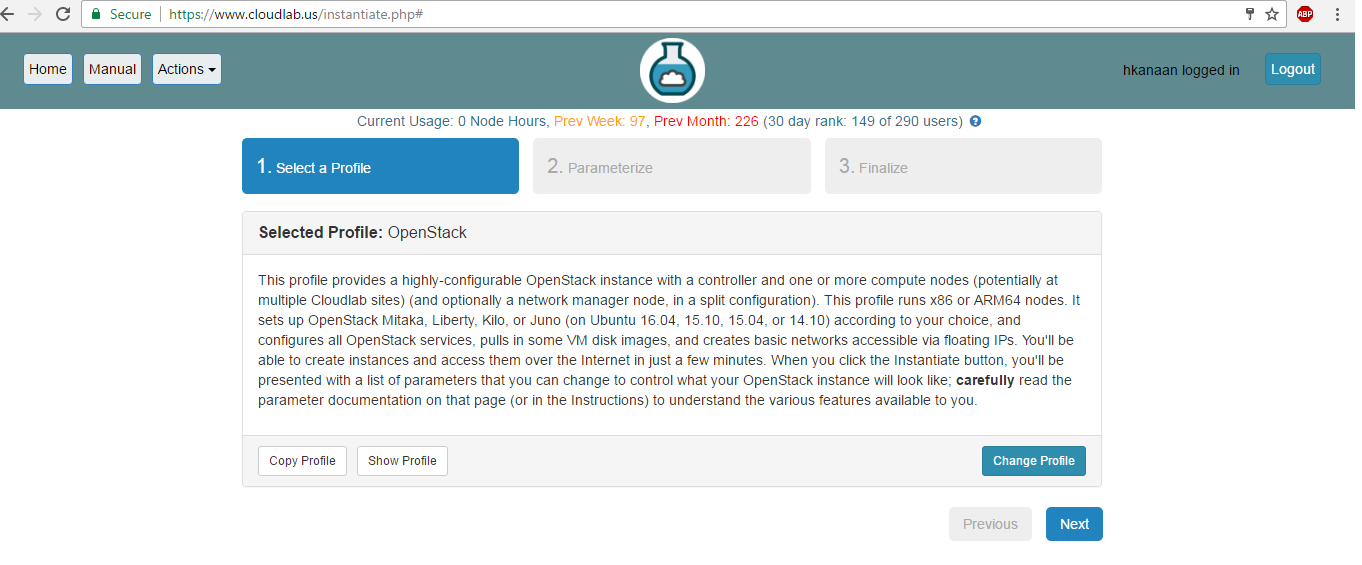


Figure 1: CloudLab initial OpenStack profile creation

After choosing next, you will be prompted to configure the parameters of the OpenStack instance. Of the more important parameters, **we advise you to select more than 4 Public IP addresses to be allocated for your instance** (as shown in the figures below).

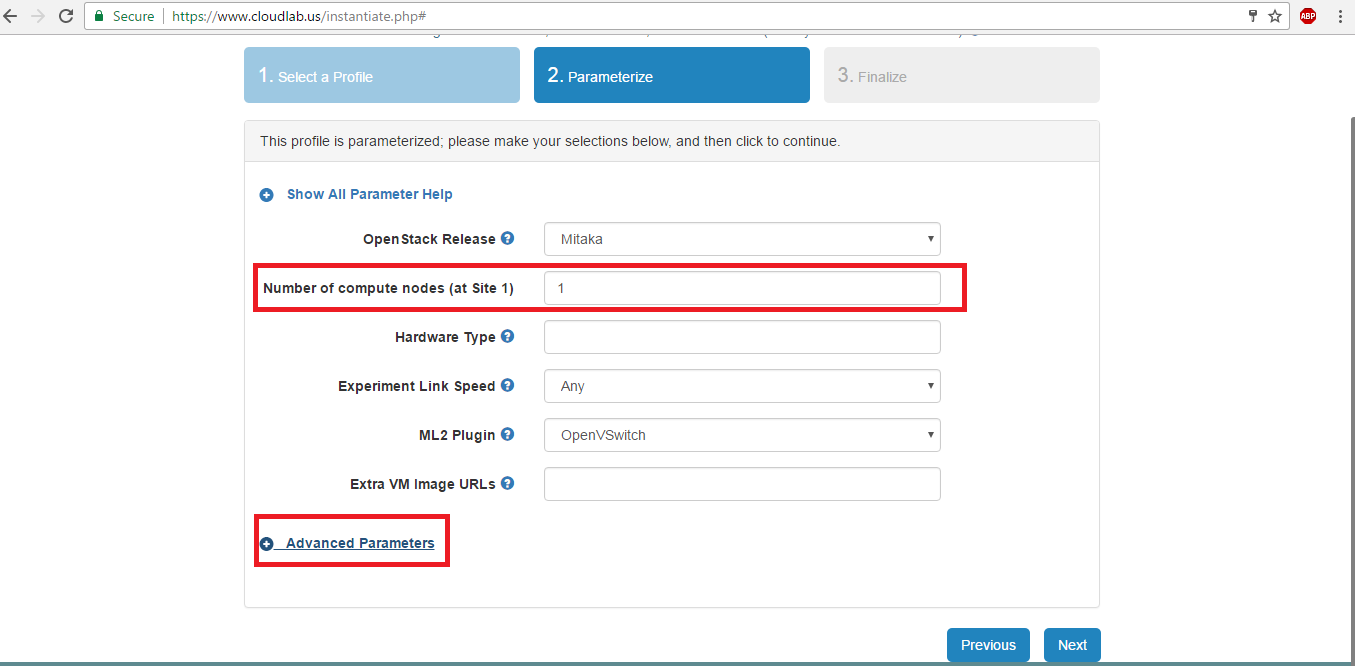


Figure 2: OpenStack parameters, highlighted are ones of particular interest

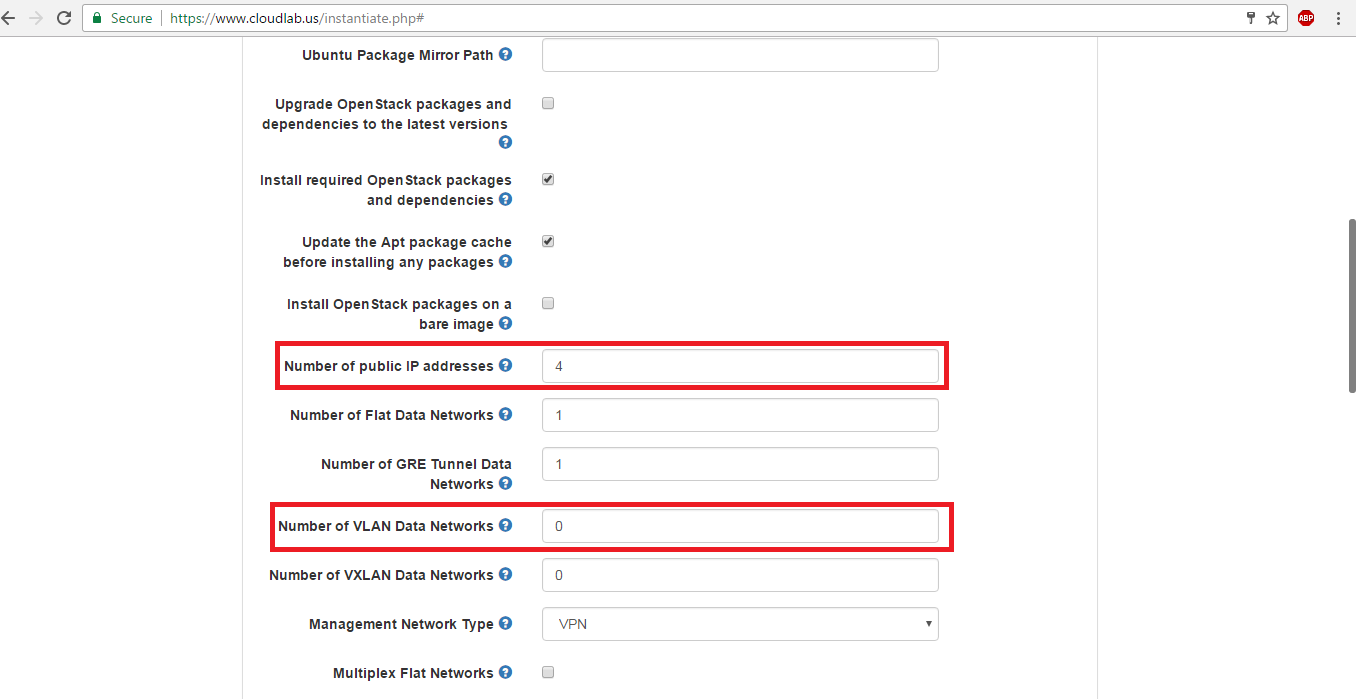


Figure 3: OpenStack parameters showing how to change number of IP addresses and VLANs

This is because by default, 2 of the 4 public IP addresses are allocated for the network routers of ext-net and flat-net. As a best practice by iteration, we are selecting **8 Public IP** addresses. We will stress the importance of this step in later sections. In case there is a plane to build logically differentiated networks, then the number of VLANs can be increased as displayed above. If you are running the attack scenario experiment, you must select **16 Public IP** addresses. This will be explained in more depth later in this document.

In case of preference of a certain cluster over the other due to hardware/compute requirements, then in the next page, we are able to choose the preferred one. In our case, we prefer to deploy over Wisconsin.

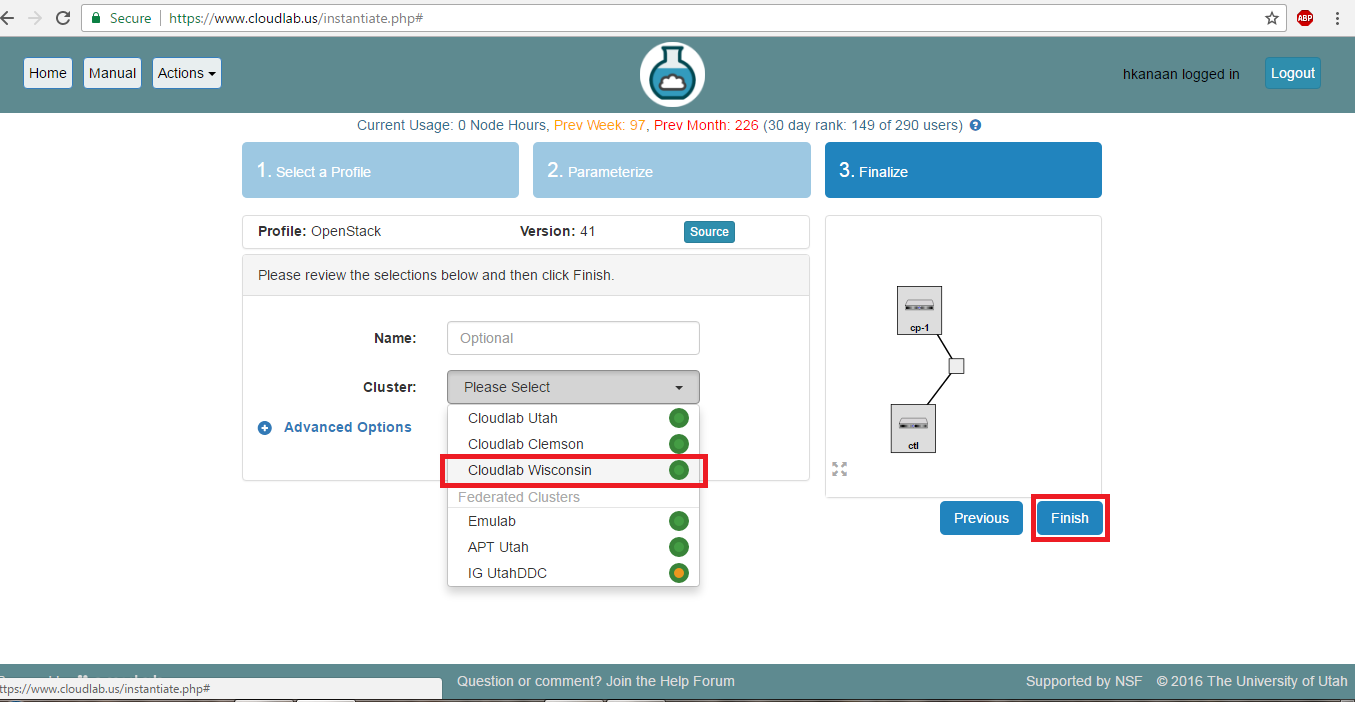


Figure 4: Cloudlab choosing the cluster desired for implementation

One we click on finish, Cloudlab will need approximately 15 minutes to completely boot up the OpenStack instance we just configured. The status “**booted**” means that not all services are yet activated. When the status changes to running, we may enter our instance to configure as we will see in the below sections. You can extend the experiment by clicking the “Extend” button.

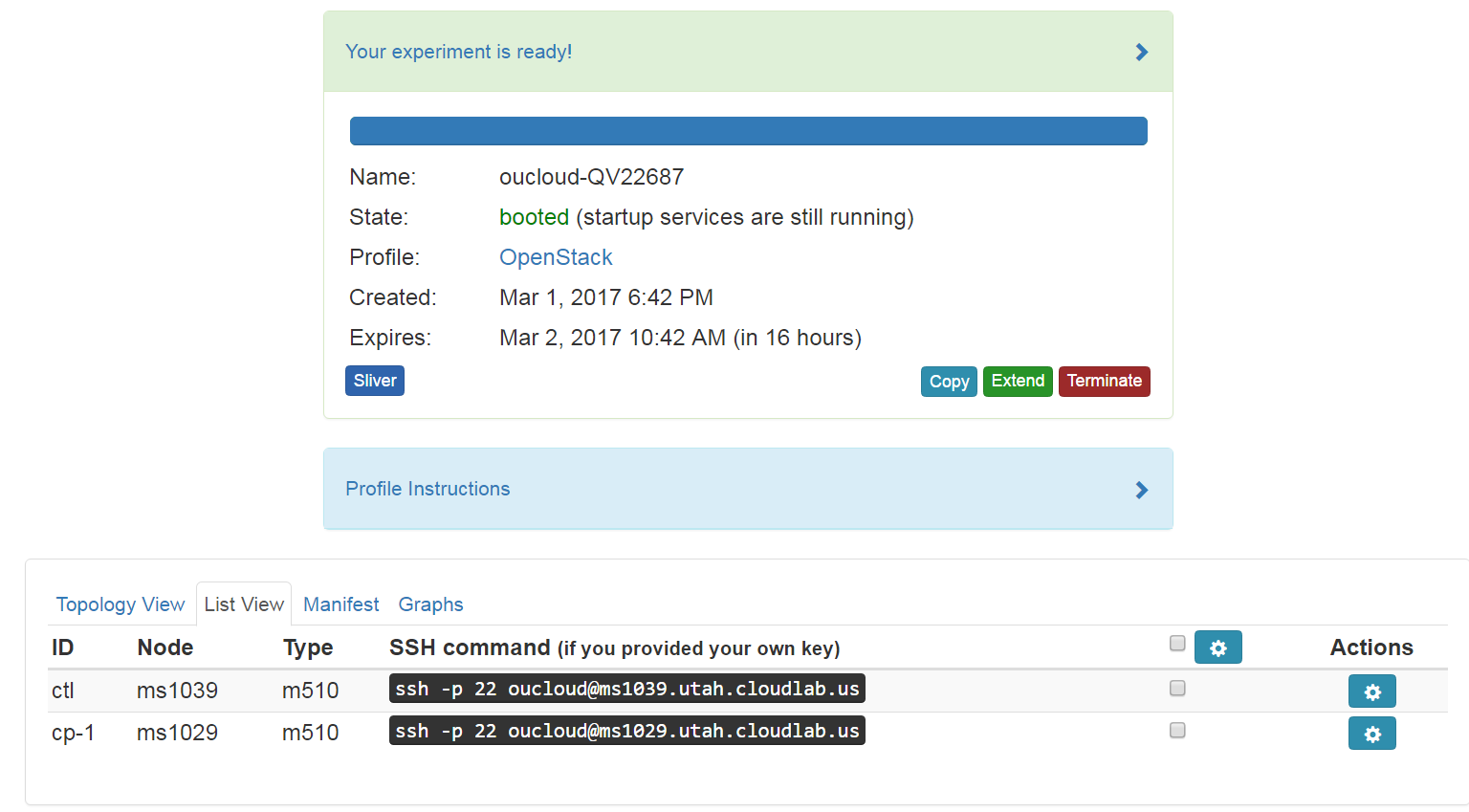


Figure 5: Cloudlab experiment homepage

# Section II: Logging in to OpenStack and Profile Instructions overview.

Once the OpenStack instance is created, the page will have a clickable “Profile Instructions”. We expand this view to have a look over the following as highlighted in the below figure:

1. The default username of the OpenStack Dashboard: **admin**
2. The domain of the OpenStack Dashboard Login: **default**
3. The default username of any create VM: **ubuntu**
4. The password for the OpenStack Dashboard as well as all VMs: the randomized password highlighted as below.

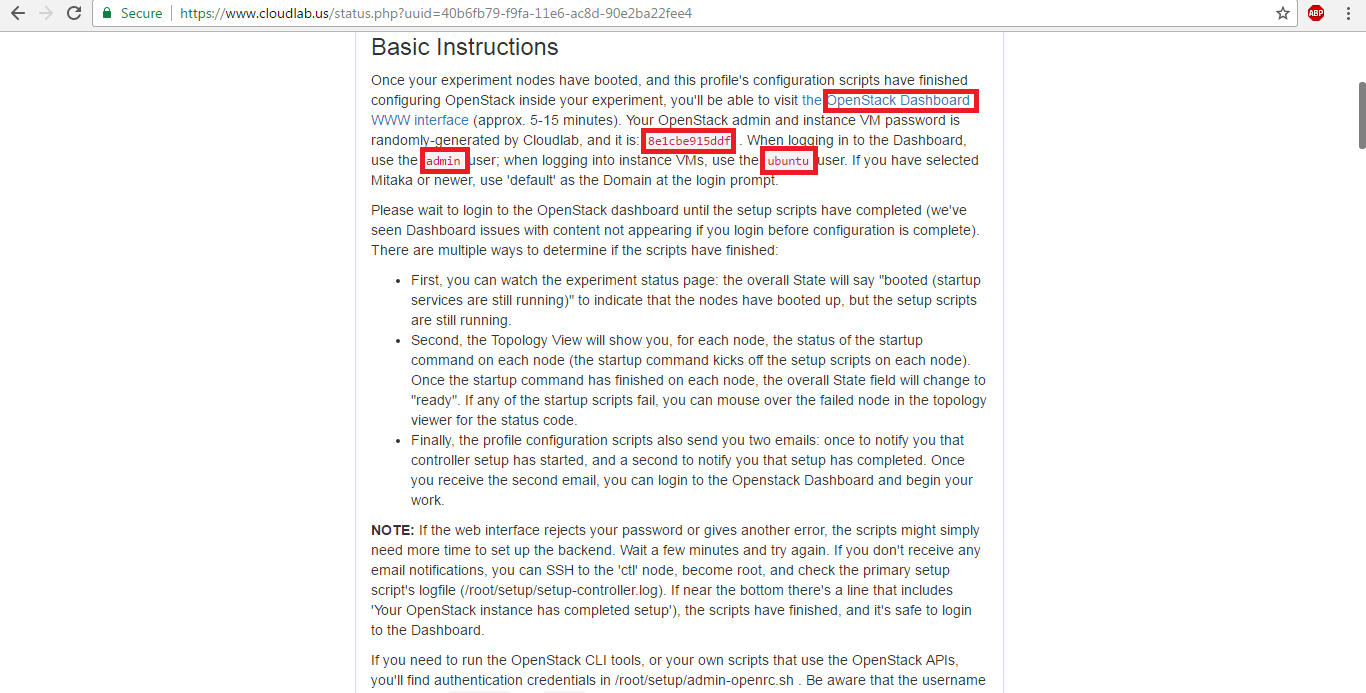


Figure 6: OpenStack instance details in CloudLab

To login to the OpenStack Dashboard, we can click on the highlighted ink as shown above.

Credentials will be entered as stated above (Figure Displayed below)

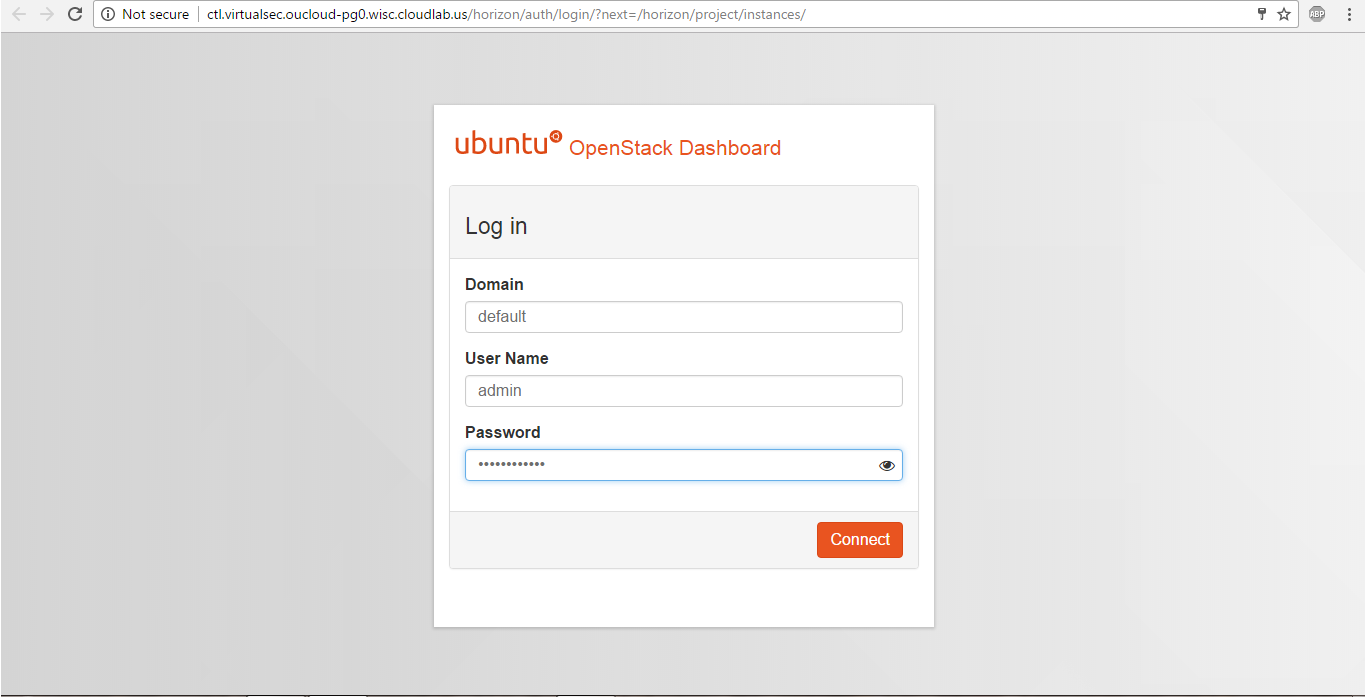


Figure 7: OpenStack Login page

In the next sections, we will go through 3 different processes to create a simple network of one router and one host. We will start by configuring a network, then the corresponding router, followed by configuring the host that is connected to the aforementioned router.

# Section III: Creating a Network

To create a network, once signed in to the OpenStack instance dashboard, Click on Network. Then in the sub-menu, click on Networks. After that click on Create Network. This is displayed in the figure below.

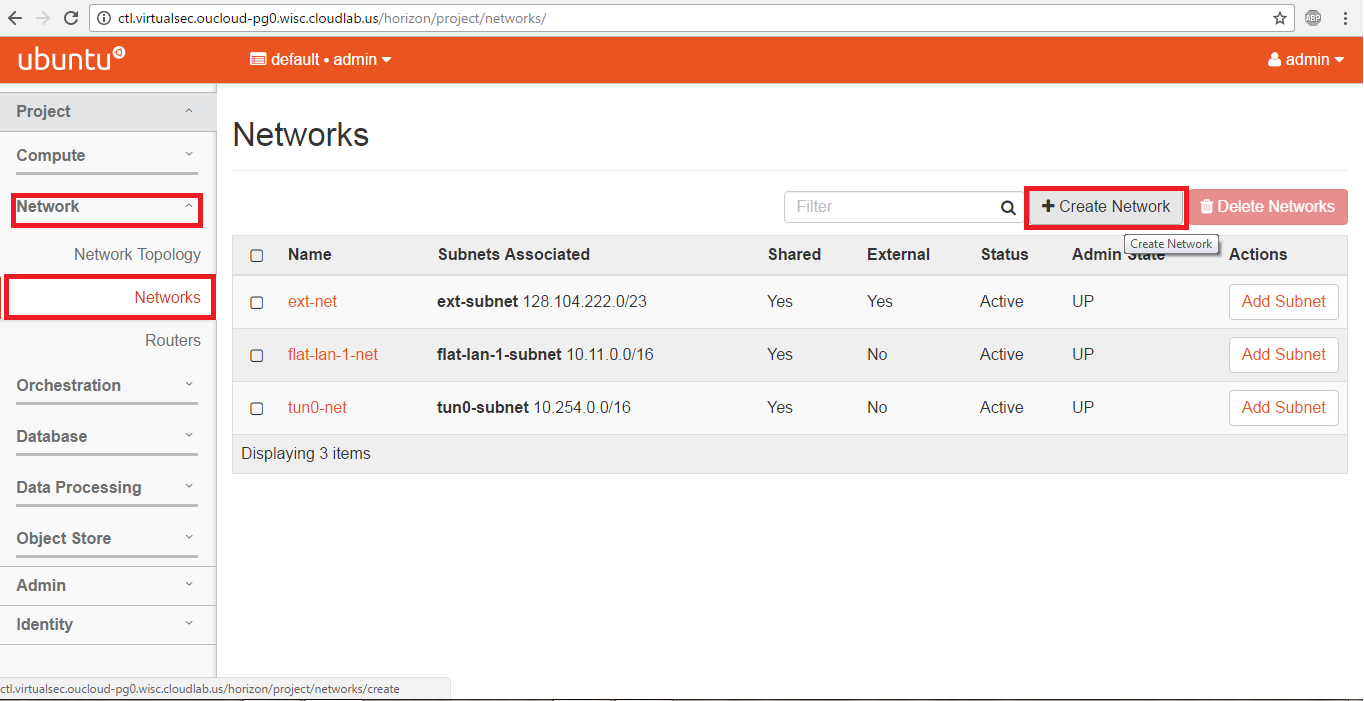


Figure 8: Creating a new network in OpenStack

The ext-net is the external (public) network interface of the instance. Connecting routers/hosts to this network makes them reachable over the public internet.

When the network creation dialog shows, you can enter the details required as per your design. In this instance, we are configuring a VictimNetwork Subnet to be of the network 10.0.0.0/24

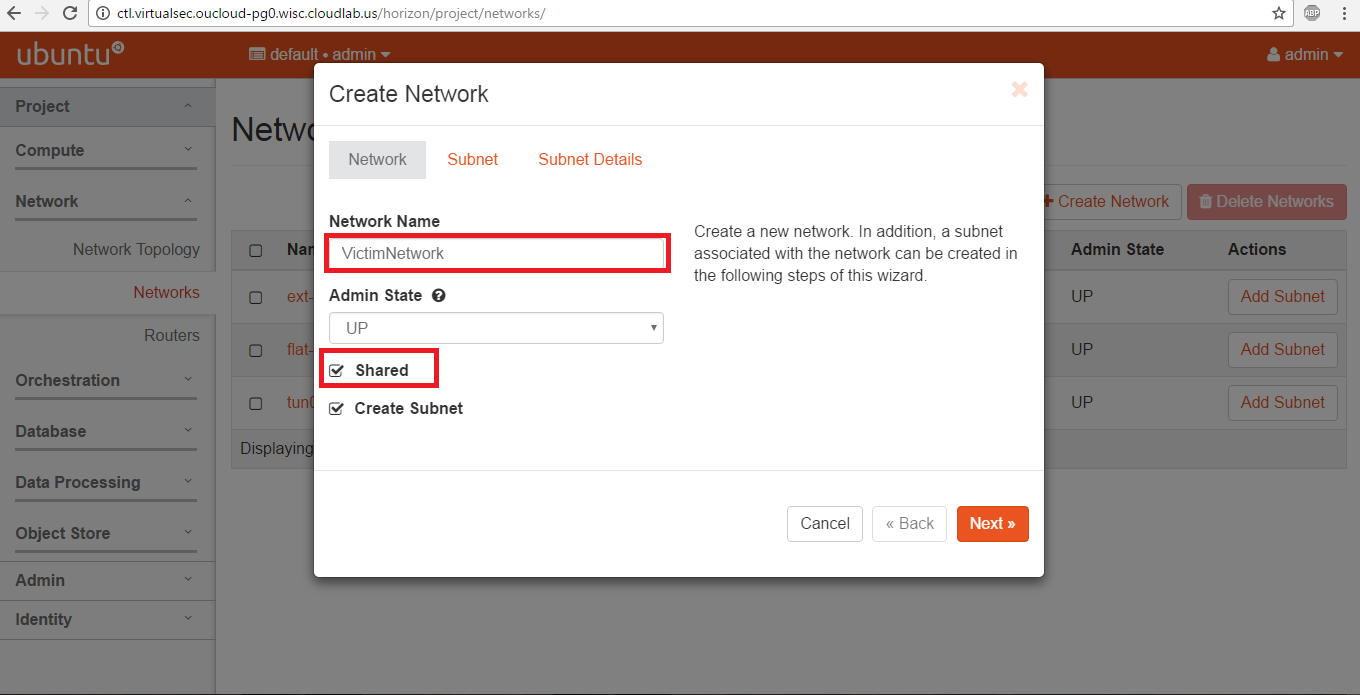


Figure 9: Input network parameters of name and nature in network creation

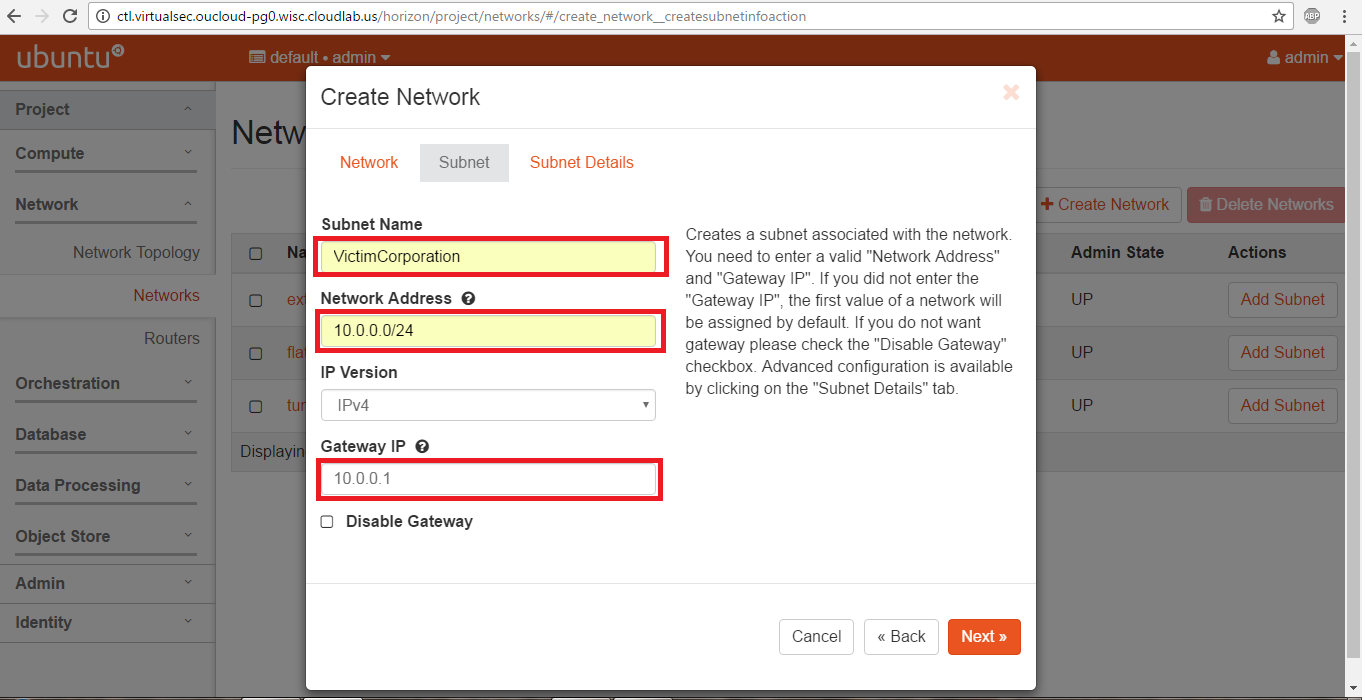


Figure 10: Input subnet details relevant to the created network

Upon the completion of each dialog, click on next. When done, click on finish for the settings to be saved.

The procedure can be duplicated to create networks and subnets as desired, as well as control DHCP allocation method if needed.

# Section IV: Creating a Router

Once the network has been created, to create and configure a new router instance, in the Sub-menu of Network, click on Routers, followed by Create Router as displayed below.

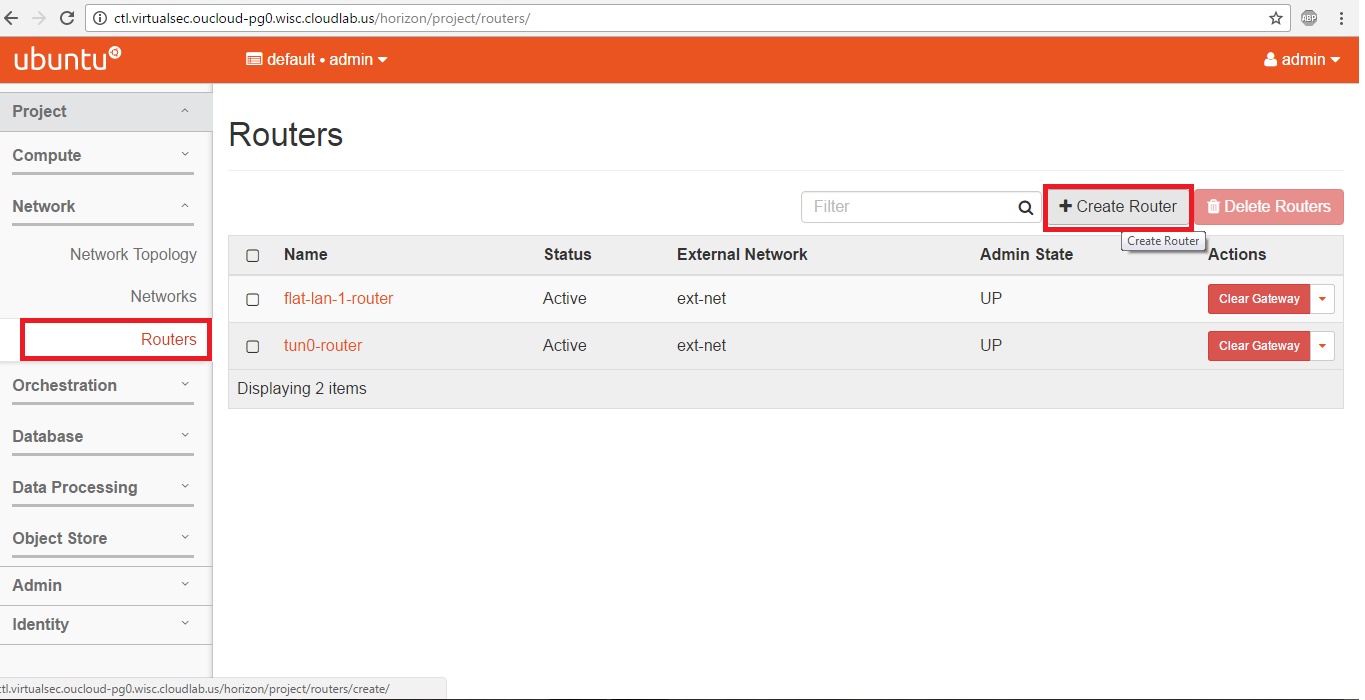


Figure 11: OpenStack creating a new router instance

In the new pop-up dialog, give the router a new name, and connect it to the public external network if need be. In our case, we will connect it to the ext-net for reasons to be explained in the VM management section.

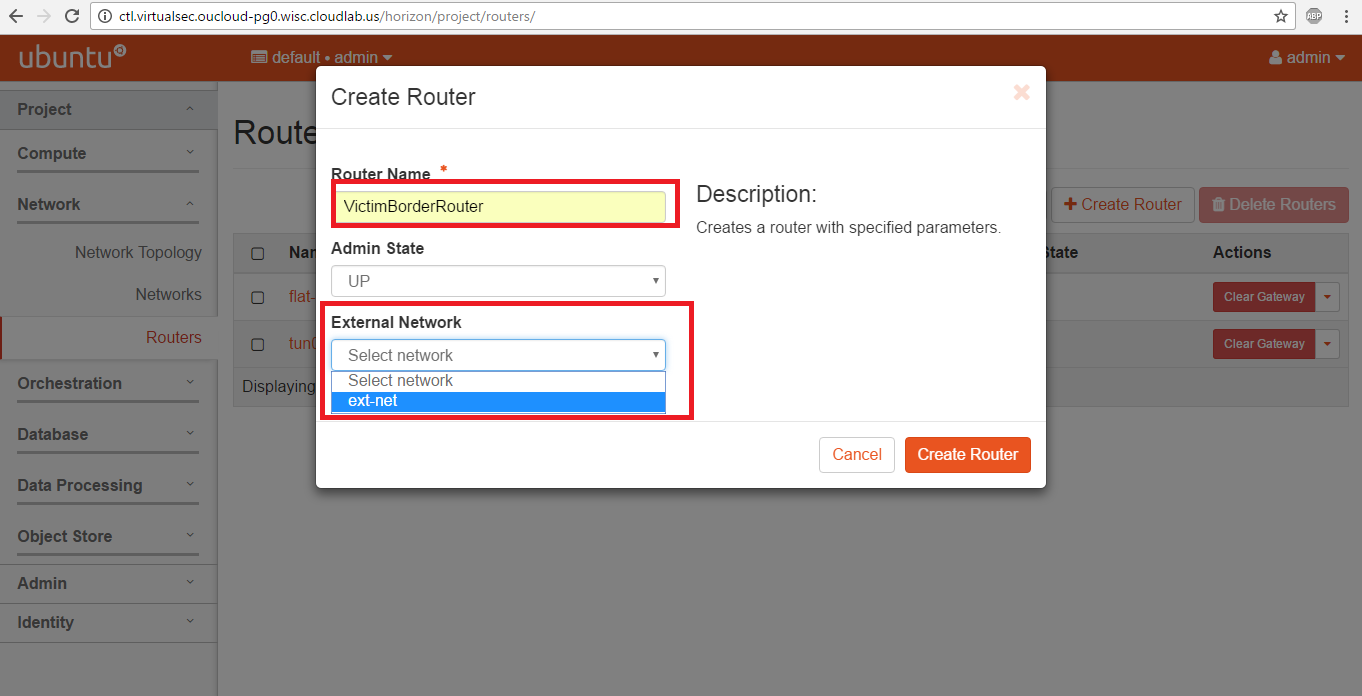


Figure 12: OpenStack modifying new router parameters

After clicking on create, to configure a new interface on the router that is connected to the newly created network, click on the desired router from the router list, then click on Instances followed by Add Interface, as displayed in the figure below.

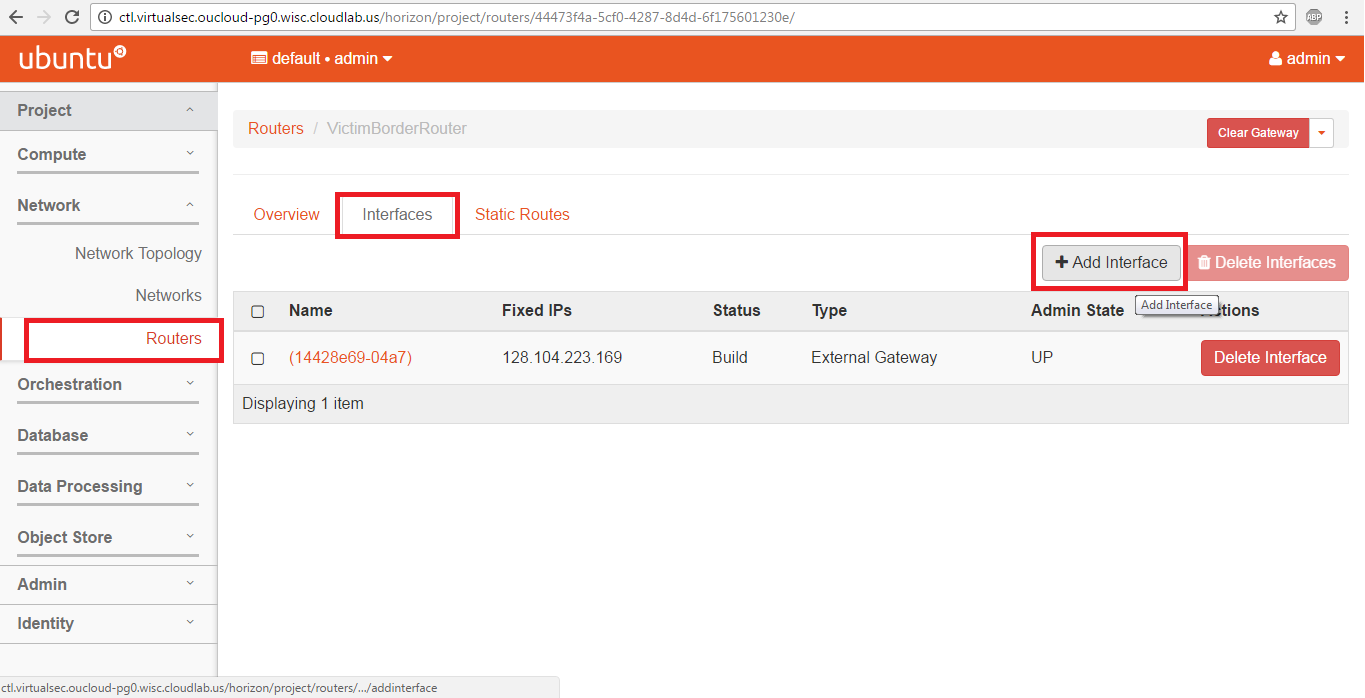


Figure 13:OpenStack connecting routers to new networks

In the new popup dialog, Select which network would you like the new interface to reside on.

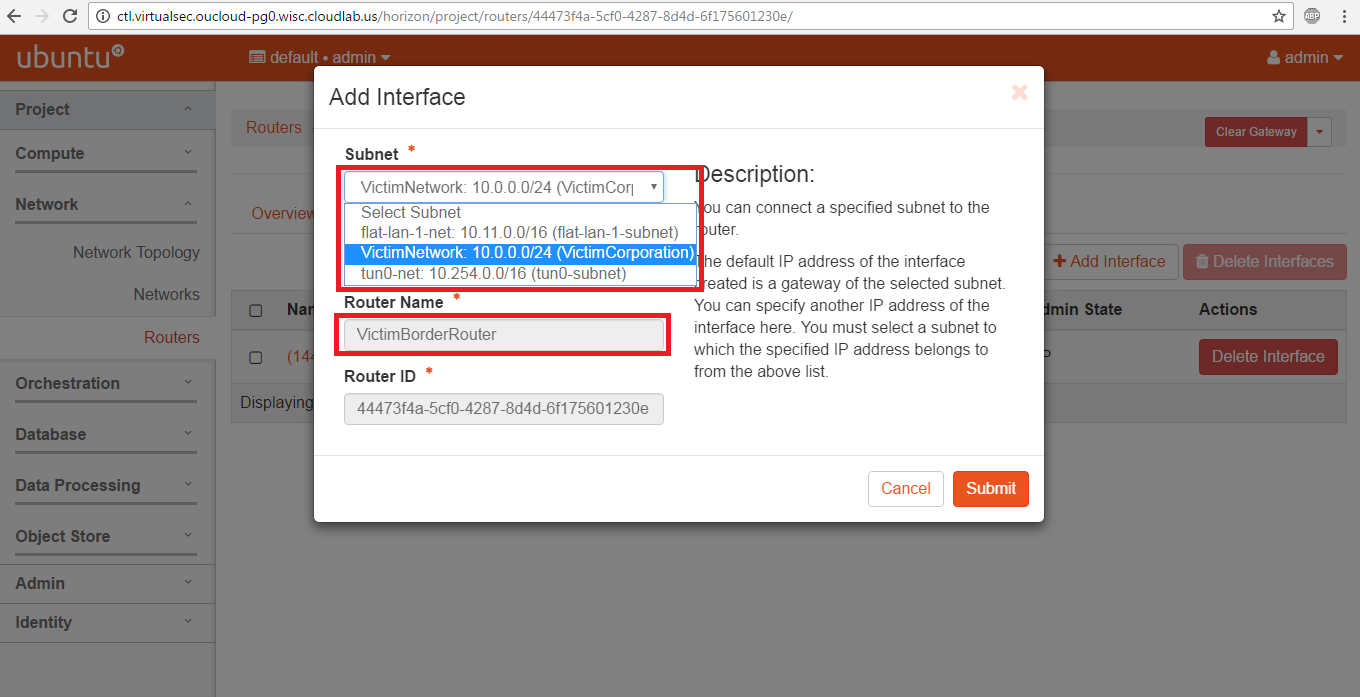


Figure 14: Choosing which new network router should connect to in OpenStack

In our displayed configurations, we have just created a new VictimGateRouters that has 2 interfaces: One is connected to the public network, while the other is connected to the VictimNetwork we created earlier. Next we will create a new VM and connect it to the mentioned network

# Section V: Creating a new VM

To create a new VM, click on Compute in the left-menu, followed by Instances in the sub-menu, then click on Launch Instance as displayed below.

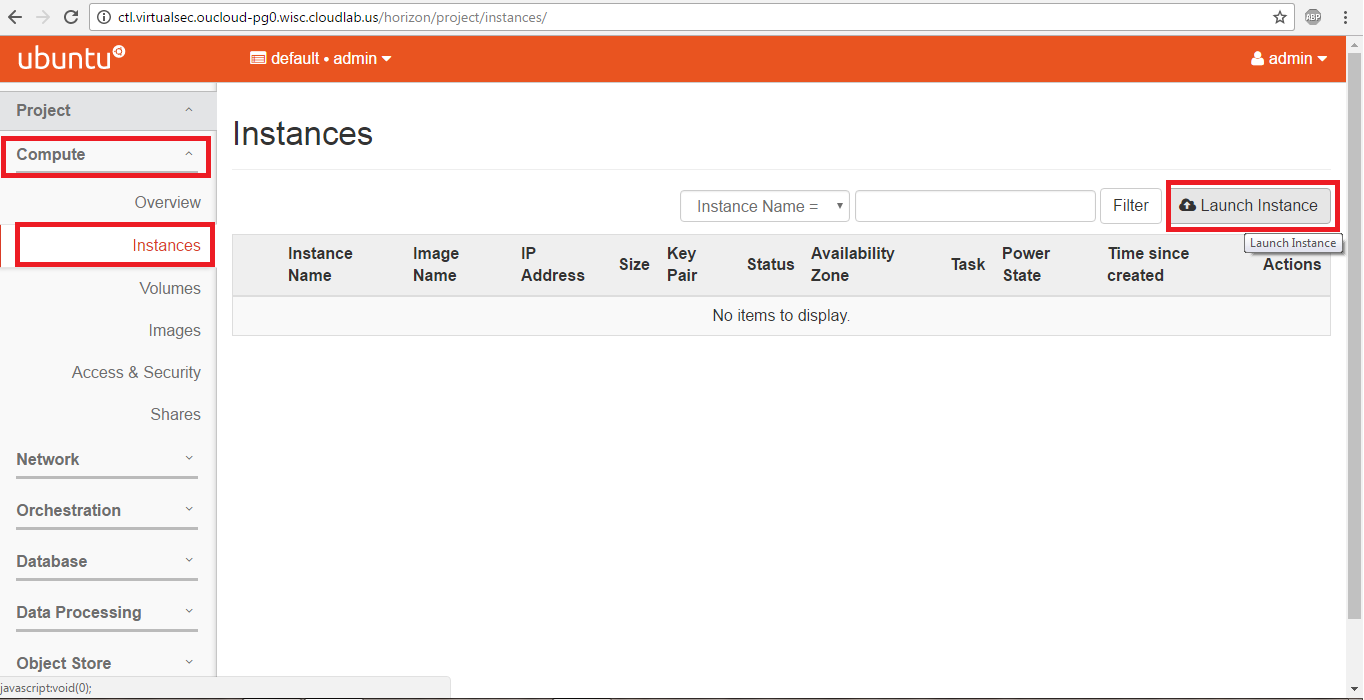


Figure 15: OpenStack launching a new VM instance

The newly configured instance will hold either Ubuntu14 (Trusty) or Ubuntu12 (Manila) image.

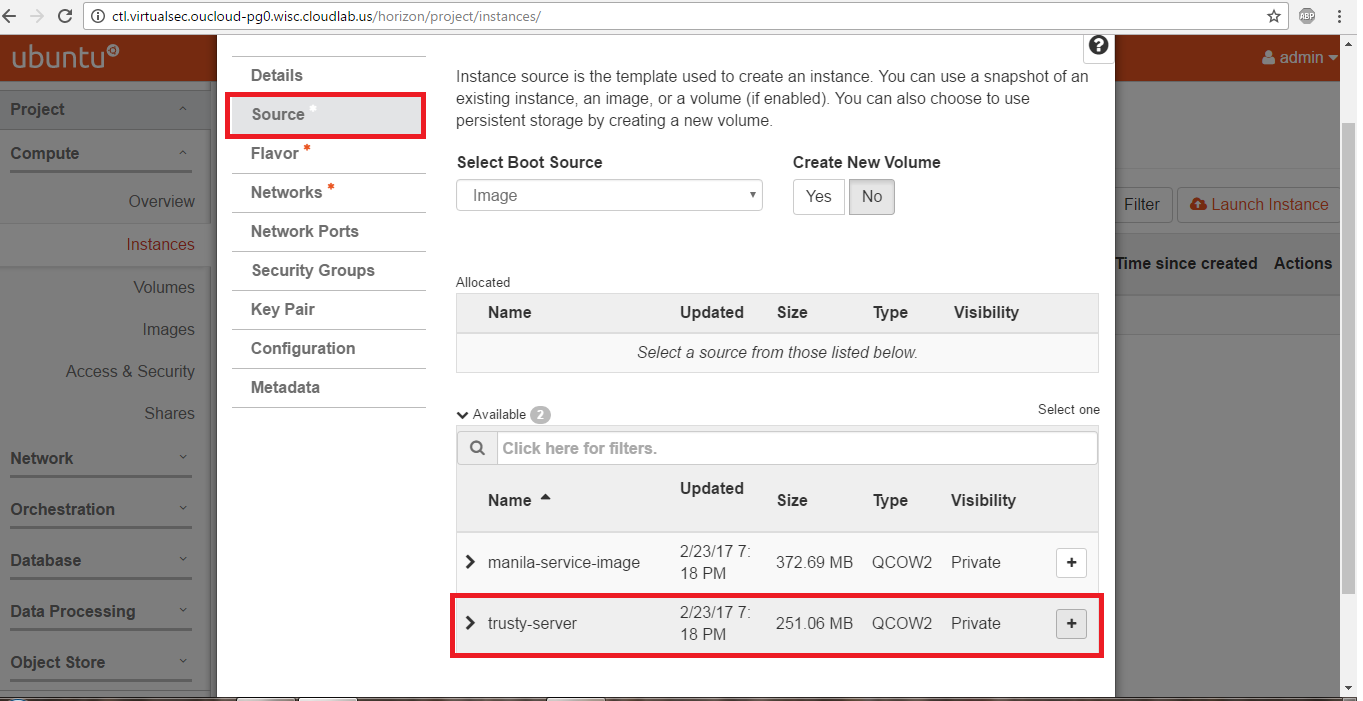


Figure 16: Modifying image source for new VM

It is advised in the configuration of which image to install *Not to select the tiny image* as you will face errors due to size mismatch with the disk space, and you will not be able to boot your VM. You can choose your desired image and option by clicking on the “+” option that is present to the edge of the option. This is also displayed in our screenshots below. With custom images, you may have to choose a larger size. Medium is the recommended size for custom images.

For our purposes, we will select the Trusty image, with the “Small” flavor. The major difference in the flavors of the Chosen Operating System is the installed services that will run by default on the first spin of the VM.

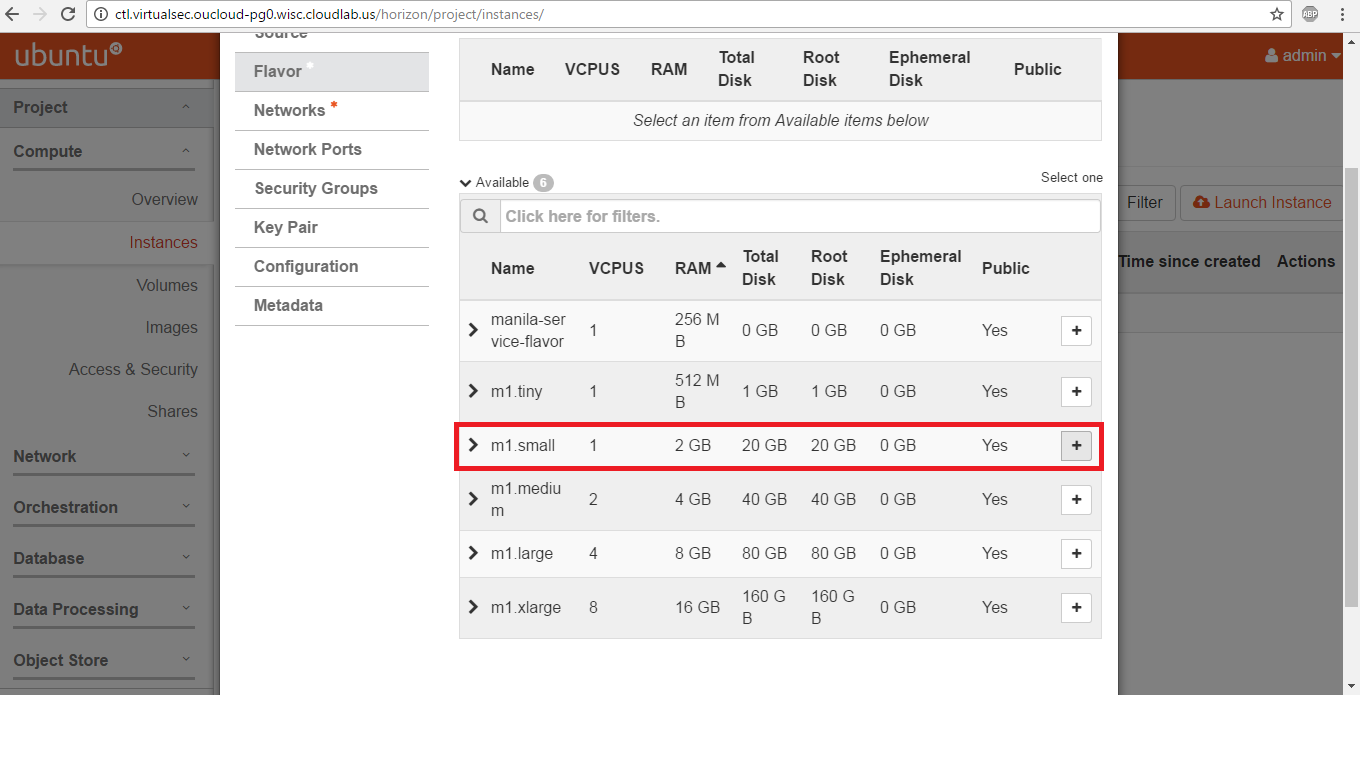


Figure 17: Modifying flavor type for new VM

The third option to configure on the VM instance before launching is creating an interface to connect it to a desired network. In this case, we will connect the VM to the Victim Network, as displayed in our screenshot below.

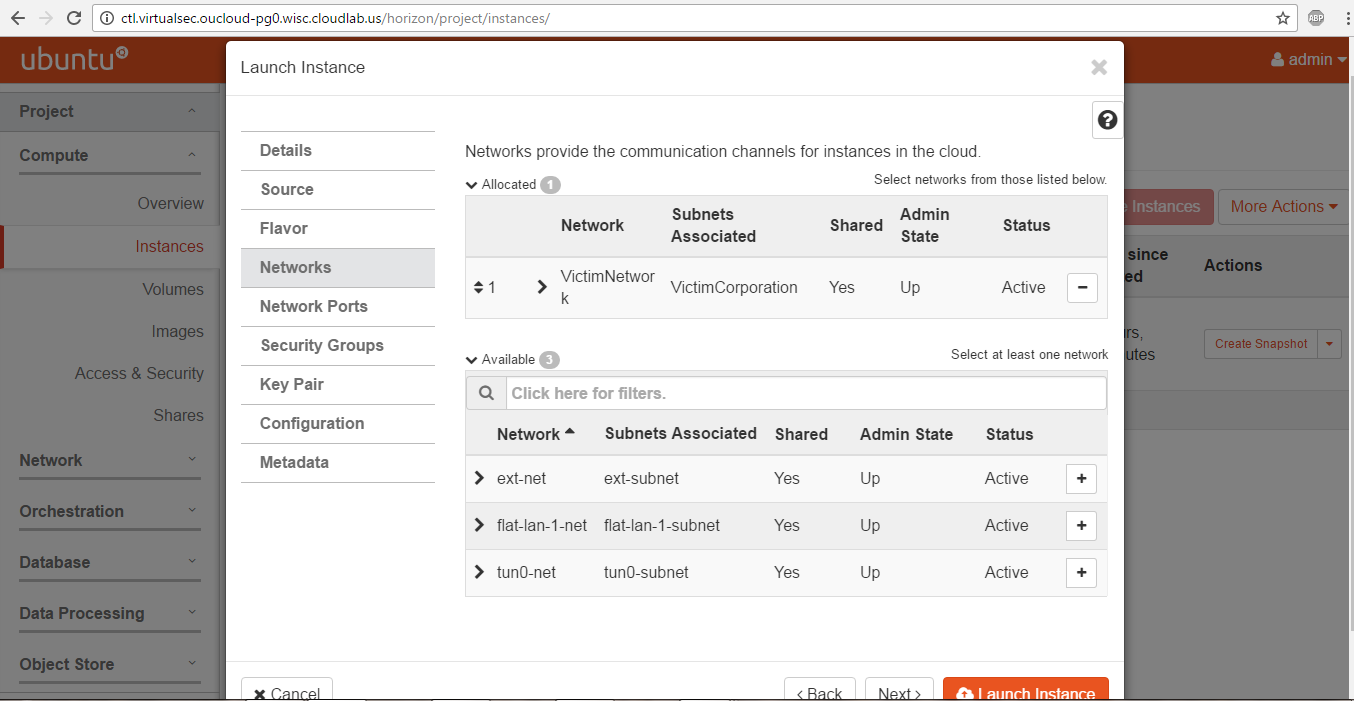


Figure 18: Attaching new VM to desired network

Once this is done, Click on “Launch Instance”. Within a couple of minutes, your VM should be ready to log in to.

You can snapshot the VM by clicking “Create Snapshot”.

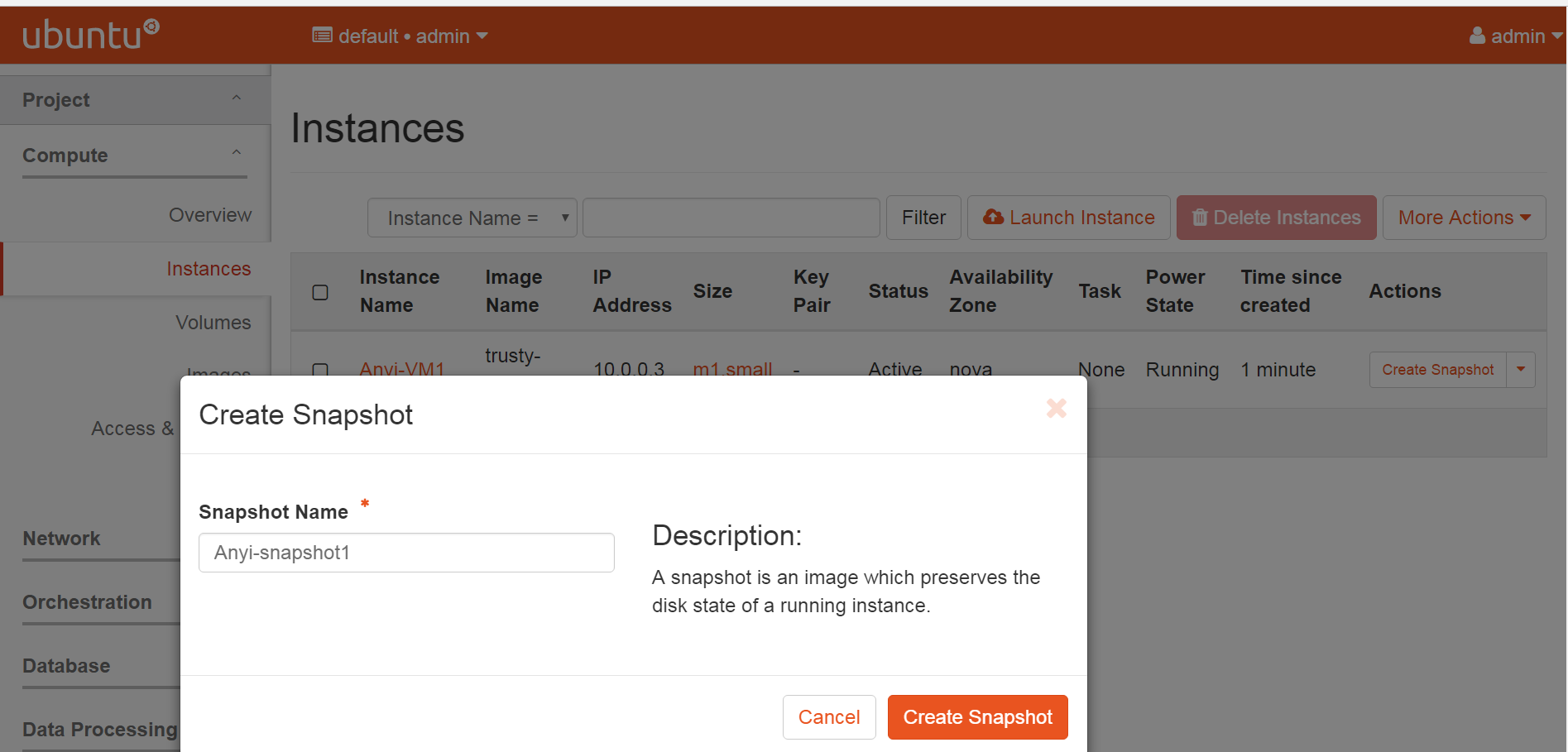


Figure 19: OpenStack creating image snapshots

# Section VI: Managing the created VM

## Option A: Managing through the default console of VM

There are different options for managing the VM. The simplest and default way that is deployed by OpenStack is through the “Console” Tab, when you click on the instance you want to work on from the Instances list. The corresponding screenshot is displayed below.

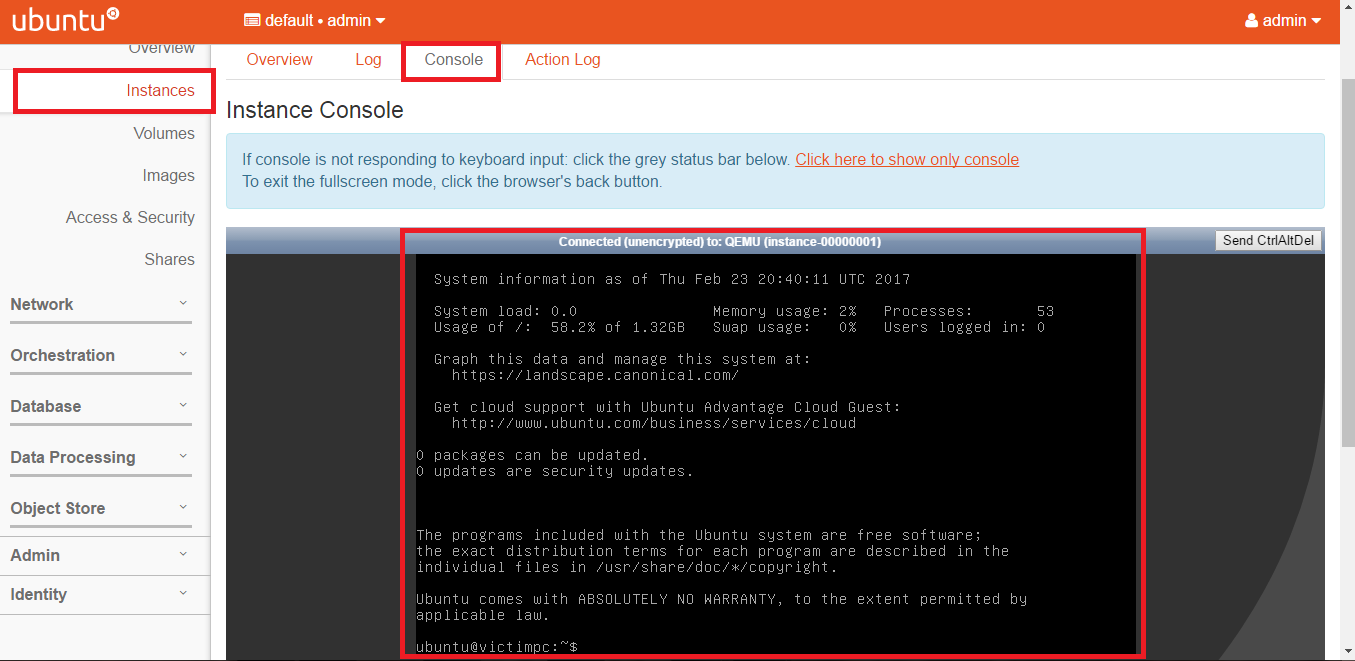


Figure 20: Viewing a VM console in OpenStack

However, we have noticed that access and configuration in the console window is considerably slow. A command will take long time to finish, and the output may not show. The console session more often than not suffers from hanging, and we end up having to open a new dashboard instance in the browser to use the console again. Depending on network firewalls and restrictions, you may not be able to access the console at all.

Hence, we highly recommend Option B.

## Option B: Managing through SSH over VM Public IP

As mentioned before, we have recommended increasing the number of public IPs allocation in the OpenStack. This will especially come in handy in this step, as we attempt to SSH to our created VMs. To perform that, we will go through the process of associating a public IP to the VM (same as configuring a NAT for the VM IP). Then we can SSH from Putty/SecureCRT/MobaXterm to the machine on its public IP. Once initial configurations have been completed, we dissociate the public IP and free it for use.

To associate a public IP to the VM, we start by click on Instances in sub-menu of compute. Then, in the options of the desired VM, click on the drop down arrow, and choose “Associate Floating IP”. A screenshot of this step can be seen below.

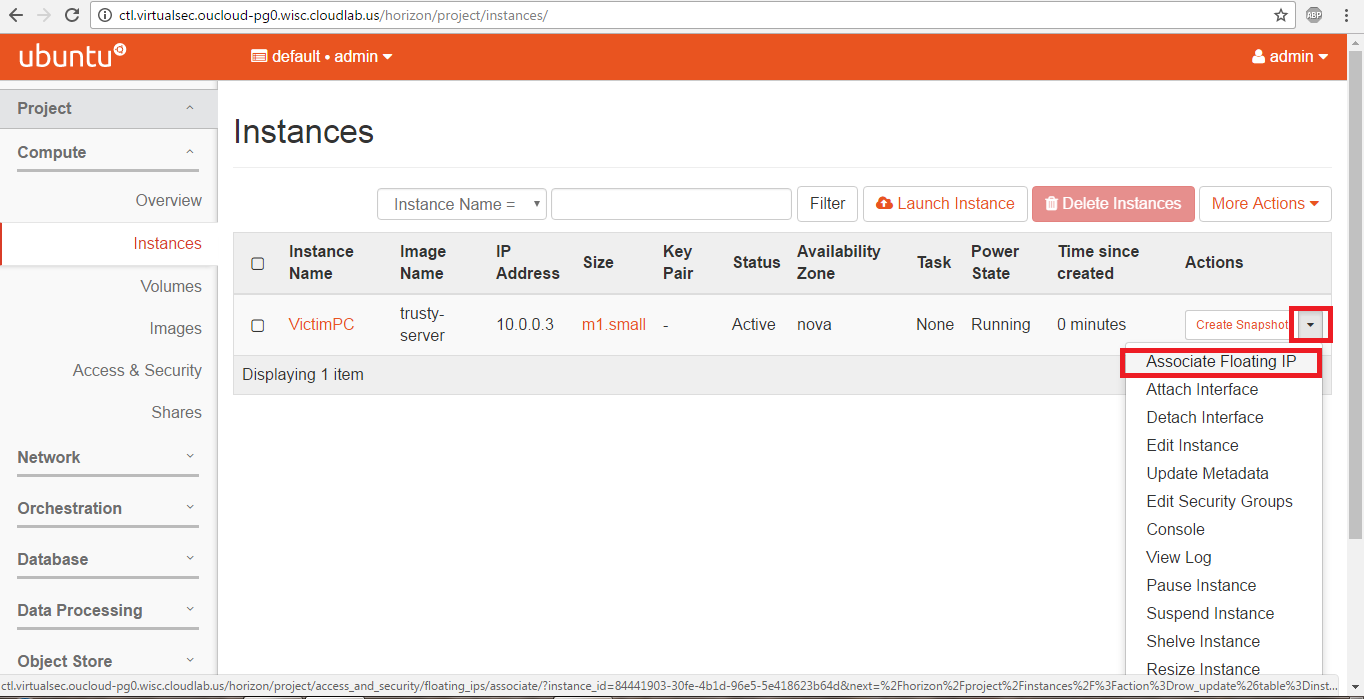


Figure 21: Associating a Floating IP for a VM using OpenStack GUI

In the default case of no previously allocated public IP addresses in the instance, we will need to allocate it by pressing on the “+” sign in the following dialog pop-up next to the IP address field.

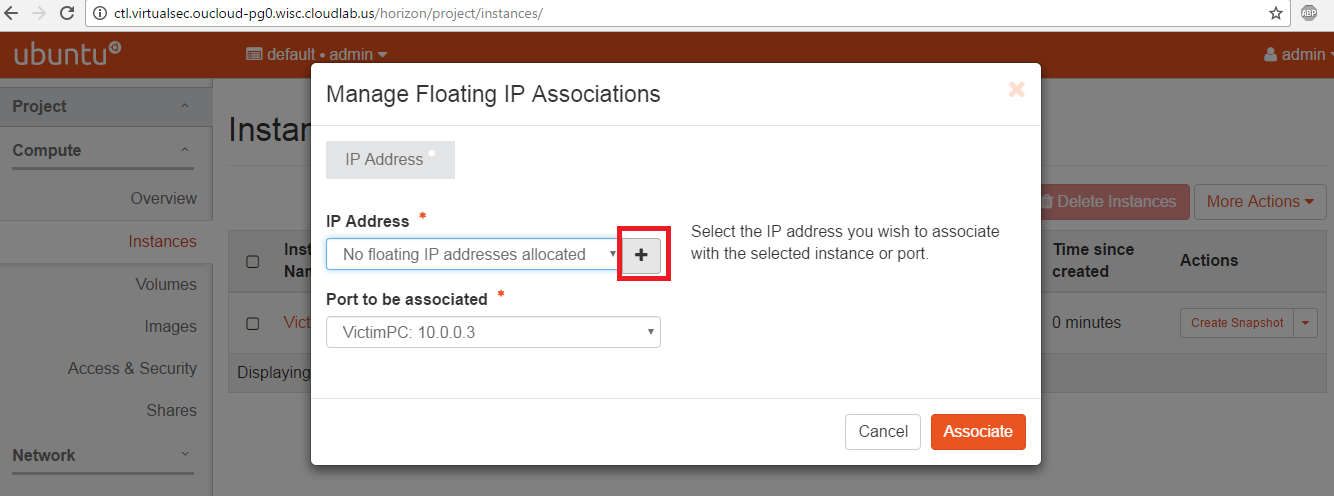


Figure 22: Creating a new Floating IP using OpenStack GUI

Once a new Public IP has been allocated, we can choose it, and choose to which port we would like to associate it, then click on Associate. Once that is done, we can see the new IP address in the instance details. The steps are depicted below.

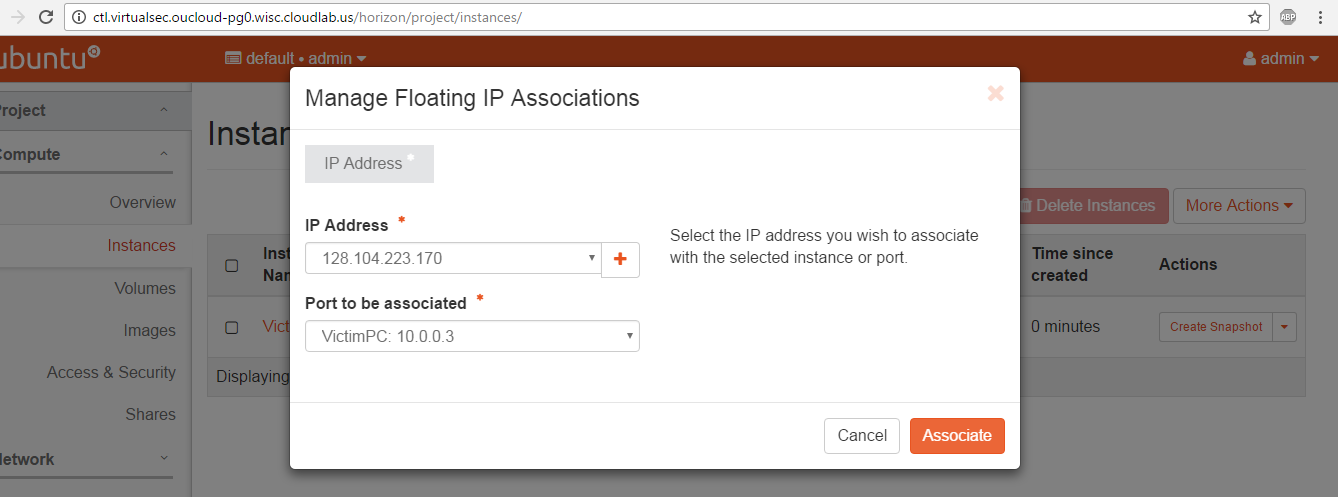


Figure 23: Assign the Floating IP to VM in OpenStack GUI

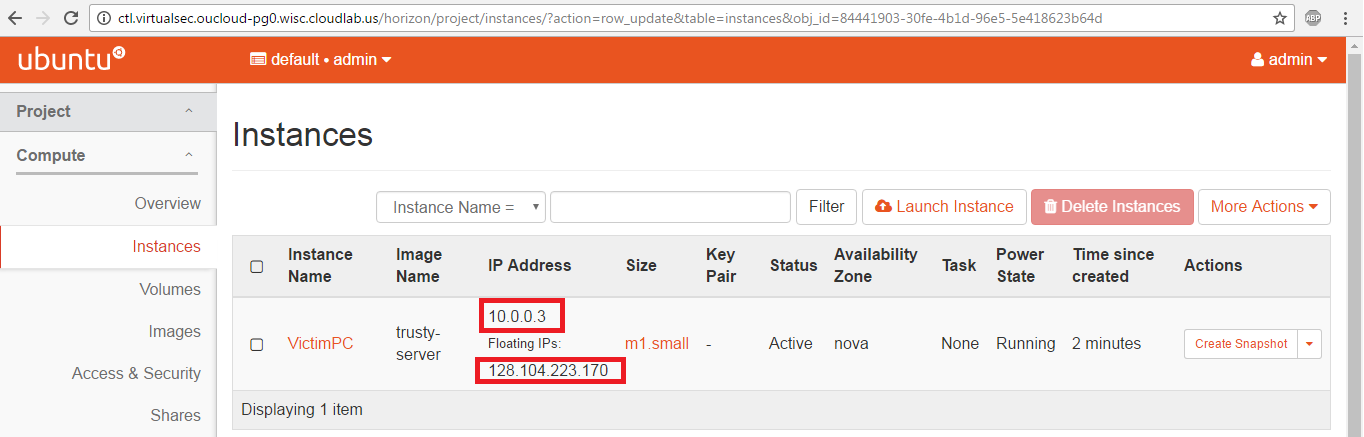


Figure 24: VM status if Floating IP association is successful.

With this step done, we can SSH using our local machine emulator (such as Putty) to the public IP of the VM to configure it. This is highly recommended as it is highly responsive, and mimics our manipulation in a real environment. The downside is in the case the associated interface of the VM fails, then the SSH session will fail correspondingly, and the only access is back via the Console.

Once all the above steps are done, we have configured a simple network of router and VM, connected to the public internet via the gateway, and mimicked the corporate connection. The Topology can be viewed by clicking on Network, followed by Network Topology, as seen in the image below.



Figure 25: Network Topology view using OpenStack GUI

# Section VII: Managing OpenStack Instances Using Python

## General Description

In this project, we leverage OpenStack instances in CloudLab to construct our virtual lab. Figure 26 illustrates the architecture of our design. In particular, we first connect to the EC2 instance in AWS (via SSH). Then, we connect to the physical machine of OpenStack CTL (via HTTP Authentication protocols). Note that our CloudLab project uses two physical machines, OpenStack CTL and OpenStack Compute (CP-1). The OpenStack CTL serves as the control and storage node, which provides OpenStack services, run the administrative tools/commands, and store the VM images; while OpenStack Compute provides the platform to run and instantiate the virtual network and its artifacts. As we authenticate with the CTL, we run the Python code from Amazon VM instance and manipulate the VM instances, networks, subnets, flavors, IP addresses within OpenStack.

The list of Python code is listed below:



Figure 26: General Network Diagram of our access from AWS to the CTL

* **ProfileScript.py** (Appendix A) -- This file profiles (fingerprints) the OpenStack instance and generates an XML file that contains the information that is necessary to recreate virtual network.
* **ReloadScript.py** (Appendix B) -- This file automatically re-constructs an OpenStack instance based on an input XML file. It requires a connection to be established with a new OpenStack instance, on which we will load our environment
* **MasterScript.py** (Appendix C) -- This file contains all basic functionalities used by the system, such as listing, adding, removing and updating the artifacts of the OpenStack instances.
* **TesterScript.py** (Appendix D) -- This file contains the testing script as the proof-of-concept interface to test the effectiveness of the implementation.
* **VMSSH.py** (Appendix E) -- This script enables the administrator to login and manage a VM instances declared in OpenStack without the need to expose the machine to the Internet by a floating (world-accessible) IP.
* **MasterScript.py (updated)** (Appendix F) – This script is the updated Master code with added functionality created by the second group of students who worked on this project. Added changes are described later in this document.
* **TesterScript.py (updated)** (Appendix G) – This script is the updated Tester code with added functionality created by the second group of students who worked on this project. Added changes are described later in this document.

In following sections, we describe each program, along with the code snippets and screenshots, to demonstrate their usage. The complete source code can be found in the Appendix section.

## Technical Challenges

In the course of development, several challenges are needed to be tackled. Some can be managed and worked around by us; while others need the future works. We briefly describe them as follows:

### Reachability from Oakland University infrastructure

In the beginning, we faced the challenge to establish an authenticated connection with the OpenStack instance. After performing some network trouble-shooting through Wireshark, tcpdump, and netstat, we identified the root-cause being Oakland Universities’ network perimeter infrastructure filtering out the outbound traffic connected to the non-standard ports of OpenStack (e.g. TCP Port 5000). There are two ways to solve this issue. Option 1: we connect (SSH) to an Amazon AWS instance (12 months free for the 1st time user) and then connect (SSH) to CloudLab from the AWS instance. Option 2: we can open a ticket with UTS to request a number of ports to be allowed for our experiment. We chose Option1 for our current implementation.

### Lack/inconsistency of documentation

Considering that we have built on top of the OpenStackSDK package, we have faced several problems as OpenStack have been migrating several of their components to newer code bases. Some functions were not completed, and the documentation of other functions was not accurate when deployed between Python2.7 and Python3.x

### Inability to manipulate routing tables

In the current version of the code framework, the user is not able to dynamically manipulate the routing tables over the host instances. We have noted this as our upcoming future works with Dr. Liu to contribute to the OpenStack Community.

### Router Interfaces

When recreating router interfaces using our scripts, we noticed that their status was automatically set to “DOWN” instead of “ACTIVE”. When the router interfaces have this status, we are unable to associate floating IPs to VMs on networks that are connected to those routers. After looking through the OpenStack documentation, we found no way to change this status using our code. The only way to create router interfaces with the “ACTIVE” status is to create them manually through the OpenStack dashboard.

## Profiling the OpenStack Instance (**ProfileScript.py**)

This Python script profiles an OpenStack running instance. It defines the function Profile\_OpenStack which takes as input an established an authenticated connection to the OpenStack instance. As a final output, the function delivers an XML file that is time-stamped by the time at which the profiling function was done.

The algorithm described in the above pseudocode is explained as follows:

1. Start a blank XML file rooted as “Profile”;
2. Retrieve all the current network interfaces through the OpenStack API call conn.network.ports(), which returns an iterator over all ports/interfaces configured by the topology;
3. Iterate through each port to get the information of the attributes, such as the Interface\_ID, Interface\_Name, Device\_ID (The device to which this interface is connected to), Administrative state (If UP or DOWN) as well as the network, subnet and ip\_address.

Code snippet of the interfaces being processed is shown in the figure below.

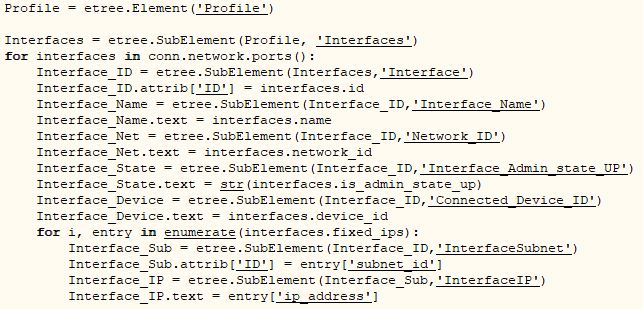


Figure 27: Code snippet showing the profiling of interfaces

1. Two things need to note: First, each Interface instance is considered as a SubElement of the XML Profile, identified by its corresponding unique ID to preserve the data dependecy for the rebuilding section of the framework. The relevant attributes of each interface are classified as SubElements of that Interface ID.

Second, since “fixed\_ips” is an object, it is necessary to access each attributes (could be the objects as well) of this object through the iteration (Same as processing a dictionary). The similar implementation is also applied to elsewhere throughout the code, in order to retrieve the information of the networks, routers, images, and flavors, etc. The subnets were classified through a nested loop during the processing of the networks. This is because each network class is considered to represent a Layer-2 network, which can related to multiple subnets that represent Layer-3 networks. To preserve this mapping relationship between the network and subnets, we have processed the subnets within the Networks block.

The VM instances are processed near the end of the function. For each VM instance, we access it corresponding class identifiers that we need for its re-creation. The attributes include the ID, along with the configured flavors, images, and IP addresses.

***Note:*** *It is worth to note that by using the default XML package of Python and the downloaded LXML package (both installed at the AWS host), the resulting XML file is just a single line that represents the entire XML file. To make the XML file easy to read, we included the* prettify *function that is defined in the figure below, which parses the file and inserts indentation to make the XML file easy to read.*

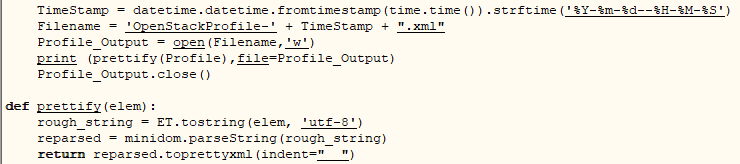


Figure 28: Code snippet of our prettify function and the XML file saving

The output of the function Profile\_OpenStack is a XML file, whose name appended with the timestamp when the instance was profiled. Figure 29 shows the complete XML file.

<?xml version="1.0" ?>

<Profile>

<Interfaces>

<Interface ID="213fbc80-84c7-43e4-acc3-40de8fe434eb">

<Interface\_Name/>

<Network\_ID>5a742f66-0a82-498a-b5f1-5d02240750ec</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>2e5136f9-d4b7-432c-9a19-c882df8baade</Connected\_Device\_ID>

<InterfaceSubnet ID="12a8c86f-5b67-4981-b3de-ad91189cc856">

<InterfaceIP>128.110.155.153</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="3033035d-6905-4169-bd04-d9817e80e877">

<Interface\_Name/>

<Network\_ID>5a742f66-0a82-498a-b5f1-5d02240750ec</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>b91a534e-386f-4d6f-af6b-5af6a79a9303</Connected\_Device\_ID>

<InterfaceSubnet ID="12a8c86f-5b67-4981-b3de-ad91189cc856">

<InterfaceIP>128.110.155.152</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="429b0f25-b1ce-498a-a69f-2462fe5f364e">

<Interface\_Name/>

<Network\_ID>985310aa-f409-435c-b6d6-9834c649f922</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>dhcpc47b6c48-a620-52a0-9f83-b828d5a7b1ec-985310aa-f409-435c-b6d6-9834c649f922</Connected\_Device\_ID>

<InterfaceSubnet ID="5434429b-99b1-47d1-96ca-061c1401d42b">

<InterfaceIP>192.168.0.2</InterfaceIP>

</InterfaceSubnet>

<InterfaceSubnet ID="7e53ac12-b1a1-4b6e-a166-16fc705a7103">

<InterfaceIP>192.168.2.2</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="54935852-cb09-4854-a48a-40d8d2162331">

<Interface\_Name/>

<Network\_ID>37212864-6421-4801-bc16-5f8b9f3a29b6</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>b91a534e-386f-4d6f-af6b-5af6a79a9303</Connected\_Device\_ID>

<InterfaceSubnet ID="034648ae-463c-41a3-9909-b0537060788f">

<InterfaceIP>10.254.0.1</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="5b2ccc5a-4210-4250-9552-4c1dbed84a89">

<Interface\_Name/>

<Network\_ID>7b818451-fc9a-434c-bc55-b6e828998196</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>a7bc24ac-6d4e-4ef5-a913-226b6f69ea8f</Connected\_Device\_ID>

<InterfaceSubnet ID="2e48b6dc-9329-4dc4-a34c-e55bca02acf8">

<InterfaceIP>10.11.10.11</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="6b394a7e-2dfa-42c2-a748-57abef97715e">

<Interface\_Name/>

<Network\_ID>985310aa-f409-435c-b6d6-9834c649f922</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>4c6e67e4-7e2d-44e4-bfae-dc3ca8b42e6b</Connected\_Device\_ID>

<InterfaceSubnet ID="5434429b-99b1-47d1-96ca-061c1401d42b">

<InterfaceIP>192.168.0.3</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="6ec86ebc-1439-47ee-88d5-9a6f74bd71c7">

<Interface\_Name/>

<Network\_ID>7b818451-fc9a-434c-bc55-b6e828998196</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>2e5136f9-d4b7-432c-9a19-c882df8baade</Connected\_Device\_ID>

<InterfaceSubnet ID="2e48b6dc-9329-4dc4-a34c-e55bca02acf8">

<InterfaceIP>10.11.10.3</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="8a4f8fff-a131-4d2e-a197-5ab64d7d6cd9">

<Interface\_Name/>

<Network\_ID>985310aa-f409-435c-b6d6-9834c649f922</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>a97f2ab2-439d-4b4d-91f1-b687a672b855</Connected\_Device\_ID>

<InterfaceSubnet ID="5434429b-99b1-47d1-96ca-061c1401d42b">

<InterfaceIP>192.168.0.4</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="8e2d3fce-7935-45ec-889c-28c5a4f16d9e">

<Interface\_Name/>

<Network\_ID>7b818451-fc9a-434c-bc55-b6e828998196</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>01d88e10-f6f6-404d-b4a3-1c2809fec296</Connected\_Device\_ID>

<InterfaceSubnet ID="2e48b6dc-9329-4dc4-a34c-e55bca02acf8">

<InterfaceIP>10.11.10.9</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="91986092-9429-44ef-aa7b-5c5e2a67534e">

<Interface\_Name/>

<Network\_ID>7b818451-fc9a-434c-bc55-b6e828998196</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>dhcpc47b6c48-a620-52a0-9f83-b828d5a7b1ec-7b818451-fc9a-434c-bc55-b6e828998196</Connected\_Device\_ID>

<InterfaceSubnet ID="2e48b6dc-9329-4dc4-a34c-e55bca02acf8">

<InterfaceIP>10.11.10.4</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="ad98aa68-fc5f-4765-8750-091e79f7345e">

<Interface\_Name/>

<Network\_ID>7b818451-fc9a-434c-bc55-b6e828998196</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>a1c8a506-70db-41b0-b129-cd3ca0260223</Connected\_Device\_ID>

<InterfaceSubnet ID="2e48b6dc-9329-4dc4-a34c-e55bca02acf8">

<InterfaceIP>10.11.10.8</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="dd4cfa4e-b068-401d-8517-efc7fe03c83f">

<Interface\_Name/>

<Network\_ID>7b818451-fc9a-434c-bc55-b6e828998196</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>ea9bbae9-d2cf-4ea4-ace0-8266df97bbb0</Connected\_Device\_ID>

<InterfaceSubnet ID="2e48b6dc-9329-4dc4-a34c-e55bca02acf8">

<InterfaceIP>10.11.10.6</InterfaceIP>

</InterfaceSubnet>

</Interface>

<Interface ID="e83ed4f8-827c-4311-9b94-b49f0e6ea6e9">

<Interface\_Name/>

<Network\_ID>37212864-6421-4801-bc16-5f8b9f3a29b6</Network\_ID>

<Interface\_Admin\_state\_UP>True</Interface\_Admin\_state\_UP>

<Connected\_Device\_ID>dhcpc47b6c48-a620-52a0-9f83-b828d5a7b1ec-37212864-6421-4801-bc16-5f8b9f3a29b6</Connected\_Device\_ID>

<InterfaceSubnet ID="034648ae-463c-41a3-9909-b0537060788f">

<InterfaceIP>10.254.0.2</InterfaceIP>

</InterfaceSubnet>

</Interface>

</Interfaces>

<Networks>

<Network ID="5a742f66-0a82-498a-b5f1-5d02240750ec">

<Network\_Name>ext-net</Network\_Name>

<Network\_Admin\_State\_UP>True</Network\_Admin\_State\_UP>

<Subnet ID="12a8c86f-5b67-4981-b3de-ad91189cc856">

<Subnet\_Name>ext-subnet</Subnet\_Name>

<Subnet\_CIDR>128.110.152.0/22</Subnet\_CIDR>

<Subnet\_IP\_Version>4</Subnet\_IP\_Version>

<Subnet\_IP\_GW>128.110.152.1</Subnet\_IP\_GW>

</Subnet>

</Network>

<Network ID="985310aa-f409-435c-b6d6-9834c649f922">

<Network\_Name>Test\_Net</Network\_Name>

<Network\_Admin\_State\_UP>True</Network\_Admin\_State\_UP>

<Subnet ID="5434429b-99b1-47d1-96ca-061c1401d42b">

<Subnet\_Name>Test\_Sub</Subnet\_Name>

<Subnet\_CIDR>192.168.0.0/23</Subnet\_CIDR>

<Subnet\_IP\_Version>4</Subnet\_IP\_Version>

<Subnet\_IP\_GW>192.168.0.1</Subnet\_IP\_GW>

</Subnet>

<Subnet ID="7e53ac12-b1a1-4b6e-a166-16fc705a7103">

<Subnet\_Name>Test\_Sub\_2</Subnet\_Name>

<Subnet\_CIDR>192.168.2.0/23</Subnet\_CIDR>

<Subnet\_IP\_Version>4</Subnet\_IP\_Version>

<Subnet\_IP\_GW>192.168.2.1</Subnet\_IP\_GW>

</Subnet>

</Network>

<Network ID="7b818451-fc9a-434c-bc55-b6e828998196">

<Network\_Name>flat-lan-1-net</Network\_Name>

<Network\_Admin\_State\_UP>True</Network\_Admin\_State\_UP>

<Subnet ID="2e48b6dc-9329-4dc4-a34c-e55bca02acf8">

<Subnet\_Name>flat-lan-1-subnet</Subnet\_Name>

<Subnet\_CIDR>10.11.0.0/16</Subnet\_CIDR>

<Subnet\_IP\_Version>4</Subnet\_IP\_Version>

<Subnet\_IP\_GW>10.11.10.3</Subnet\_IP\_GW>

</Subnet>

</Network>

<Network ID="37212864-6421-4801-bc16-5f8b9f3a29b6">

<Network\_Name>tun0-net</Network\_Name>

<Network\_Admin\_State\_UP>True</Network\_Admin\_State\_UP>

<Subnet ID="034648ae-463c-41a3-9909-b0537060788f">

<Subnet\_Name>tun0-subnet</Subnet\_Name>

<Subnet\_CIDR>10.254.0.0/16</Subnet\_CIDR>

<Subnet\_IP\_Version>4</Subnet\_IP\_Version>

<Subnet\_IP\_GW>10.254.0.1</Subnet\_IP\_GW>

</Subnet>

</Network>

</Networks>

<Routers>

<Router ID="2e5136f9-d4b7-432c-9a19-c882df8baade">

<Router\_Name>flat-lan-1-router</Router\_Name>

<Router\_Admin\_State\_UP>True</Router\_Admin\_State\_UP>

<Zone>nova</Zone>

<Interface ID="213fbc80-84c7-43e4-acc3-40de8fe434eb">

<IP>128.110.155.153</IP>

</Interface>

<Interface ID="6ec86ebc-1439-47ee-88d5-9a6f74bd71c7">

<IP>10.11.10.3</IP>

</Interface>

</Router>

<Router ID="b91a534e-386f-4d6f-af6b-5af6a79a9303">

<Router\_Name>tun0-router</Router\_Name>

<Router\_Admin\_State\_UP>True</Router\_Admin\_State\_UP>

<Zone>nova</Zone>

<Interface ID="3033035d-6905-4169-bd04-d9817e80e877">

<IP>128.110.155.152</IP>

</Interface>

<Interface ID="54935852-cb09-4854-a48a-40d8d2162331">

<IP>10.254.0.1</IP>

</Interface>

</Router>

</Routers>

<Images>

<Image Name="manila-service-image">

<ContainerFormat>bare</ContainerFormat>

<DiskFormat>qcow2</DiskFormat>

</Image>

<Image Name="trusty-server">

<ContainerFormat>bare</ContainerFormat>

<DiskFormat>qcow2</DiskFormat>

</Image>

</Images>

<Flavors>

<Flavor Name="m1.tiny">

<VCPU>1</VCPU>

<Disk>1</Disk>

<RAM>512</RAM>

</Flavor>

<Flavor Name="manila-service-flavor">

<VCPU>1</VCPU>

<Disk>0</Disk>

<RAM>256</RAM>

</Flavor>

<Flavor Name="m1.small">

<VCPU>1</VCPU>

<Disk>20</Disk>

<RAM>2048</RAM>

</Flavor>

<Flavor Name="m1.medium">

<VCPU>2</VCPU>

<Disk>40</Disk>

<RAM>4096</RAM>

</Flavor>

<Flavor Name="m1.large">

<VCPU>4</VCPU>

<Disk>80</Disk>

<RAM>8192</RAM>

</Flavor>

<Flavor Name="m1.xlarge">

<VCPU>8</VCPU>

<Disk>160</Disk>

<RAM>16384</RAM>

</Flavor>

</Flavors>

<Instances>

<Instance ID="a7bc24ac-6d4e-4ef5-a913-226b6f69ea8f">

<Name>T-4</Name>

<Status>ACTIVE</Status>

<Flavor>m1.small</Flavor>

<Image>manila-service-image</Image>

<Interface ID="5b2ccc5a-4210-4250-9552-4c1dbed84a89">

<IP>10.11.10.11</IP>

</Interface>

</Instance>

<Instance ID="01d88e10-f6f6-404d-b4a3-1c2809fec296">

<Name>T-3</Name>

<Status>ACTIVE</Status>

<Flavor>m1.small</Flavor>

<Image>manila-service-image</Image>

<Interface ID="8e2d3fce-7935-45ec-889c-28c5a4f16d9e">

<IP>10.11.10.9</IP>

</Interface>

</Instance>

<Instance ID="a1c8a506-70db-41b0-b129-cd3ca0260223">

<Name>T-2</Name>

<Status>ACTIVE</Status>

<Flavor>m1.small</Flavor>

<Image>manila-service-image</Image>

<Interface ID="ad98aa68-fc5f-4765-8750-091e79f7345e">

<IP>10.11.10.8</IP>

</Interface>

</Instance>

<Instance ID="ea9bbae9-d2cf-4ea4-ace0-8266df97bbb0">

<Name>T-1</Name>

<Status>ACTIVE</Status>

<Flavor>m1.small</Flavor>

<Image>manila-service-image</Image>

<Interface ID="dd4cfa4e-b068-401d-8517-efc7fe03c83f">

<IP>10.11.10.6</IP>

</Interface>

</Instance>

<Instance ID="a97f2ab2-439d-4b4d-91f1-b687a672b855">

<Name>Test\_2</Name>

<Status>ACTIVE</Status>

<Flavor>m1.medium</Flavor>

<Image>manila-service-image</Image>

<Interface ID="8a4f8fff-a131-4d2e-a197-5ab64d7d6cd9">

<IP>192.168.0.4</IP>

</Interface>

</Instance>

<Instance ID="4c6e67e4-7e2d-44e4-bfae-dc3ca8b42e6b">

<Name>Test\_1</Name>

<Status>ACTIVE</Status>

<Flavor>m1.small</Flavor>

<Image>trusty-server</Image>

<Interface ID="6b394a7e-2dfa-42c2-a748-57abef97715e">

<IP>192.168.0.3</IP>

</Interface>

</Instance>

</Instances>

</Profile>

Figure 29: Sample from the XML File produced by the Script

## Re-Construct OpenStack Instance from XML (**ReloadScript.py**)

This python script is saved under the ReloadScript.py file. This file defines the function Create\_From\_XML, which takes as a XML file as the input, established and authenticated connection to the OpenStack, and reconstruct the Openstack instance based on the profile of the XML file. As the result, a new OpenStack instance will be created, which contains the topology specified by the XML file.

The algorithm described in the above pseudocode is explained as follows:

1. The function loops through the XML file, taking each of the SubElements, create a new dictionary that contains <key:value> pair that indicates the type of information being retrieved. For example, in the following code snippet below, we extract the router information by parsing the XML tags ID, reading the corresponding texts of its attributes. The main dictionary key is the element tag ID, then, we iterate through its children tags to get the result and store it in the dictionary with corresponding values and Key value pairs in a nested dictionary implementation.

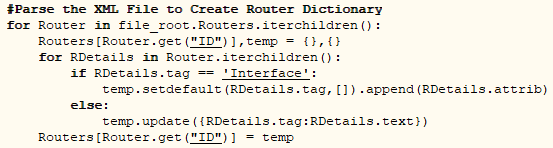


Figure 30: Code Snippet showing how Routers are parsed from the XML to create their dictionary

1. After all tags of the XML file has been parsed and the values of the attributes have been saved in the dictionaries, the function then access each dictionary separately to create the corresponding entities. The order of operations (dependency) is critical. For example, a subnet cannot be created before its corresponding network been created. Similarly, an instance cannot be created before its corresponding flavors, images, subnets and interfaces been created. The function below shows how the Networks and Subnet dictionaries be instantiated.

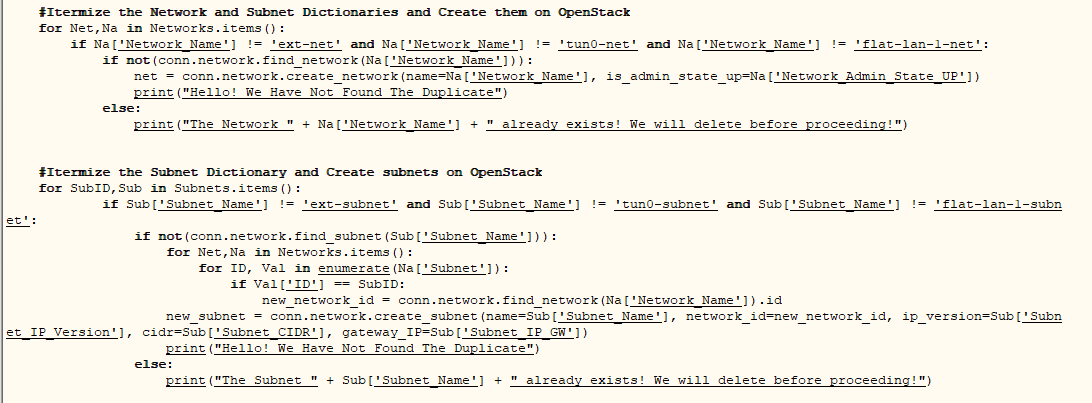


Figure 31: Code Snippet showing how networks and subnets are created on the OpenStack instance using attributes from their dictionary

It is worth to note that, since the OpenStack instance owns the default-created entities, including “ext-net” and “flat-lan-1-net”, we omit the creation of these two network and their corresponding subnets. In the current version, we start by creating the networks, followed by subnets, routers and instances.

Although the previously-saved IDs of the artifacts cannot be reused during the process of re-construction[[1]](#footnote-1), they still play an important role in this function. That is, when creating a new instance, we need to compare the old instance ID with the old device\_id attribute saved for the interfaces, to perform the needed correlation and guarantee the correctness of topology and dependencies in network artifacts.

***Note:*** *In our current implementation, we do not push/pull the VM images and their snapshots in this script due to the time constraint. However, we have successfully implemented this functionality and put them into the MasterScript.py file (Refer to upload\_new\_image, take\_VM\_snapshot* and *download\_new\_image). The following developer can simply call these functions and achieve the push/pull of saved snapshot images and run them in a new OpenStack instance.*

***Note:*** *The implementation of the interfaces in the recreation routine is done inside a nested call when the instances are created.*

***Note:*** *There is a known bug when recreating router interfaces by uploading a profile, their port status will automatically be set to “DOWN” instead of “ACTIVE”. There is currently no known way to change this. Therefore, we recommend that router interfaces be created manually. If you plan to recreate an experiment profile, document every network that each router is connected to so that this can be manually recreated later.*

## Managing a VM instance from the internet (**VMSSH.py**):

The objective of this script is to manage VM instances we have created from the control domain that is present on the public internet in our case, the Amazon EC2 instance.. We use Paramiko package[[2]](#footnote-2)[[3]](#footnote-3) of Python to accomplish this objective. First, we establish the first SSH connection from AWS instance to the CTL node in the OpenStack instance. Then, we establish the second SSH connection from the CTL node to the VM instance, using its private IP address. Once the second connection is established, the CTL sends the Shell commands to the VM instance. This nested implementation mimics the manual process we performed by connecting to the CTL and the VM instance sequentially.

As the future goal, we plan to let this function implement an interactive session, instead of throwing Shell commands to the VM. The future goal also include implementing a SOCKS proxy on the CTL, or simply configure the CTL as an SSH proxy server to forward the SSH requests that are addressing the internal instances present in the internal/private network.

## Library Program (**MasterScript.py)**

In addition to the scripts above, we also build a library script (MasterScript.py) of user-friendly functions that can be used with ease for the user wishing to interact with their OpenStack Instance. In the current version, MasterScript.py contains 17 distinct functions.

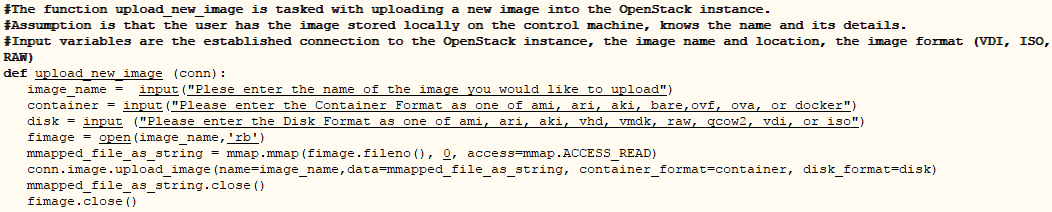


Figure 32: Code Snippet showing the function to upload a new image

For instance, the function “upload\_new\_image” shown above is to upload a VM image to the user’s OpenStack instance in CloudLab. It first takes a connection to the OpenStack instance as the input, and asks the user to enter the image parameters, such as the image location on disk, disk and/or container’s format. After that, it calls mmap[[4]](#footnote-4) to map the image file to the memory and uploads the images. Finally, it specifies the attributes (e.g., image\_name, flavor\_name, network\_name, instance\_name) of the image on the remote OpenStack instance.

Another critical function is the function “update\_hosts\_file”. It takes the IP address of the CTL node, and modifies the hosts file to include a new entry “CTL [CTL\_current\_IP\_address]”. Therefore, this function first allocates and deletes the existing entry of CTL in the hosts file. Then, it appends the IP address of the CTL into the hosts file. We do this because OpenStack APIs rely heavily on the JSON calls, which embed CTL in their URLs. Since every time we have a new OpensStack instance, the CTL could be a new one. We call this function to accommodate the newly assigned CTL host.

## Testing Program (TesterScript.py)

In the script, we first ask the user to input the IP address of the CTL before any function call is invoked. This allow the program calls the function update\_hosts\_file. This also ensures that our authentication using the Keystone (OpenStacks authentication client) service does not face any difficulties[[5]](#footnote-5), as it will also call stored procedures (JSON calls) using the CTL acronym.

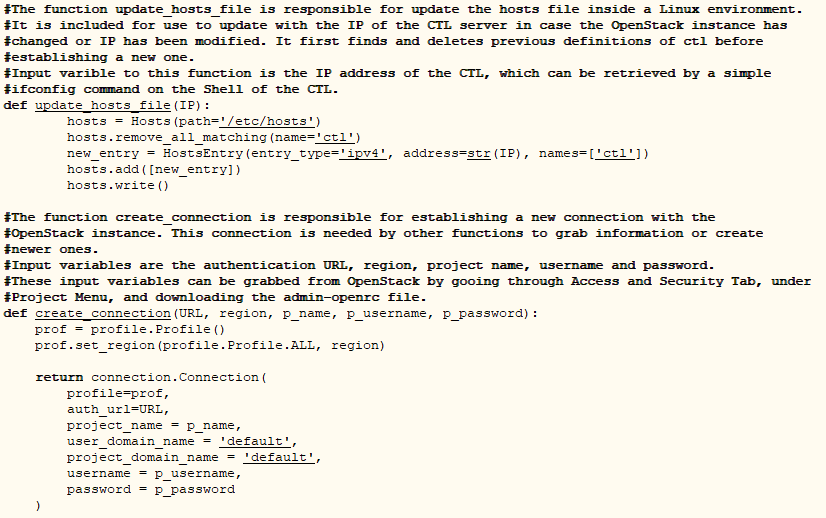


Figure 33: Code Snippet showing how hosts file is changed and how to create and authenticate an OpenStack connection

The figure above shows the “create\_connection” function. In the TesterScript file, after the user inputs the IP address of the CTL, they are asked for a host of information that are needed to create a successfully authenticated session with the OpenStack instance. This information can be found out from the “admin-openrc.sh” file, which is downloadable from OpenStack. It is crucial to extract the login parameters (e.g., URL, region, project name, project username,…) and input them “***as they are shown up in the admin-open file***”[[6]](#footnote-6); Otherwise the authentication to the CTL of the OpenStack instance will fail. The admin-openrc.sh file can be downloaded “Access and Security” 🡪 “API Access”, as shown below.

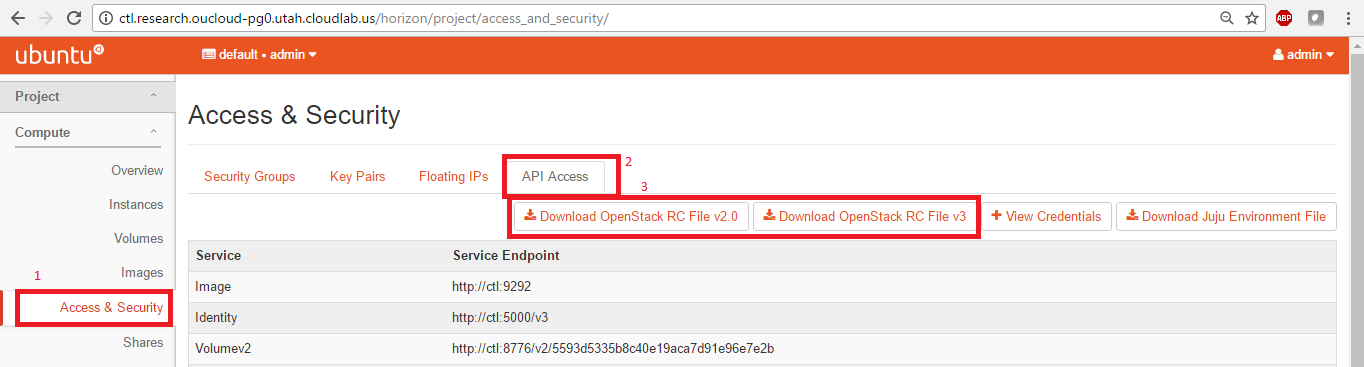


Figure 34: Location of the RC file to download from OpenStack GUI

***Note:*** *Please note that, the service endpoints are only accessible through the CTL node. That is why the* “update\_hosts\_file” function *modifies the* hosts *files to interact with the OpenStack instance.*

Running the TesterScript.py, as mentioned before, require that the user enters the IP of the CTL, and the detailed information for authentication. After that, a connection is established successfully, the user is presented with menu that lists the functions she can perform. Most of the functions can be found in MasterScript.py, as well as the ProfileScript.py and ReloadScript.py functions. The figure below shows how the user can enter the authentication details, as well as the host of several functions that are provided for the user to try out.

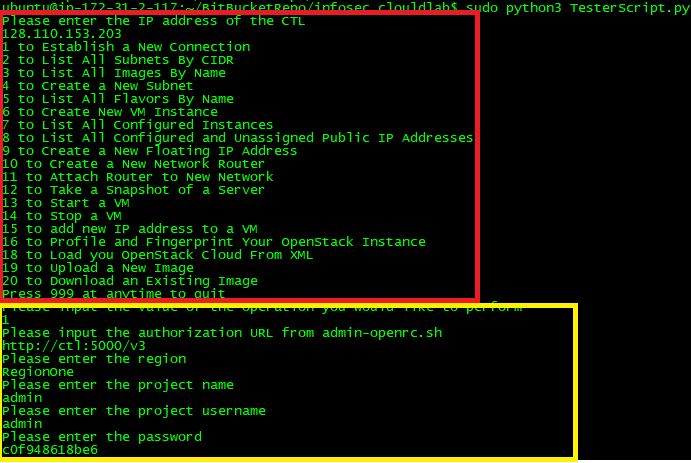
**

Figure 35: Sample of how to run the Tester Script, and the list of provided functionalities.

**The following steps show how to run this system:**

1. First, use our desktop SecureCRT to connect to AWS VM. Then, use AWS VM to control the machine CTL in the CloudLab. Since OU set the firewall to stop all the outgoing packets, we leverage AWS VM. This strategy should be a general step if other universities set up similar network security rules.
2. After login to AWS VM, install the required dependencies, then run a python3 program (.py) that creates the virtual lab in the CloudLab.

OpenStack

API (in .py)

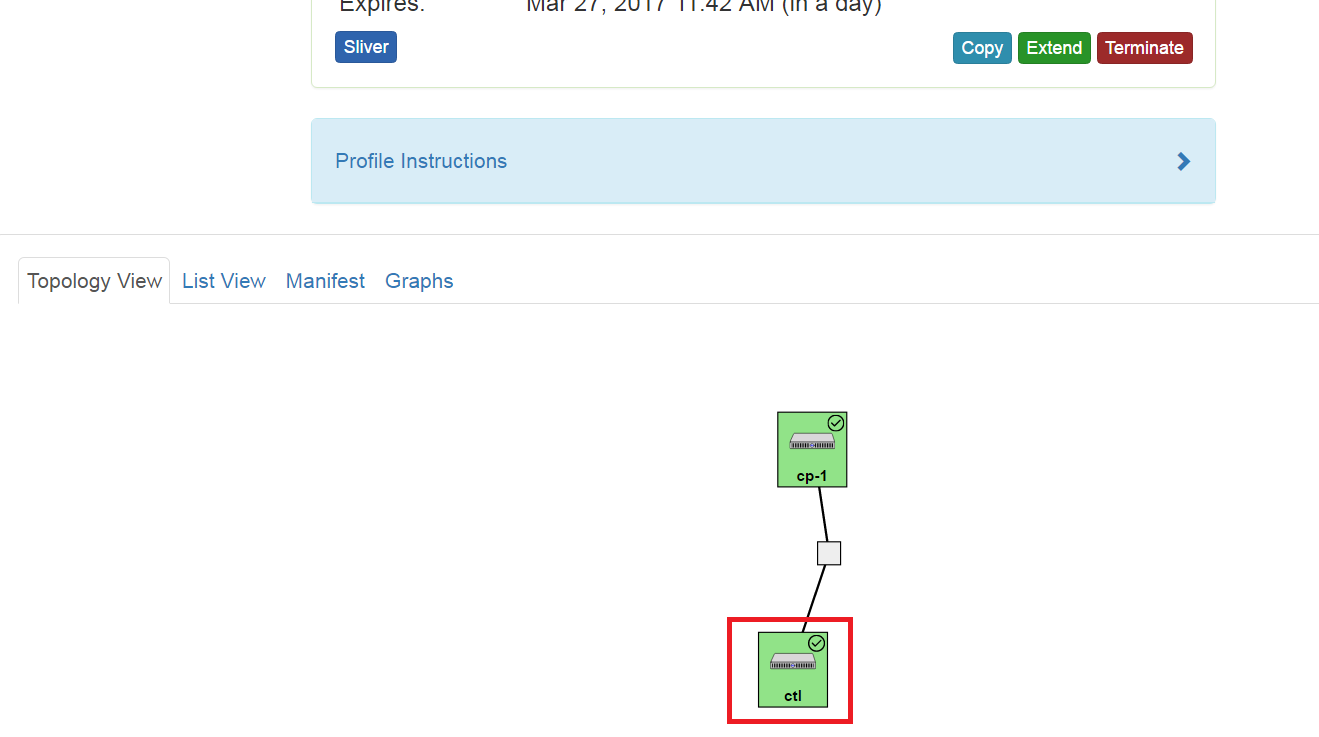
Our desktop

SecureCRT

SSH

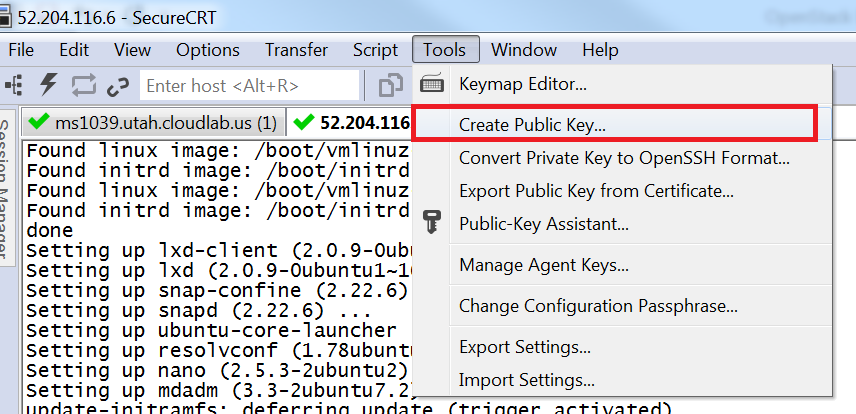
AWS VM

Our desktop

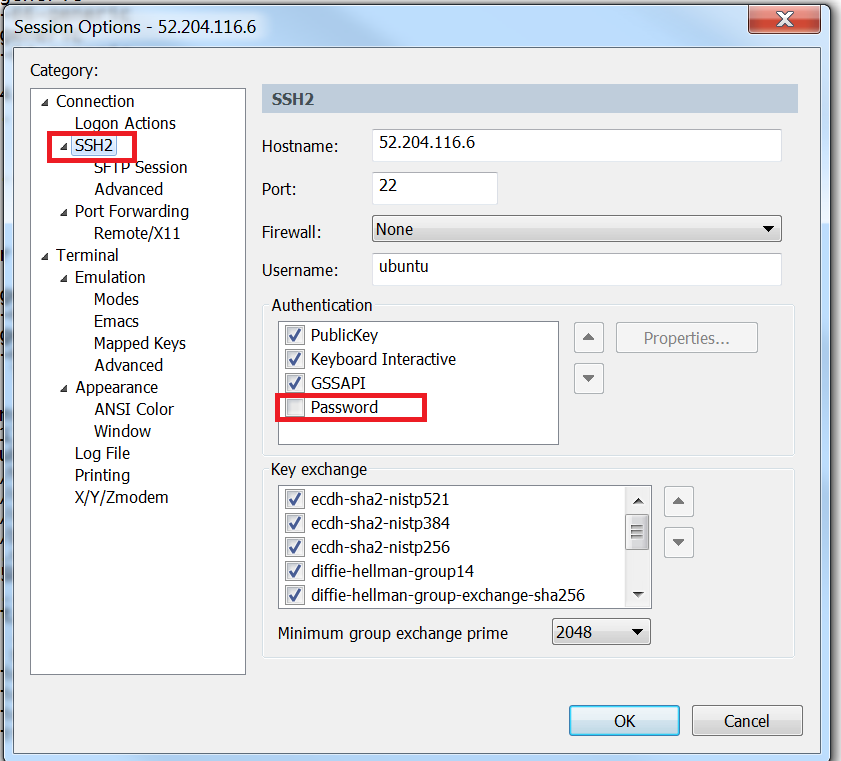


**Step1:** Use our desktop SecureCRT to connect to AWS VM. Then, use AWS VM to control the machine CTL in the CloudLab:

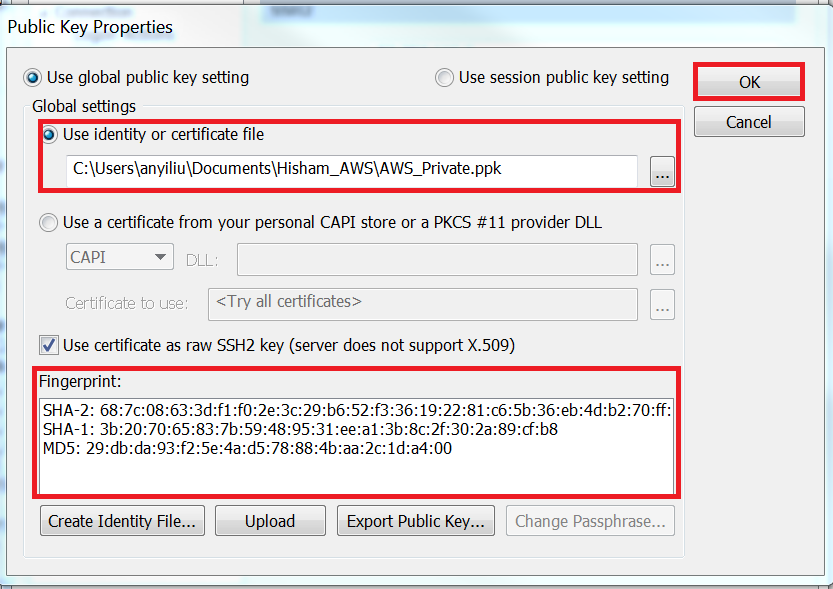
1.a: Generate a SSH Keypair on SecureCRT**:**



Follow the following instruction to generate the key pair:  
<https://www.vandyke.com/support/tips/publickeyauth.html>. Also, remember to **disable** the password authentication,



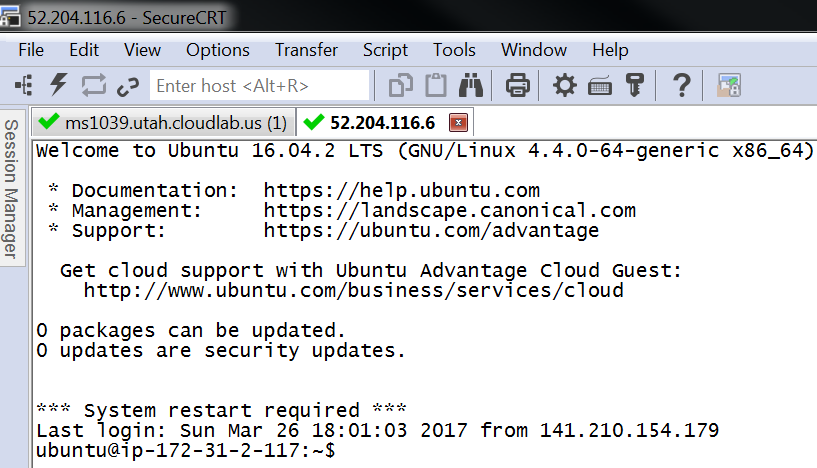
Then, click PublicKey 🡪 Property to pull out the following screen. Load the private key (the .ppk file). If it successfully loaded, you will see that Fingerprint of the key will be changed as well.



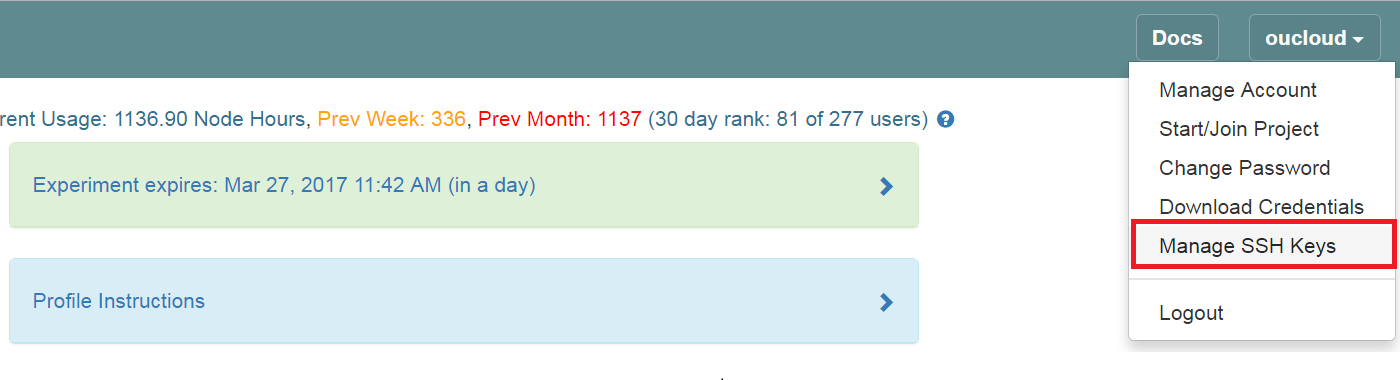
Why this is the private key?

**(How to upload the public key to AWS VM?)**

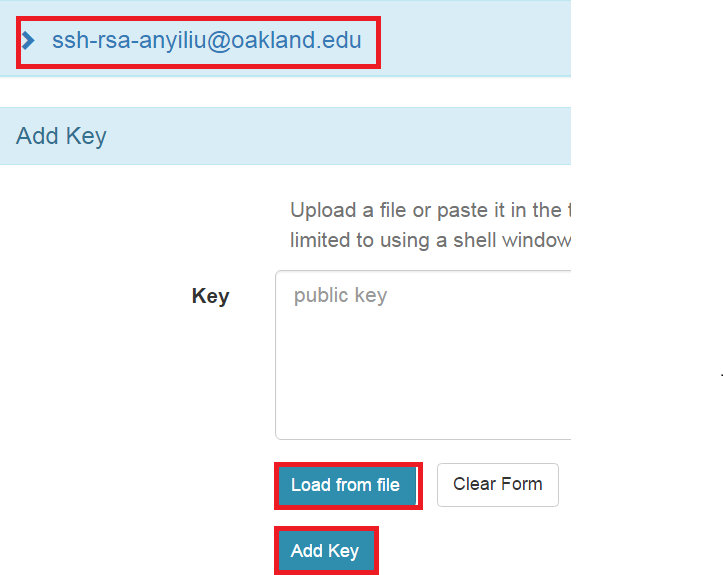
Once you’ve done the following setup, you should login AWS VM as follows:



You can do the similar like the above to login to the CTL machine on CloudLab (say, ms1039.utah.cloudlab.us). Click oucloud 🡪 Manage SSH Keys



Then, upload your public key as follows:



After that, setup the client (setup the private key) from the client as the above.

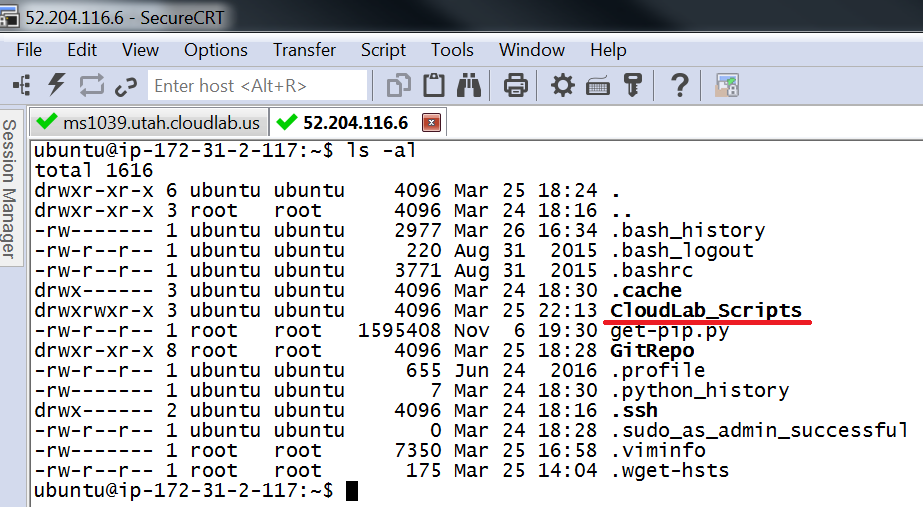
**Step2:** After login to AWS VM, run a python3 program (.py) that create the virtual lab in the CloudLab:

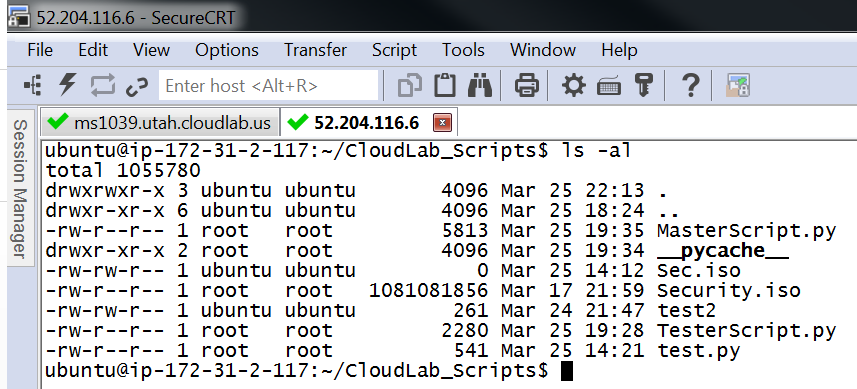
Before running these scripts, several dependencies are needed. You can run the following commands to install all the needed dependencies.

1. **sudo apt-get update**
2. **sudo apt-get install python3-pip**
3. **sudo pip3 install os\_client\_config**
4. **sudo pip3 install openstacksdk**
5. **sudo pip3 install python\_hosts**
6. **sudo pip3 install paramiko**
7. **sudo apt-get install python3-lxml**

You will then have to transfer the scripts to your AWS instance.

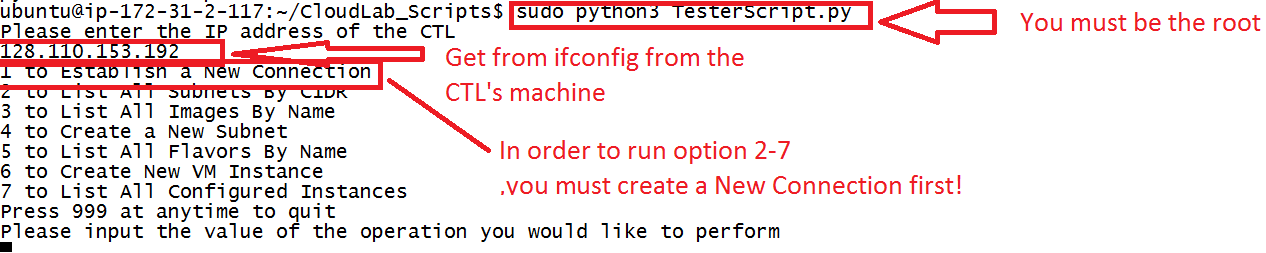
Say, you have the following login screen. Go to CloudLab\_Scripts



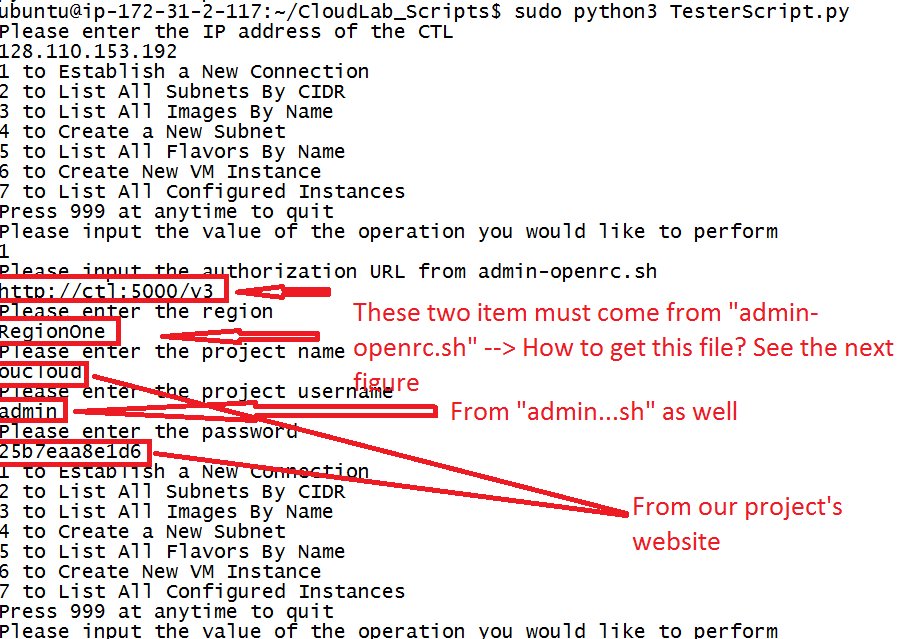


Pay attention to the following files and run TesterScript.py

1. MasterScript.py --- The program that contains all the single-out and generic functions
2. TesterScript.py --- The driver program

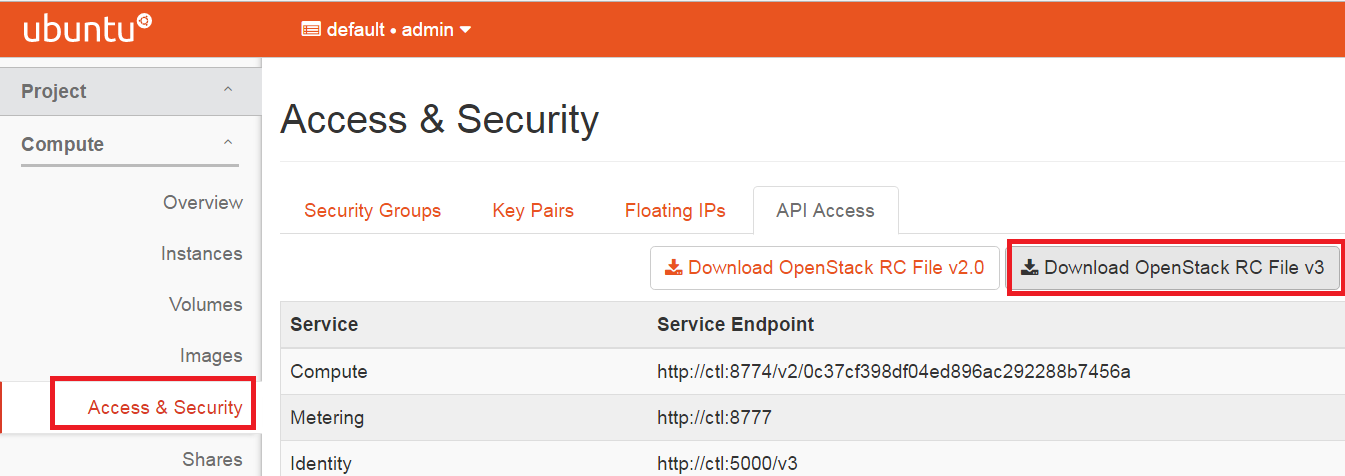


Choose option1, you will have the following:

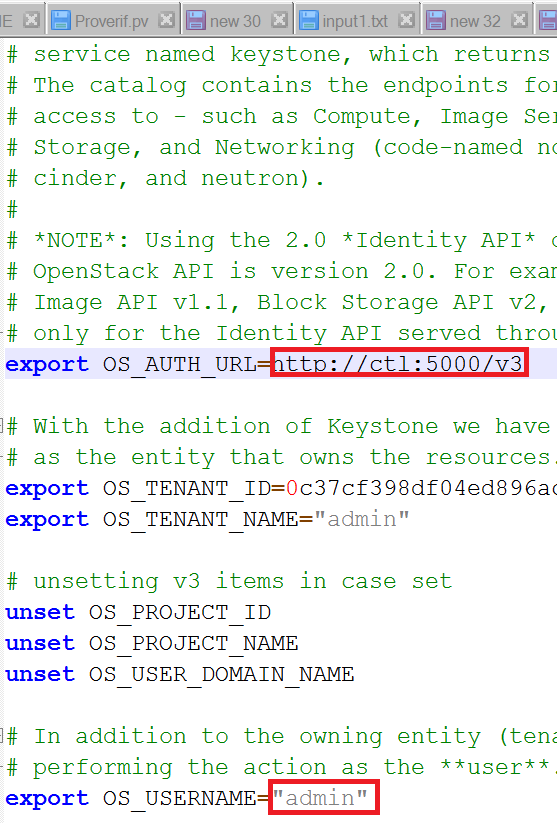


How to get “admin-openrc.sh”? See the following figures:

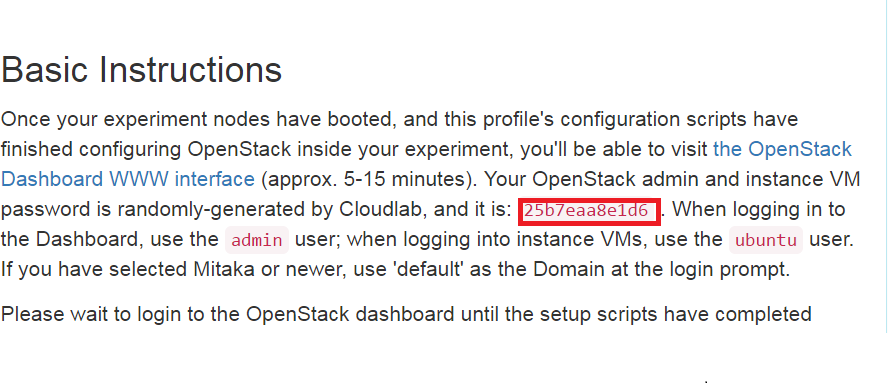
First, login to the Openstack Dashboard as follows. Click “Access and Security” 🡪 “Download OpenStack RC File v3”. You should be able to download \*rc.sh from there.



In that file, you should see the following and get the “project URL” and “project username” from this file.



As far as the password, you can get it from the project’s webpage.



The information extracted from above mini-steps can be put into on .txt file as the following. It can be copied and pasted into the Python program while running.

128.110.153.205

1

http://ctl:5000/v3

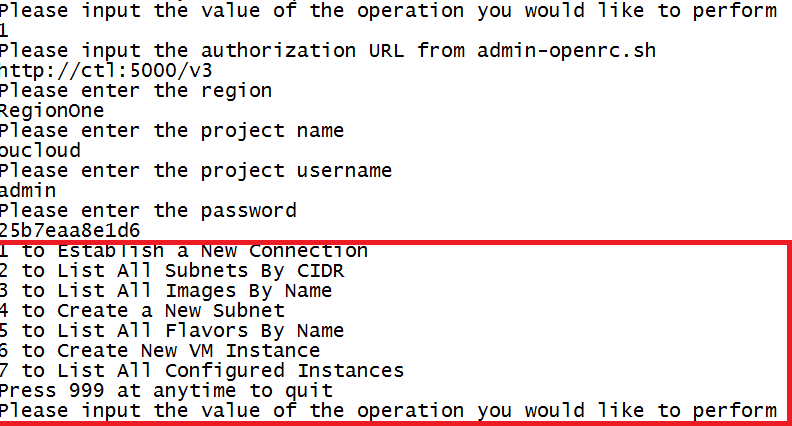
RegionOne

admin

admin

c0f948618be6

If you can see the following figure without any complaint/error/warning, you should be OK choose option 2-7.



# Future Works

Our current project only provides the basic proof-of-the-concept implementation. As the future plan, we will do the following:

* We will implement a user-friendly GUI/web interface that consolidates generic (fine-grained) functions into more coarse-grained functions. For example, we can provide a simpler 1st level menu and a more detailed 2nd level menu.
* We will leverage some APIs of OpenStack to define different groups/domains that conform to certain security policies.
* We will implement a Proxy/SOCKS that allows students to SSH into the VM instances (with the internal IP) in the OpenStack instance launched in CloudLab.
* Currently, the reconstruction process is *linear*, which does not support any parallel processing. We will resolve the dependency problem by using techniques such as topology sorting[[7]](#footnote-7)
* We plan to study the feasibility of modifying the authentication URLs that are built by default by OpenStack to push more secure services, such as HTTPS for the authentication, to replace the default HTTP URL.

# Conclusions

This report is a basic summary and user guide with interpretation of the code that we have developed. As we have previously stated, there has not been a real effort that can build on the powers and robustness of OpenStack to achieve a high programmability of the networks. In addition, there has been no effort publicly shared, that would discuss the needs and advantages for fully automating the processes of backing up and creating the networks from the start. This is extremely handy for the reproducing of results in the area of empirical research. In addition, it can be used powerfully as a backup and recovery mechanism for corporations that deploy OpenStack, with its wide current adoption in the market. Other critical use-cases of our effort is the field of education, enabling higher education organizations to encourage hands-on work with the ability of providing complex architectures that the students can import and use with ease.

Added to the above is the fact that our backup is produced in the XML format, which makes it easier to use such files in other third party applications. Adhering to the standards used by the W3C for information sharing using XML stresses the versatility of our architecture.

# Acknowledgements

We would like to express our deep gratitude to Dr. Anyi Liu for his direct supervision and oversight in this project, as well as his invaluable insight during the process of the development of the code.

# References

1. Cloudlab Basic Concepts: <http://docs.cloudlab.us/basic-concepts.html>
2. Cloudlab: <http://cloudlab.us>
3. OpenStack: <https://www.openstack.org/software>
4. OpenStackSDK: <https://docs.openstack.org/user-guide/sdk.html>
5. Python IP Address Package: <https://docs.python.org/3/library/ipaddress.html>
6. Python Paramiko Package: <http://www.paramiko.org/>

# Amendment from the REU 2017 Team

The following sections contain information about the contributions by the REU 2017 team. These sections will cover the added functionality to the scripts written by the previous REU team, the attack scenario experiment implementation, documentation written for experiment implementation, and the web client developed for use with experiments. For the sake of brevity and to avoid reiterating previous information, the explanation for each portion of this section will be shortened. The full documentation for instructors and students can be found in Appendix H. Our updated MasterScript and Testerscript Python scripts can be found in Appendix F and Appendix G.

# Added Functionality as of July 17, 2017

Option 21

Option 20

Option 18

Option 7

Option 1

write\_to\_sql

create\_multiple\_networks

create\_from\_xml

list\_all\_instances

list\_all\_images

Figure 36: Updated list of options

1. Added function to list images when connection is successful. Beforehand, there would be no indication that a connection was successful. **(Melissa)**
2. Added function that saves IP addresses of configured instances to a text file, which can then be uploaded into our SQL database. **(Nathan)**
3. Fixed recreation of router interfaces. Previously, the code would upload everything except router interfaces. With Hisham’s help, we were able to fix this issue. **(Melissa)**
4. Added specific function that creates networks containing Attack and Victim images for our attack scenario experiment. This function is multithreaded, so the creation process does not take long**. (Nathan)**
5. Added function that takes IP addresses for an experiment and pushes them to a SQL database, which is then used with our web client. Option 7 provides the file that Option 21 pulls IP addresses from. **(Nathan)**

# Using Virtual Laboratory to Run Attack Scenario Experiment

By building upon the existing framework, we have designed an example experiment that can be recreated by instructors and students. The purpose of this experiment is to simulate an attacking scenario on a virtual network. Two virtual images are needed: one for the attacker, one for the victim. The attack image is a Ubuntu image with Metasploit installed. The victim image is Metasploitable, a purposefully vulnerable Linux image. We then push these images to CloudLab, and use our updated scripts to create the network structure. We also use our scripts to create a database of IP addresses that can be accessed using an in-browser web client and SSH terminal, eliminating the use of external SSH clients. The basic outline of running the experiment is as follows:

1. Creating a new CloudLab experiment with 16 floating IP addresses assigned
2. Uploading preconstructed Attack and Victim virtual machines
3. Using scripts to create web client framework
4. Using our scripts to create network structure – each network “bubble” contains one Victim and one Attacker
5. Manually associating floating IPs with Attacker images
6. Using our scripts to push IP addresses to web client database
7. Running attack scenario experiment with students using web client

# Creation of Laboratory Environment (For Instructors)

Various portions of this creation process have already been explained previously in this documentation. Therefore, only the parts that are relevant to the attack scenario experiment are included. Instructions on how to complete certain tasks, such as attach a network to a router, can be found in its proper section in this documentation.

1. It is very important that 16 public IP addresses as associated with an experiment during creation. This maximizes the amount of virtual machines that can be assigned to students.

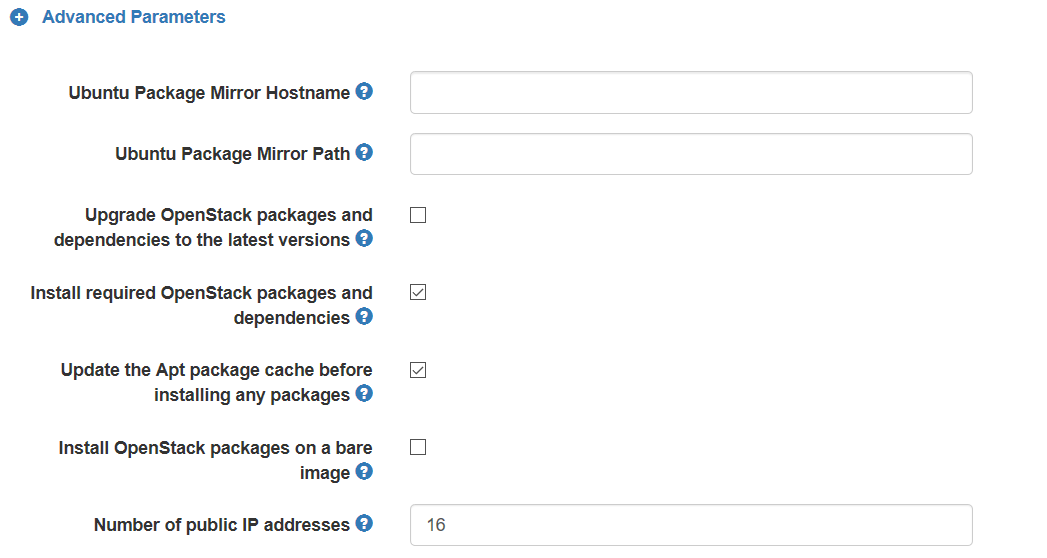


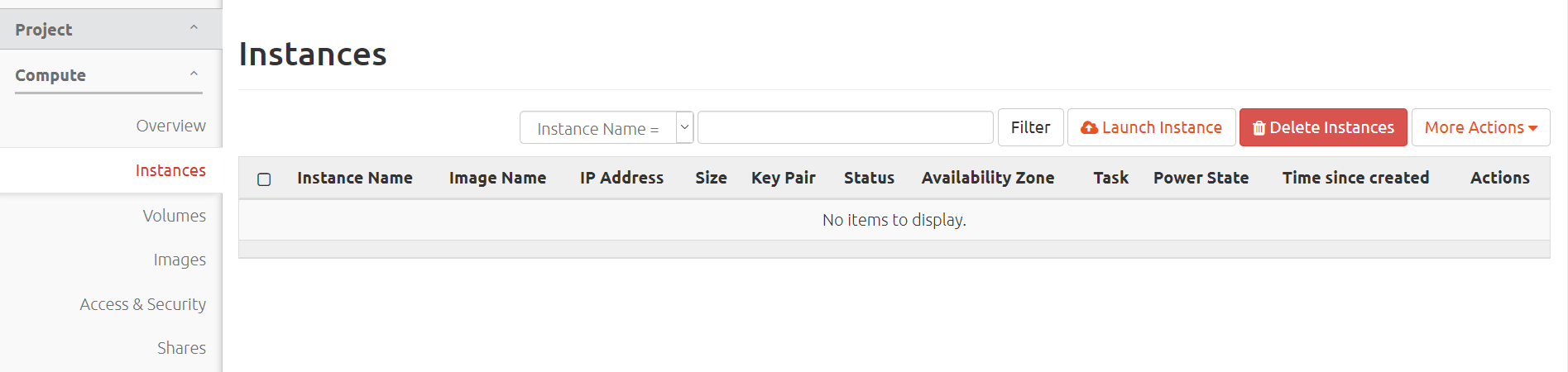
Figure 37: Showing number of public IP addresses

1. For our attack scenario experiment, we choose to use the Utah cluster. After creation of the experiment, two virtual images need to be uploaded. We have uploaded these images to a downloadable archive, which then can be pushed to the OpenStack instance either through direct link or from ones computer. The images can be found at the following links: **[https://archive.org/download/Metasploitable/Metasploitable.vmdk](https://archive.org/download/Metasploitable/Metasploitable.vmdk" \t "_blank)**

**[https://archive.org/download/Metasploit2/Metasploit2](https://archive.org/download/Metasploit2/Metasploit2" \t "_blank)**

Instructions to upload a new image are as follows:

* + - 1. The first page that will open is the list of instances. Because this experiment is newly created, there will be no instances listed here. On the left side, click *“Images*”.
      2. On the Images page, click *“Create Image”.*
      3. We will upload the Attack image first since it is the largest. For the name, type *“Attack”* without quotations. **(It is very important you give it this exact name, or else the experiment will not upload properly in later steps.)** For image location, copy and paste this address: **[https://archive.org/download/Metasploit2/Metasploit2](https://archive.org/download/Metasploit2/Metasploit2" \t "_blank)** For image format, select “Raw”.
      4. Once you have confirmed these settings are correct, scroll down and click*“Create Image”.*
      5. Now we will import the Victim image. Click *“Create Image”* once more.
      6. For the name, type *“Victim”* without the quotations. **(It is very important you give it this exact name, or else the experiment will not upload properly in later steps.)** For image location, copy and paste this address: **[https://archive.org/download/Metasploitable/Metasploitable.vmdk](https://archive.org/download/Metasploitable/Metasploitable.vmdk" \t "_blank)** Confirm that VMDK is the image format selected.
      7. Once you have confirmed these settings are correct, scroll down and click *“Create Image”.*
      8. It will take a while for these images to upload. When the image status says *“Saving”*, the image has not finished uploading. When the image status says *“Active”,* the image is finished uploading.

Figure 38: Where to find the “Images” page

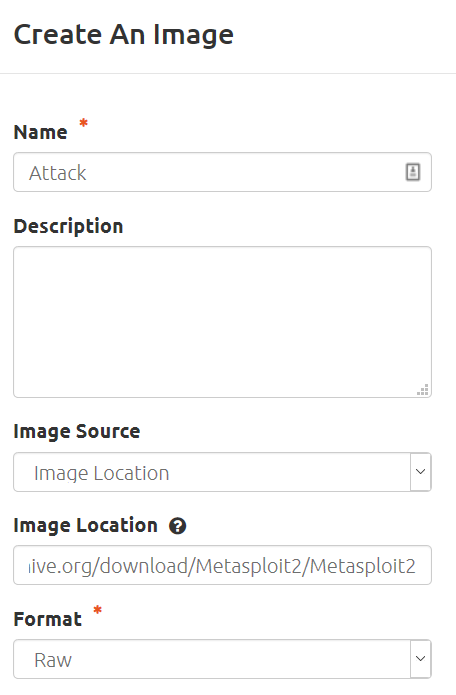
**

Figure 39: Image settings for Attack image

1. We chose to use MobaXterm as our SSH client to connect to an Amazon EC2 instance. We install the needed dependencies and transfer our scripts to this EC2 instance before proceeding. MobaXterm allows file transfer without an external client.
2. We run our TesterScript as shown previously in this report, and run **Option 20** to create our networks. What this Option does is create network “bubbles” to the users specification, with each network containing one Attack and one Victim instance. Once this has finished running, this function also automatically allocates **14** floating IP addresses to the experiment. The attack instance contains a shell script which allows a process to run at boot time. This process starts Run Wetty which enables the student to connect to the instance via the browser SSH client.

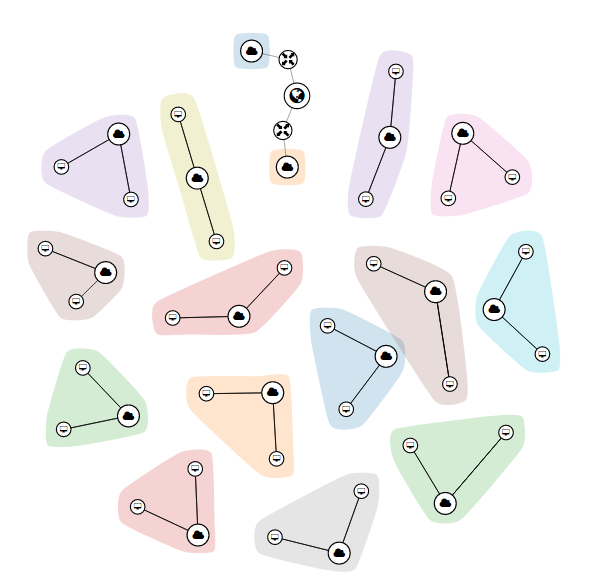
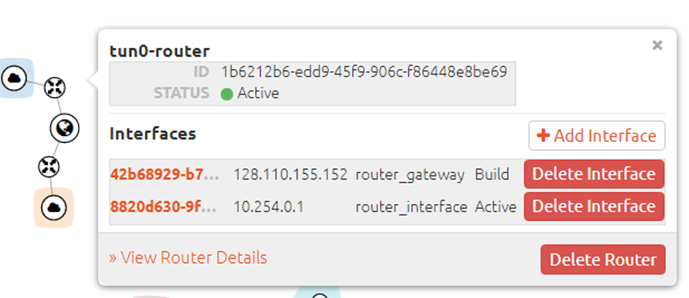
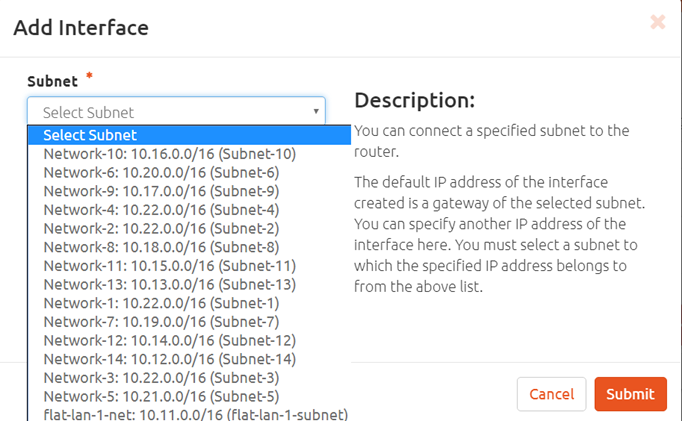


Figure 40: Decentralized networks created using Option 20

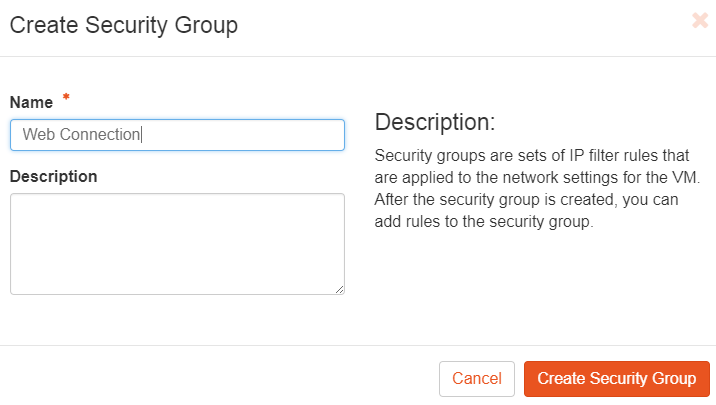
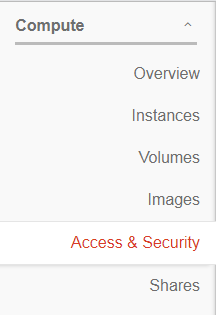
1. Because of the router interface “DOWN” status error, each network must be manually connected to one of the two default routers. One of the default routers can be discarded if needed, freeing up one more public IP address. The process of manually connecting each network to the router does not take more than a few minutes.





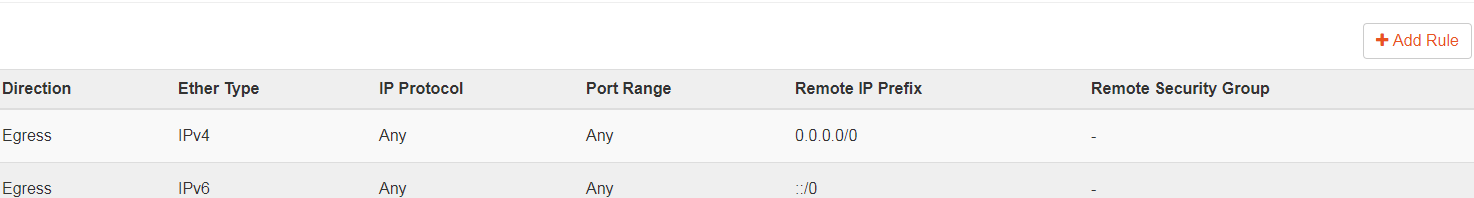
*Figure 40: Manually adding a router interface*

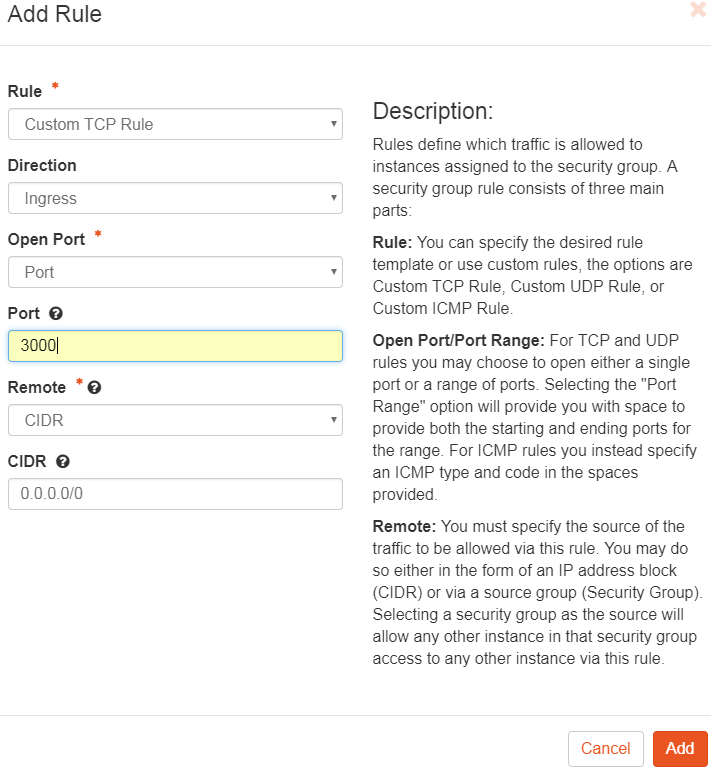
1. Each allocated floating IP must be manually associated to each Attack instance, but only the Attack instances. Once again, this process does not take more than a few minutes.
2. After a floating IP address is associated to each attack instance, a set of security rules must be established so that the Attack image may be accessible to the students.
3. RunWetty utilizes TCP port 3000 to establish an in-browser SSH terminal. Create a security group which allows this port. Figure 41 details how to create a security group.



*Figure 41: Creating a security group*

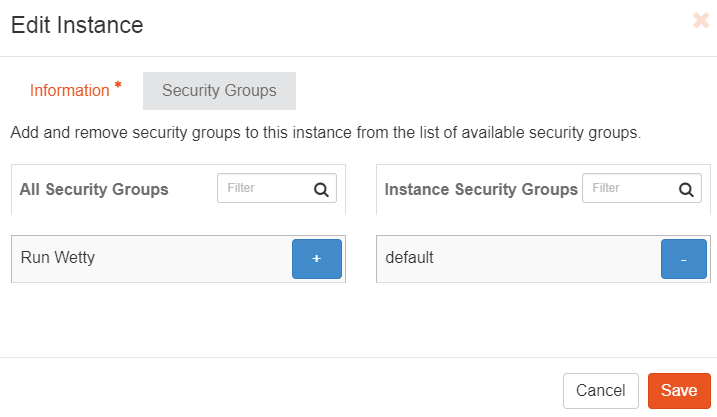
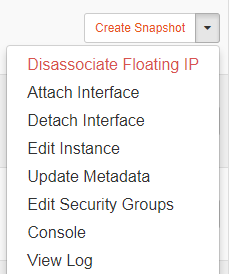
1. After the security group is created, a security rule must be added. This security rule will allow traffic from port 3000.





*Figure 42: Adding a rule to a security group*

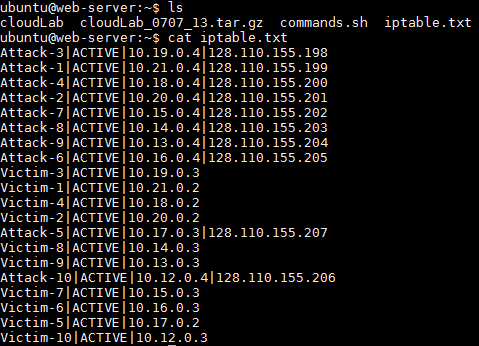
1. The rule must now be added to each Attack instance, figure 43 explains the process.



*Figure 43: Adding a security group to a virtual machine*

1. The next steps are to set up the web server.
2. Use the following link to upload the Web Server to an experiment: **<https://archive.org/download/WebServer/Web%20Server>**
3. When the image is uploaded, create an instance from it and allocate a floating IP address. Connect to this instance using “Moba X-term” or any SSH client. When prompted for a password, please enter the following: **“7770ddc5d198”**. This image contains both the Master Script and Tester Script to allow you to connect to the control node of the experiment as well as a MySQL database which will store the IP addresses from the experiment.
4. The web server will need to be configure with a public ip address. You will use this public IP address to connect to the server in an SSH client and access the database.
5. The IP addresses of the attack and victim nodes must be obtained so that the Web Server can access them and allow the students to connect to them. Run the TesterScript and connect to the control node where the attack and victim machines are located. When the connection is established, select option 7. Option 7 will obtain the IP addresses of the attack and victim machines and store them into a file.

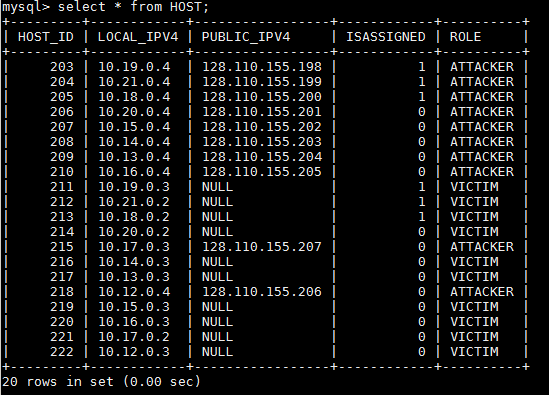




*Figure 44: Selecting option 7 and the output of the file that stores the IP addresses*

1. Next, select option 21. Option 21 takes the values in the file and uploads them to the SQL database.



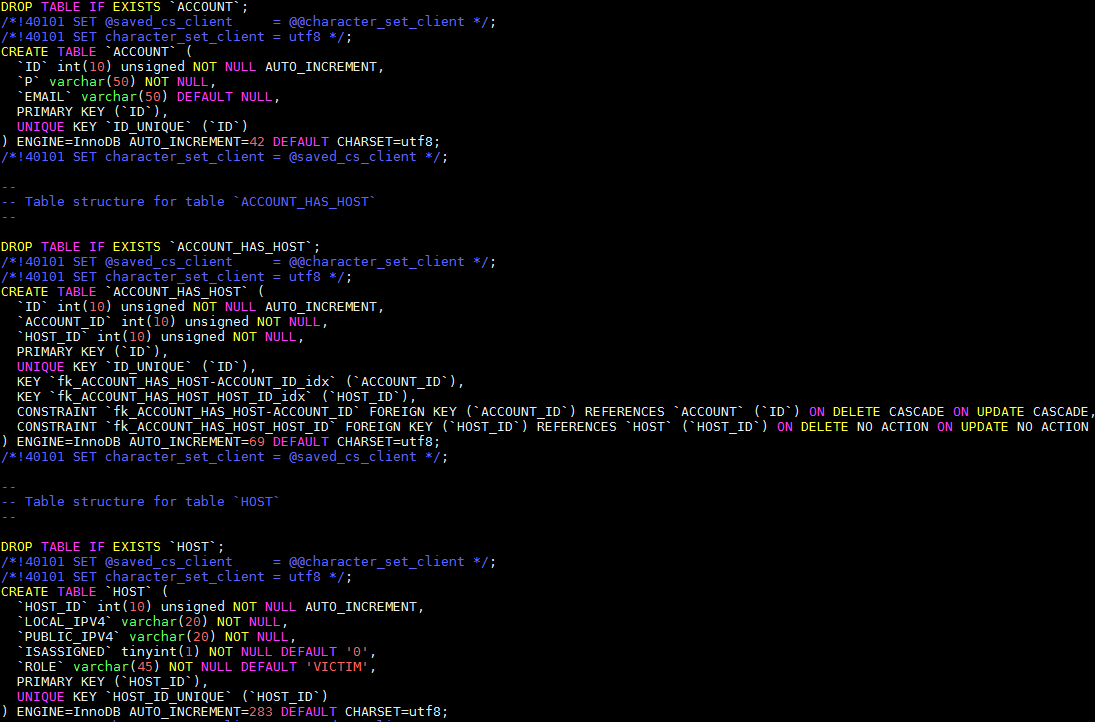


*Figure 45: Selecting option 21 and The IP addresses stored in the database*

1. You can export the database schema and output the schema into a file. This is done by the following command: **mysqldump -u root -p --no-data CLOUDLAB > schema.sql**
2. You can view the schema using a text editor or a concatenation.

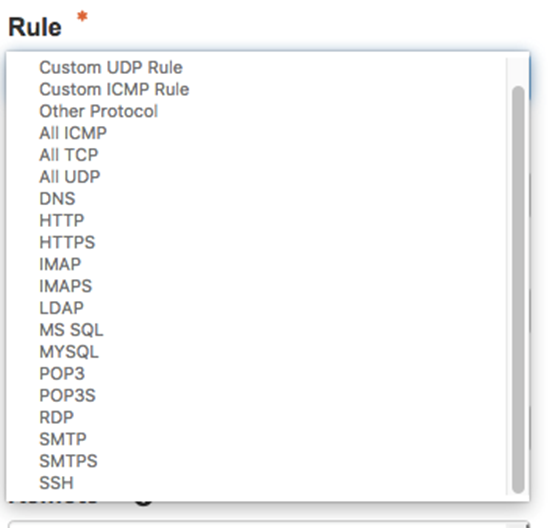
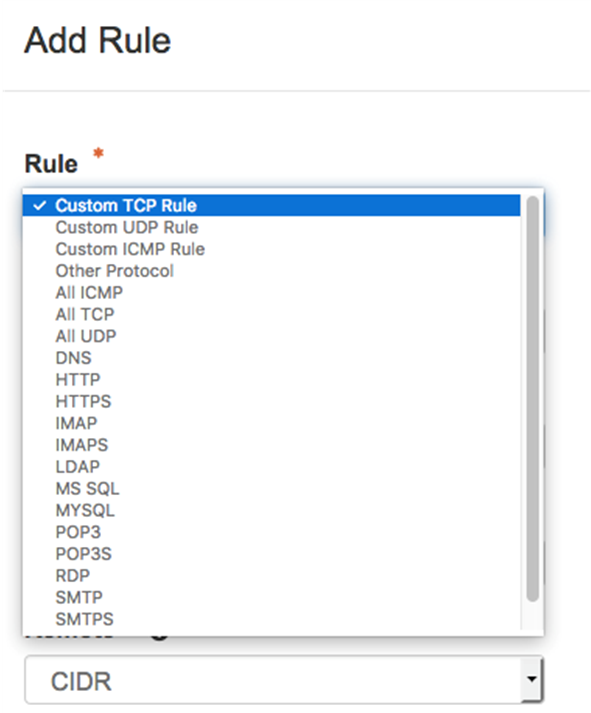


*Figure 46a: Command to export the database schema*

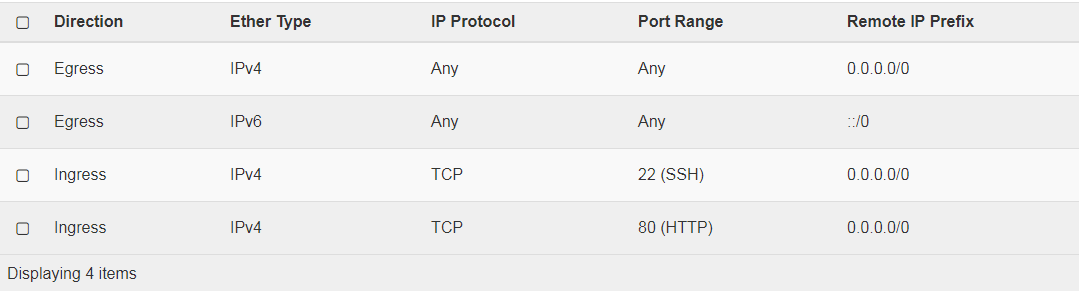


*Figure 46b: The database schema*

1. After the values are placed in the database, a new security rule needs to be added to the web server. Please reference figures 41-43 as a guide. The only difference is that the web server will utilize port 80 and port 22 as opposed to port 3000. Please make the necessary adjustment.



*Figure 47: Configuring the proxy server with the correct rules*



*Figure 48: Table containing the rules for the web server*

1. After the rule is added, the web server is ready for the student to use. To reset the database after students are done, execute the Tester Script, connect to the control node for the experiment, and run option 21. This will clear and repopulate the database with the corresponding IP addresses from the file

# Running the Virtual Laboratory Environment Using Web Client (For Students)

The following steps outline how a student would connect to the web client, and an example of an exploit that would be run.

1. The IP address to access the web client varies from experiment to experiment. This IP address should be given to students beforehand.

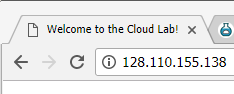


Figure 48: IP address to access the web client for one experiment (This will not be the same for every experiment.)

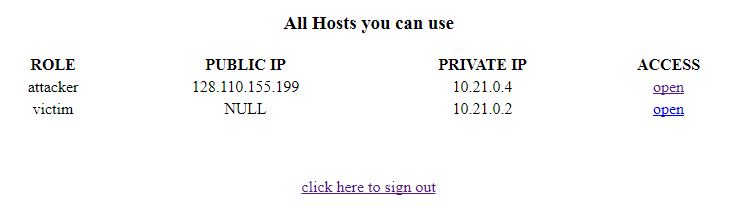
1. When accessing this IP address, a page will be shown with a Login and a Sign Up option. Each student must first sign up before they can log in. After signing up, the student will be redirected to a login page where they can enter their credentials.

Figure 49: Attack and Victim IP’s that the Student will see

1. The page will redirect to a page showing two virtual machines: the attacker and the victim. The victim machine cannot be accessed via the web client, but it not necessary for this experiment. By clicking on **“open”** next to the attacker virtual machine, a new tab will be opened with an in-browser SSH terminal.
2. The student will then be prompted for a username and password. For this image, the username is **ubuntu** and the password is **ce17e6285b09**.
3. After logging into the virtual machine, the **postgresql** service must be started. Typing **sudo service postgresql start** will start this service.
4. The Metasploit console can be launched by typing **sudo msfconsole.**
5. The following steps outline an example of one exploit that can be ran by students:
   1. Type **use exploit/unix/irc/unreal\_ircd\_3281\_backdoor into the Metasploit console and press enter.**
   2. **Find the IP address for your victim machine. This should be present in the web client.**
   3. **Type set RHOST followed by the IP address of your victim machine, as seen in the figure below. Press enter.**
   4. **Type exploit and press enter.**
   5. To verify the exploit was successful, you will run some system identification commands. Type **uname -a.** You should see that **Metasploitable** will be the image name shown. Then, type **ifconfig.**You should see the IP address for the Victim machine shown next to **inet addr.** See the figure below.
   6. When you are ready to quit, press **CTRL + C** and enter **y** to exit and enter back into the Metasploit console.
   7. Type **back** and press enter to quit using the current exploit.

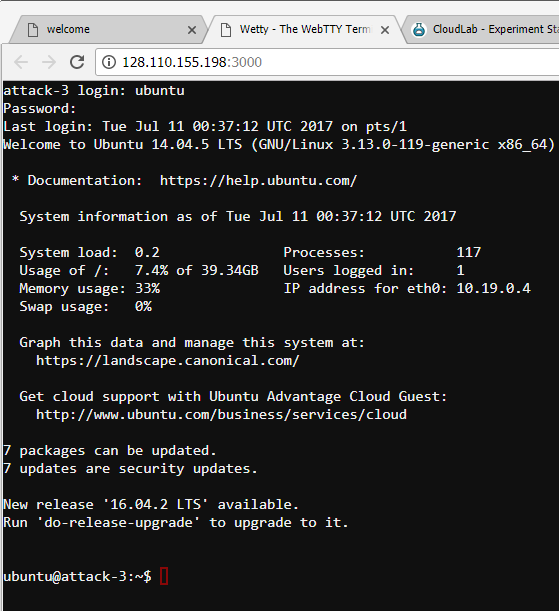


Figure 50: Screenshot showing in-browser SSH client

# Appendix A: ProfileScript.py

import sys

import hashlib

import requests

import ipaddress

import os

import os\_client\_config

import time

import datetime

import xml.etree.ElementTree as ET

import mmap

from lxml import objectify,etree

from openstack import connection

from openstack import profile

from openstack import utils

from python\_hosts import Hosts, HostsEntry

from xml.etree import ElementTree as etree

from xml.dom import minidom

#The function Profile\_OpenStack takes as input the connection to the OpenStack Instance

#The functions calls for the lists of networks, subnets, flavours, routers, and instances and parses the output

#The function then creates the corresponding tree structure at every iterate, taking the ID as the appropriate differentiating attribute.

#The function saves the output with a Timestamped name, and then calls Prettify function to neatly display the document in a human-readable form.

def Profile\_OpenStack(conn):

Profile = etree.Element('Profile')

Interfaces = etree.SubElement(Profile, 'Interfaces')

for interfaces in conn.network.ports():

Interface\_ID = etree.SubElement(Interfaces,'Interface')

Interface\_ID.attrib['ID'] = interfaces.id

Interface\_Name = etree.SubElement(Interface\_ID,'Interface\_Name')

Interface\_Name.text = interfaces.name

Interface\_Net = etree.SubElement(Interface\_ID,'Network\_ID')

Interface\_Net.text = interfaces.network\_id

Interface\_State = etree.SubElement(Interface\_ID,'Interface\_Admin\_state\_UP')

Interface\_State.text = str(interfaces.is\_admin\_state\_up)

Interface\_Device = etree.SubElement(Interface\_ID,'Connected\_Device\_ID')

Interface\_Device.text = interfaces.device\_id

**#Since Fixed\_IPS is a nested dictionary within the class, we need to enumerate its elements to access the keys, get the corresponding values**

for i, entry in enumerate(interfaces.fixed\_ips):

Interface\_Sub = etree.SubElement(Interface\_ID,'InterfaceSubnet')

Interface\_Sub.attrib['ID'] = entry['subnet\_id']

Interface\_IP = etree.SubElement(Interface\_Sub,'InterfaceIP')

Interface\_IP.text = entry['ip\_address']

**#Create new SubElement under Profile, declaring the start of the networks portion of the profile**

Networks = etree.SubElement(Profile, 'Networks')

**#Iterate over the list of networks to get corresponding attributes of each**

for networks in conn.network.networks():

Network\_ID = etree.SubElement(Networks,'Network')

Network\_ID.attrib['ID'] = networks.id

Network\_Name = etree.SubElement(Network\_ID,'Network\_Name')

Network\_Name.text = networks.name

Network\_State = etree.SubElement(Network\_ID,'Network\_Admin\_State\_UP')

Network\_State.text = str(networks.is\_admin\_state\_up)

**#Iterate over list of subnets. Get corresponding attributes and save them in dictionary, index by the ID as key. This preserves the Subnet-Network dependency**

for subnets in conn.network.subnets():

**#Preseve Subnet-Network dependency by associating the Subnet and network IDs in the dictionary.**

if subnets.network\_id == networks.id:

Subnet\_Detail\_ID = etree.SubElement(Network\_ID,'Subnet')

Subnet\_Detail\_ID.attrib['ID'] = subnets.id

Subnet\_Details = etree.SubElement(Subnet\_Detail\_ID,'Subnet\_Name')

Subnet\_Details.text = subnets.name

Subnet\_Detail\_CIDR = etree.SubElement(Subnet\_Detail\_ID,'Subnet\_CIDR')

Subnet\_Detail\_CIDR.text = subnets.cidr

Subnet\_Detail\_IP\_Ver = etree.SubElement(Subnet\_Detail\_ID,'Subnet\_IP\_Version')

Subnet\_Detail\_IP\_Ver.text = str(subnets.ip\_version)

Subnet\_Detail\_IP\_GW = etree.SubElement(Subnet\_Detail\_ID,'Subnet\_IP\_GW')

Subnet\_Detail\_IP\_GW.text = str(subnets.gateway\_ip)

**#Create new SubElement under Profile, declaring the start of the routers portion of the profile**

Routers = etree.SubElement(Profile,'Routers')

**# Iterate over list of routers. Get corresponding attributes and save them in dictionary, index by the ID as key.**

for routers in conn.network.routers():

Router\_ID = etree.SubElement(Routers, 'Router')

Router\_ID.attrib['ID'] = routers.id

Router\_Name = etree.SubElement(Router\_ID,'Router\_Name')

Router\_Name.text = routers.name

Router\_Status = etree.SubElement(Router\_ID,'Router\_Admin\_State\_UP')

Router\_Status.text = str(routers.is\_admin\_state\_up)

**#Routers may reside in different availability Zones. Save Each.**

for AZ in routers.availability\_zones:

Router\_AvailabilityZone = etree.SubElement(Router\_ID,'Zone')

Router\_AvailabilityZone.text = AZ

**#Iterate over the list of interfaces with device\_id matching the current router\_id. This preservers data dependency.**

for ports in conn.network.ports():

if ports.device\_id == routers.id:

Router\_Interface = etree.SubElement(Router\_ID,'Interface')

Router\_Interface.attrib['ID'] = ports.id

Router\_IP = etree.SubElement(Router\_Interface,'IP')

Router\_IP.text = ports.fixed\_ips[0]['ip\_address']

**#Create new SubElement under Profile, declaring the start of the images portion of the profile**

Images = etree.SubElement(Profile,'Images')

**#Iterate over list of images. Get corresponding attributes and save them in dictionary, index by the ID as key.**

for images in conn.image.images():

Image\_Name = etree.SubElement(Images,'Image')

Image\_Name.attrib['Name'] = images.name

Image\_Container\_Format = etree.SubElement(Image\_Name,'ContainerFormat')

Image\_Container\_Format.text = images.container\_format

Image\_Disk\_Format = etree.SubElement(Image\_Name,'DiskFormat')

Image\_Disk\_Format.text = images.disk\_format

**#Create new SubElement under Profile, declaring the start of the Flavorss portion of the profile**

Flavors = etree.SubElement(Profile,'Flavors')

for flavors in conn.compute.flavors():

Flavor\_Name = etree.SubElement(Flavors,'Flavor')

Flavor\_Name.attrib['Name'] = flavors.name

Flavor\_VCPU = etree.SubElement(Flavor\_Name,'VCPU')

Flavor\_VCPU.text = str(flavors.vcpus)

Flavor\_Disk = etree.SubElement(Flavor\_Name,'Disk')

Flavor\_Disk.text = str(flavors.disk)

Flavor\_Ram = etree.SubElement(Flavor\_Name,'RAM')

Flavor\_Ram.text = str(flavors.ram)

**#Create new SubElement under Profile, declaring the start of the instances portion of the profile. Iterate over each instance and get corresponding attributes.**

Instances = etree.SubElement(Profile,'Instances')

for instances in conn.compute.servers():

Instances\_ID = etree.SubElement(Instances,'Instance')

Instances\_ID.attrib['ID'] = instances.id

Instances\_Name = etree.SubElement(Instances\_ID,'Name')

Instances\_Name.text = instances.name

Instances\_Status = etree.SubElement(Instances\_ID,'Status')

Instances\_Status.text = instances.status

**#Get List of flavors, compare links to the flavor link variable in the instance to get the flavor ID and name.**

for flavor in conn.compute.flavors():

if flavor.links[1]['href'] == instances.flavor['links'][0]['href']:

Instances\_Flavor = etree.SubElement(Instances\_ID,'Flavor')

Instances\_Flavor.text = flavor.name

**# Get List of images, compare links to the image link variable in the instance to get the image ID and name.**

for image in conn.image.images():

if image.id == instances.image['id']:

Instances\_Image = etree.SubElement(Instances\_ID,'Image')

Instances\_Image.text = image.name

**#Get list of ports/interfaces, compare their IDs to with the interface ID connected to this instance. Get the corresponding interfaceID and IPAddress**

for ports in conn.network.ports():

if ports.device\_id == instances.id:

Instances\_Interface = etree.SubElement(Instances\_ID,'Interface')

Instances\_Interface.attrib['ID'] = ports.id

Instances\_IP = etree.SubElement(Instances\_Interface,'IP')

Instances\_IP.text = ports.fixed\_ips[0]['ip\_address']

**#Take stamp when profiling is finished. Save the XML file with a timestamped name.**

TimeStamp = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d--%H-%M-%S')

Filename = 'OpenStackProfile-' + TimeStamp + ".xml"

Profile\_Output = open(Filename,'w')

print (prettify(Profile),file=Profile\_Output)

Profile\_Output.close()

**#Take the XML in it’s initial one line form, parse it with indentation**

def prettify(elem):

rough\_string = etree.tostring(elem, 'utf-8')

reparsed = minidom.parseString(rough\_string)

return reparsed.toprettyxml(indent=" ")

# Appendix B: ReloadScript.py

import sys

import hashlib

import requests

import ipaddress

import os

import os\_client\_config

import time

import datetime

import xml.etree.ElementTree as ET

import mmap

from lxml import objectify,etree

from openstack import connection

from openstack import profile

from openstack import utils

from python\_hosts import Hosts, HostsEntry

from xml.etree import ElementTree as etree

from xml.dom import minidom

#The function Create\_From\_XML takes as input the connection to the OpenStack Instance

#The function iterates through the XML file. It parses and the elements and create the corresponding dictionaries

#The function then creates the corresponding entities, starting with subnets, and networks, to routers and instances

def Create\_From\_XML(conn):

xmlFile = input("Please enter the name of the XML to load from\n")

file\_root = objectify.parse(xmlFile).getroot()

Networks, Subnets, Routers, Interfaces, Instances = {}, {}, {}, {}, {}

#Parse the XML File to Create Router Dictionary

for Router in file\_root.Routers.iterchildren():

Routers[Router.get("ID")],temp = {},{}

**#Parse all children under Router node, iterate over each child by ID, and save the corresponding values in a dictionary.**

for RDetails in Router.iterchildren():

**#The ID value is saved in the Key:Value pair. When we have interfaces details, we have to take children iteration one step lower.**

**#This is detected by the Interface tag.**

if RDetails.tag == 'Interface':

temp.setdefault(RDetails.tag,[]).append(RDetails.attrib)

else:

temp.update({RDetails.tag:RDetails.text})

Routers[Router.get("ID")] = temp

#Parse the XML File to Create Interfaces Dictionary

for Interface in file\_root.Interfaces.iterchildren():

Interfaces[Interface.get("ID")],temp = {}, {}

**#Parse all children under Interfaces node, iterate over each child by ID, and save the corresponding values in a dictionary**

for IDetails in Interface.iterchildren():

**#Save each ID as unique key, with corresponding dictionary as the value**

if IDetails.tag == "InterfaceSubnet":

temp3 = IDetails.attrib

**#Subnet of the interface is a nested child in the XML file, hence need to iterate one step lower and save the results in Interfaces dictionary**

for IDS in IDetails.iterchildren():

temp2 = {}

temp2 = {IDS.tag:IDS.text}

temp2.update(temp3)

temp.setdefault(IDetails.tag,[]).append(temp2)

else:

temp.update({IDetails.tag:IDetails.text})

Interfaces[Interface.get("ID")] = temp

#Parse the XML File to Create Networks Dictionary

for Network in file\_root.Networks.iterchildren():

Networks[Network.get("ID")],temp,temp3 = {},{},{}

**#Parse all children under Networknode, iterate over each child by ID, and save the corresponding values in a dictionary**

for NDetails in Network.iterchildren():

temp2 = {}

**#Each network has corresponding subnets nested within it. Hence, we need to iterate one step lower when a Subnet tag is identified. Subnet and Network results are saved in the corresponding dictionaries.**

if NDetails.tag == "Subnet":

Subnets[NDetails.get("ID")]={}

for NDS in NDetails.iterchildren():

temp2.update({NDS.tag:NDS.text})

Subnets[NDetails.get("ID")]=temp2

temp.setdefault(NDetails.tag,[]).append(NDetails.attrib)

else:

temp.update({NDetails.tag:NDetails.text})

Networks[Network.get("ID")] = temp

#Parse the XML File to Create Instance Dictionary

for Instance in file\_root.Instances.iterchildren():

Instances[Instance.get("ID")],temp, temp3 = {}, {}, {}

**#Parse all children under Instances node, iterate over each child by ID, and save the corresponding values in a dictionary**

for IDetails in Instance.iterchildren():

**#Loop over the XML SubElements of Instances, and saved the details in a dictionary accessed by the unique key InstanceID**

if IDetails.tag == "Interface":

temp3 = IDetails.attrib

**#When Interface tag is found, iterate one step lower to parse and save all corresponding values**

for IDS in IDetails.iterchildren():

temp2 = {}

temp2 = {IDS.tag:IDS.text}

temp2.update(temp3)

temp.setdefault(IDetails.tag,[]).append(temp2)

else:

temp.update({IDetails.tag:IDetails.text})

Instances[Instance.get("ID")] = temp

#Itemize the Network and Subnet Dictionaries and Create them on OpenStack

for Net,Na in Networks.items():

**#Access the key:value pairs of the Networks dictionary and create the new networks.**

if Na['Network\_Name'] != 'ext-net' and Na['Network\_Name'] != 'tun0-net' and Na['Network\_Name'] != 'flat-lan-1-net':

**#Only create networks that are different from the ones created by default in OpenStack**

if not(conn.network.find\_network(Na['Network\_Name'])):

net = conn.network.create\_network(name=Na['Network\_Name'], is\_admin\_state\_up=Na['Network\_Admin\_State\_UP'])

print("Hello! We Have Not Found The Duplicate")

else:

print("The Network " + Na['Network\_Name'] + " already exists! We will delete before proceeding!")

#Itemize the Subnet Dictionary and Create subnets on OpenStack

for SubID,Sub in Subnets.items():

**#Access the subnets dictionary to get the key value pairs.**

if Sub['Subnet\_Name'] != 'ext-subnet' and Sub['Subnet\_Name'] != 'tun0-subnet' and Sub['Subnet\_Name'] != 'flat-lan-1-subnet':

if not(conn.network.find\_subnet(Sub['Subnet\_Name'])):

**#When the subnet is not one of the default created by OpenStack, we need to create the subnet.**

for Net,Na in Networks.items():

**#Map the subnet to the corresponding Network from the Network dictionary by comparing the values in the dictionary entries.**

for ID, Val in enumerate(Na['Subnet']):

if Val['ID'] == SubID:

new\_network\_id = conn.network.find\_network(Na['Network\_Name']).id

new\_subnet = conn.network.create\_subnet(name=Sub['Subnet\_Name'], network\_id=new\_network\_id, ip\_version=Sub['Subnet\_IP\_Version'], cidr=Sub['Subnet\_CIDR'], gateway\_IP=Sub['Subnet\_IP\_GW'])

print("Hello! We Have Not Found The Duplicate")

else:

print("The Subnet " + Sub['Subnet\_Name'] + " already exists! We will delete before proceeding!")

#Itemize the Router Dictionary and Create the Routers on OpenStack

for RID,RTR in Routers.items():

**#Access the key:value pairs of the Routers dictionary and parse them**

if RTR['Router\_Name'] != 'tun0-router' and RTR['Router\_Name'] != 'flat-lan-1-router':

**#Only create routers when they are different than the default ones created by OpenStack**

if not(conn.network.find\_router(RTR['Router\_Name'])):

rtr = conn.network.create\_router(name=RTR['Router\_Name'], is\_admin\_state\_up=RTR['Router\_Admin\_State\_UP'])

print("Hello! We Have Not Found The Duplicate")

else:

print("The Router " + RTR['Router\_Name'] + " already exists! We will delete before proceeding!")

else:

print("No Configured Routers Other Than Defaults in XML")

#Itemize the Router Dictionary and Create the Routers on OpenStack

for IID,Inst in Instances.items():

if not(conn.compute.find\_server(Inst['Name'])):

**#Access key value pairs of the instances dictionary**

for X,Y in enumerate(Inst['Interface']):

Port\_Details = Interfaces[Y['ID']]

Network\_Details = Networks [Port\_Details['Network\_ID']]

**#Get the corresponding subnet for each dictionary by company the Subnet\_ID in the instance to the list of newly created Subnets.**

for A,B in enumerate(Port\_Details['InterfaceSubnet']):

Subnet\_Details = Subnets[B['ID']]

SubnetID = conn.network.find\_subnet(Subnet\_Details['Subnet\_Name']).id

server = conn.compute.create\_server(name=Inst['Name'], image\_id=conn.image.find\_image(Inst['Image']).id, flavor\_id=conn.compute.find\_flavor(Inst['Flavor']).id, networks=[{"uuid": conn.network.find\_network(Network\_Details['Network\_Name']).id}])

server = conn.compute.wait\_for\_server(server)

**#Get the list of newly created ports, associate the correct IP to the corresponding server/VM**

for port in conn.network.ports():

**#Set the new IP address of the VM as the same to the one found in the XML file. Thus maintain data integrity.**

if port.device\_id == server.id:

IP\_A = [{'subnet\_id':SubnetID, 'ip\_address':Y['IP']}]

conn.network.update\_port(port, admin\_state\_up=Port\_Details['Interface\_Admin\_state\_UP'], fixed\_ips=IP\_A)

break

# Appendix C: MasterScript.py

import sys

import hashlib

import requests

import ipaddress

import os

import os\_client\_config

import time

import datetime

import xml.etree.ElementTree as ET

import mmap

from lxml import objectify,etree

from openstack import connection

from openstack import profile

from openstack import utils

from python\_hosts import Hosts, HostsEntry

from xml.etree import ElementTree as etree

from xml.dom import minidom

#The function update\_hosts\_file is responsible for update the hosts file inside a Linux environment.

#It is included for use to update with the IP of the CTL server in case the OpenStack instance has

#changed or IP has been modified. It first finds and deletes previous definitions of ctl before

#establishing a new one.

#Input variable to this function is the IP address of the CTL, which can be retrieved by a simple

#ifconfig command on the Shell of the CTL.

def update\_hosts\_file(IP):

hosts = Hosts(path='/etc/hosts')

hosts.remove\_all\_matching(name='ctl')

new\_entry = HostsEntry(entry\_type='ipv4', address=str(IP), names=['ctl'])

hosts.add([new\_entry])

hosts.write()

#The function create\_connection is responsible for establishing a new connection with the

#OpenStack instance. This connection is needed by other functions to grab information or create

#newer ones.

#Input variables are the authentication URL, region, project name, username and password.

#These input variables can be grabbed from OpenStack by going through Access and Security Tab, under

#Project Menu, and downloading the admin-openrc file.

def create\_connection(URL, region, p\_name, p\_username, p\_password):

prof = profile.Profile()

prof.set\_region(profile.Profile.ALL, region)

return connection.Connection(

profile=prof,

auth\_url=URL,

project\_name = p\_name,

user\_domain\_name = 'default',

project\_domain\_name = 'default',

username = p\_username,

password = p\_password

)

#Input variable is the established connection to the OpenStack instance.

#Output is a list of configured subnets.

def list\_all\_subnets(conn):

for networks in conn.network.networks():

name = networks.name

netid = networks.id

print ("########################################################")

print ("Network Name: "+name+" || ",end="")

for subnets in conn.network.subnets():

if subnets.network\_id == netid:

print("Subnet Name: " + subnets.name + " || CIDR Range: " + subnets.cidr+" ||",end=" ")

print ("########################################################")

print ("########################################################")

#Input variable is the established connection to the OpenStack instance.

#Output is a list of the uploaded images.

def list\_all\_images(conn):

print("Images are:")

for image in conn.image.images():

print(image.name)

#The function list\_all\_flavors is tasked with retrieving all the flavours on an instance.

#Input variable is the established connection to the OpenStack instance.

#Output is a list of the uploaded images.

def list\_all\_flavors(conn):

print("Flavors are:\n")

for flavor in conn.compute.flavors():

print(flavor.name)

#The function create\_new\_subnet is tasked with creating a new subnet into the OpenStack instance.

#Input variables are the established connection to the OpenStack instance, as well as network name, subnet name,

#CIDR Block, and gateway IP address.

def create\_new\_subnet(conn, net\_name, sub\_name, versionIP, cidr, gatewayIP):

print("Creating New Network Procedure:")

new\_network = conn.network.create\_network(name=net\_name)

new\_subnet = conn.network.create\_subnet(name=sub\_name, network\_id=new\_network.id, ip\_version=versionIP, cidr=str(cidr), gateway\_IP=str(gatewayIP))

#The function create\_new\_subnet is tasked with creating a new subnet into the OpenStack instance.

#Input variables are the established connection to the OpenStack instance, as well as network name, subnet name,

#CIDR Block, and gateway IP address.

def download\_image(conn):

list\_all\_images(conn)

image\_name = input("Please input the image name you want to download")

image = conn.image.find\_image(image\_name)

md5 = hashlib.md5()

url = utils.urljoin('/images', image.id, 'file')

session = conn.session

with open(image\_name, "wb") as local\_image:

response = session.get(url, endpoint\_filter=image.service,stream=True)

#response = conn.image.download\_image(image)

for chunk in response.iter\_content(chunk\_size=1024):

md5.update(chunk)

local\_image.write(chunk)

if response.headers["Content-MD5"] != md5.hexdigest():

raise Exception("Checksum mismatch in downloaded content")

#The function upload\_new\_image is tasked with uploading a new image into the OpenStack instance.

#Assumption is that the user has the image stored locally on the control machine, knows the name and its details.

#Input variables are the established connection to the OpenStack instance, the image name and location, the image format (VDI, ISO, RAW)

def upload\_new\_image (conn):

image\_name = input("Plese enter the name of the image you would like to upload")

container = input("Please enter the Container Format as one of ami, ari, aki, bare,ovf, ova, or docker")

disk = input ("Please enter the Disk Format as one of ami, ari, aki, vhd, vmdk, raw, qcow2, vdi, or iso")

fimage = open(image\_name,'rb')

mmapped\_file\_as\_string = mmap.mmap(fimage.fileno(), 0, access=mmap.ACCESS\_READ)

conn.image.upload\_image(name=image\_name,data=mmapped\_file\_as\_string, container\_format=container, disk\_format=disk)

mmapped\_file\_as\_string.close()

fimage.close()

#The function create\_new\_instance is tasked with instantiating a new VM instance in the OpenStack instance.

#Input variable are the established connection to the OpenStack instance, the instance name, the image name, the flavour name and the attached network.

def create\_new\_instance(conn):

image\_name = input("Please input the image name you would like to use:\n")

flavor\_name = input("Please input the flavor name you would like to use:\n")

network\_name = input("Please input the network name you would like to use:\n")

instance\_name = input("Please input the name you would like to give your instance:\n")

image = conn.compute.find\_image(image\_name)

flavor = conn.compute.find\_flavor(flavor\_name)

network = conn.network.find\_network(network\_name)

server = conn.compute.create\_server(name=instance\_name, image\_id=image.id, flavor\_id=flavor.id, networks=[{"uuid": network.id}])

server = conn.compute.wait\_for\_server(server)

#The function list\_all\_instances is tasked with returning a list of all the configured VM instances.

#It will provide the list of names, which can be modified to also add a list of IDs, images per VM,...

#Input variables are the established connection to the OpenStack instance.

def list\_all\_instances(conn):

print("Configured Servers and their states are:")

for server in conn.compute.servers():

print(server.name+"||"+server.status+"||", end='')

for ip in conn.compute.server\_ips(server):

print(str(ip.address)+"||", end='')

print(" ")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#The function start\_VM\_instance is tasked with turning on a VM.

#Input variables are the established connection to the OpenStack instance and the VM ID or VM name.

def start\_VM\_instance(conn):

list\_all\_instances(conn)

instance\_name = input("Please enter the name of the server you would like to start")

instance = conn.compute.find\_server(instance\_name)

conn.compute.start\_server(instance)

#The function stop\_VM\_instance is tasked with turning off a VM.

#Input variables are the established connection to the OpenStack instance and the VM ID or VM name.

def stop\_VM\_instance(conn):

list\_all\_instances(conn)

instance\_name = input("Please enter the name of the server you would like to stop")

instance = conn.compute.find\_server(instance\_name)

conn.compute.stop\_server(instance)

#The function list\_free\_floating is tasked with returning a list of all configured and unassigned.

#Input variables are the established connection to the OpenStack instance.

def list\_free\_floating(conn):

print("Free Unassigned Floating IP Addresses are are:")

for ip in conn.network.ips():

print (ip.floating\_ip\_address)

#The function create\_floating\_ip is tasked with creating a new floating IP address from the assigned pool.

#Input variables are the established connection to the OpenStack instance.

def create\_floating\_ip(conn):

print("Creating New Floating IP.")

external\_network = conn.network.find\_network("ext-net")

conn.network.create\_ip(floating\_network\_id=external\_network.id)

#The function create\_new\_router is tasked with creating a new network router.

#Input variables are the established connection to the OpenStack instance.

def create\_new\_router(conn):

router\_name = input("Please enter desired router name\n")

conn.network.create\_router(name=router\_name)

#The function create\_new\_router\_interface is tasked with attaching a router to an interface.

#The function will first list the different networks available for the router to hook unto.

#Input variables are the established connection to the OpenStack instance.

def create\_new\_router\_interface(conn):

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print ("List of available configured routers:")

for routers in conn.network.routers():

print (routers.name)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

router\_name = input("Please enter desired router name\n")

router\_id = conn.network.find\_router(router\_name)

router\_idd = router\_id.id

print ("Selected Router has the following configured IP addresses:")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

for ports in conn.network.ports():

if ports.device\_id == router\_idd:

print (ports.fixed\_ips[0]['ip\_address'])

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

list\_all\_subnets(conn)

new\_interface\_network = input ("Please enter the network name you would like add")

new\_network = conn.network.find\_network(new\_interface\_network)

conn.network.create\_port(admin\_state\_up=True, device\_id=router\_idd, network\_id=new\_network.id)

#The function take\_server\_snapshot is tasked with taken a snapshot of a given VM

#The snapshot will be available at the CTL on the /var/lib/glance/images folder

#We start by listing the configured instances, find the server instance based on name and passing it to the API call

#Input variables are the established connection to the OpenStack instance.

def take\_server\_snapshot(conn):

list\_all\_instances(conn)

instance\_name = input ("Please enter the name of the instance you wish to snapshot")

instance = conn.compute.find\_server(instance\_name)

conn.compute.shelve\_server(instance)

#The function add\_VM\_IP is tasked with adding a Fixed IP Address to a VM of choice

#The function starts by listing all the configured subnets and instances

#User is prompted to enter the desired VM to add an IP to, as well as the desired network

#Inpute variables are the established connection to the OpenStack instance.

def add\_VM\_IP(conn):

list\_all\_subnets(conn)

list\_all\_instances(conn)

instance\_name = input ("Please choose VM to add IP to")

instance = conn.compute.find\_server(instance\_name)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("The Selected Server has the following IP addresses configured")

for ip in conn.compute.server\_ips(instance):

print(ip)

network\_name = input ("Please enter the network name you would like to add")

network = conn.network.find\_network(network\_name)

port = conn.network.create\_port(admin\_state\_up=True, network\_id=network.id)

conn.compute.create\_server\_interface(server=instance, port\_id=port.id)

def list\_all\_routers(conn):

for router in conn.network.routers():

print (router)

# Appendix D: TesterScript.py

import argparse

import os

import os\_client\_config

import MasterScript

import test

import ipaddress

import sys

from openstack import connection

from openstack import profile

from openstack import utils

#Starting code to update the CTL IP address in the hosts file.

while True:

try:

CTL = ipaddress.ip\_address(input("Please enter the IP address of the CTL\n"))

break

except ValueError:

print("Not a valid IP address")

MasterScript.update\_hosts\_file(CTL)

while True:

print ("1 to Establish a New Connection")

print ("2 to List All Subnets By CIDR")

print ("3 to List All Images By Name")

print ("4 to Create a New Subnet")

print ("5 to List All Flavors By Name")

print ("6 to Create New VM Instance")

print ("7 to List All Configured Instances")

print ("8 to List All Configured and Unassigned Public IP Addresses")

print ("9 to Create a New Floating IP Address")

print ("10 to Create a New Network Router")

print ("11 to Attach Router to New Network")

print ("12 to Take a Snapshot of a Server")

print ("13 to Start a VM")

print ("14 to Stop a VM")

print ("15 to add new IP address to a VM")

print ("16 to Profile and Fingerprint Your OpenStack Instance")

print ("18 to Load you OpenStack Cloud From XML")

print ("19 to Upload a New Image")

print ("20 to Download an Existing Image")

print ("Press 999 at anytime to quit")

Choice = int(input("Please input the value of the operation you would like to perform\n"))

if Choice == 1:

#Code to ask for the username and password for the OpenStack instance.

#Create connection will be used.

URL = input("Please input the authorization URL from admin-openrc.sh\n")

region = input("Please enter the region\n")

p\_name = input("Please enter the project name\n")

p\_username = input("Please enter the project username\n")

p\_password = input("Please enter the password\n")

connection = MasterScript.create\_connection(URL,region,p\_name,p\_username,p\_password)

elif Choice == 2:

MasterScript.list\_all\_subnets(connection)

elif Choice == 3:

MasterScript.list\_all\_images(connection)

elif Choice == 4:

netname = input("Please input the desired network name\n")

subname = input("Please input the desired subnet name\n")

version = input("Please input the version 4 or 6\n")

while True:

try:

CIDR = ipaddress.ip\_network(input("Please input the CIDR Block\n"))

gateway = ipaddress.ip\_address(input("Please input the gatewayIP\n"))

break

except ValueError:

print("Error in your CIDR or gateway, please retry")

MasterScript.create\_new\_subnet(connection,netname,subname,version,CIDR,gateway)

elif Choice == 5:

MasterScript.list\_all\_flavors(connection)

elif Choice == 6:

MasterScript.create\_new\_instance(connection)

elif Choice == 7:

MasterScript.list\_all\_instances(connection)

elif Choice == 8:

MasterScript.list\_free\_floating(connection)

elif Choice == 9:

MasterScript.create\_floating\_ip(connection)

elif Choice == 10:

MasterScript.create\_new\_router(connection)

elif Choice == 11:

MasterScript.create\_new\_router\_interface(connection)

elif Choice == 12:

MasterScript.take\_server\_snapshot(connection)

elif Choice == 13:

MasterScript.start\_VM\_instance(connection)

elif Choice == 14:

MasterScript.stop\_VM\_instance(connection)

elif Choice == 15:

MasterScript.add\_VM\_IP(connection)

elif Choice == 16:

MasterScript.Profile\_OpenStack(connection)

elif Choice == 18:

MasterScript.create\_from\_xml(connection)

elif Choice == 19:

MasterScript.upload\_new\_image(connection)

elif Choice == 20:

MasterScript.download\_image(connection)

elif Choice == 21:

MasterScript.add\_VM\_Public\_IP(connection)

elif Choice == 999:

break

# Appendix E: VMSSH.py

import paramiko

import sys

import subprocess

k = paramiko.RSAKey.from\_private\_key\_file("/home/ubuntu/Keys/privnopass")

vm=paramiko.SSHClient()

vm.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

vm.connect( hostname = "128.110.153.205", username = "hkanaan", pkey = k )

vmtransport = vm.get\_transport()

dest\_addr = ('10.11.10.9', 22)

local\_addr = ('10.11.10.1', 22)

vmchannel = vmtransport.open\_channel("direct-tcpip", dest\_addr, local\_addr)

jhost=paramiko.SSHClient()

jhost.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

jhost.connect('10.11.10.9', username='ubuntu', password='c0f948618be6', sock=vmchannel)

stdin, stdout, stderr = jhost.exec\_command("ls -l")

print (stdout.read())

jhost.close()

vm.close()

Appendix F: MasterScript.py With Added Functionality

import sys

import hashlib

import requests

import ipaddress

import os

import os\_client\_config

import time

import datetime

import xml.etree.ElementTree as ET

import mmap

import pdb

import threading

import \_thread

import mysql.connector

from lxml import objectify,etree

from openstack import connection

from openstack import profile

from openstack import utils

from python\_hosts import Hosts, HostsEntry

from xml.etree import ElementTree as etree

from xml.dom import minidom

def create\_from\_xml(conn):

xmlFile = input("Please enter the name of the XML to load from\n")

file\_root = objectify.parse(xmlFile).getroot()

Networks, Subnets, Routers, Interfaces, Instances = {}, {}, {}, {}, {}

#Parse the XML File to Create Router Dictionary

for Router in file\_root.Routers.iterchildren():

Routers[Router.get("ID")],temp = {},{}

for RDetails in Router.iterchildren():

if RDetails.tag == 'Interface':

temp3 = RDetails.attrib

for RDS in RDetails.iterchildren():

temp2 = {}

temp2 = {RDS.tag:RDS.text}

temp2.update(temp3)

temp.setdefault(RDetails.tag,[]).append(temp2)

else:

temp.update({RDetails.tag:RDetails.text})

Routers[Router.get("ID")] = temp

#Parse the XML File to Create Interfaces Dictionary

for Interface in file\_root.Interfaces.iterchildren():

Interfaces[Interface.get("ID")],temp = {}, {}

for IDetails in Interface.iterchildren():

if IDetails.tag == "InterfaceSubnet":

temp3 = IDetails.attrib

for IDS in IDetails.iterchildren():

temp2 = {}

temp2 = {IDS.tag:IDS.text}

temp2.update(temp3)

temp.setdefault(IDetails.tag,[]).append(temp2)

else:

temp.update({IDetails.tag:IDetails.text})

Interfaces[Interface.get("ID")] = temp

#Parse the XML File to Create Networks Dictionary

for Network in file\_root.Networks.iterchildren():

Networks[Network.get("ID")],temp,temp3 = {},{},{}

for NDetails in Network.iterchildren():

temp2 = {}

if NDetails.tag == "Subnet":

Subnets[NDetails.get("ID")]={}

for NDS in NDetails.iterchildren():

temp2.update({NDS.tag:NDS.text})

Subnets[NDetails.get("ID")]=temp2

temp.setdefault(NDetails.tag,[]).append(NDetails.attrib)

else:

temp.update({NDetails.tag:NDetails.text})

Networks[Network.get("ID")] = temp

#Parse the XML File to Create Instance Dictionary

for Instance in file\_root.Instances.iterchildren():

Instances[Instance.get("ID")],temp, temp3 = {}, {}, {}

for IDetails in Instance.iterchildren():

if IDetails.tag == "Interface":

temp3 = IDetails.attrib

for IDS in IDetails.iterchildren():

temp2 = {}

temp2 = {IDS.tag:IDS.text}

temp2.update(temp3)

temp.setdefault(IDetails.tag,[]).append(temp2)

else:

temp.update({IDetails.tag:IDetails.text})

Instances[Instance.get("ID")] = temp

#Itermize the Network and Subnet Dictionaries and Create them on OpenStack

for Net,Na in Networks.items():

if Na['Network\_Name'] != 'ext-net' and Na['Network\_Name'] != 'tun0-net' and Na['Network\_Name'] != 'flat-lan-1-net':

if not(conn.network.find\_network(Na['Network\_Name'])):

net = conn.network.create\_network(name=Na['Network\_Name'], is\_admin\_state\_up=Na['Network\_Admin\_State\_UP'])

print("Hello! We Have Not Found The Duplicate")

else:

print("The Network " + Na['Network\_Name'] + " already exists! We will delete before proceeding!")

#Itermize the Subnet Dictionary and Create subnets on OpenStack

for SubID,Sub in Subnets.items():

if Sub['Subnet\_Name'] != 'ext-subnet' and Sub['Subnet\_Name'] != 'tun0-subnet' and Sub['Subnet\_Name'] != 'flat-lan-1-subnet':

if not(conn.network.find\_subnet(Sub['Subnet\_Name'])):

for Net,Na in Networks.items():

for ID, Val in enumerate(Na['Subnet']):

if Val['ID'] == SubID:

new\_network\_id = conn.network.find\_network(Na['Network\_Name']).id

new\_subnet = conn.network.create\_subnet(name=Sub['Subnet\_Name'], network\_id=new\_network\_id, ip\_version=Sub['Subnet\_IP\_Version'], cidr=Sub['Subnet\_CIDR'], gateway\_IP=Sub['Subnet\_IP\_GW'])

print("Hello! We Have Not Found The Duplicate")

else:

print("The Subnet " + Sub['Subnet\_Name'] + " already exists! We will delete before proceeding!")

#Itermize the Router Dictionary and Create the Routers on OpenStack

for RID,RTR in Routers.items():

if RTR['Router\_Name'] != 'tun0-router' and RTR['Router\_Name'] != 'flat-lan-1-router':

if not(conn.network.find\_router(RTR['Router\_Name'])):

rtr = conn.network.create\_router(name=RTR['Router\_Name'], is\_admin\_state\_up=RTR['Router\_Admin\_State\_UP'])

print("Hello! We Have Not Found The Duplicate")

for X,Y in enumerate(RTR['Interface']):

Port\_Details = Interfaces[Y['ID']]

Network\_Details = Networks[Port\_Details['Network\_ID']]

for A,B in enumerate(Port\_Details['InterfaceSubnet']):

Subnet\_Details = Subnets[B['ID']]

SubnetID = conn.network.find\_subnet(Subnet\_Details['Subnet\_Name']).id

if (Subnet\_Details['Subnet\_Name'] != 'ext-subnet'):

print (Subnet\_Details['Subnet\_Name'])

p = conn.network.create\_port(admin\_state\_up=True, device\_id=rtr.id, network\_id=conn.network.find\_network(Network\_Details['Network\_Name']).id)

for port in conn.network.ports():

if port.device\_id == rtr.id:

IP\_A = [{'subnet\_id':SubnetID,'ip\_address':Y['IP']}]

conn.network.update\_port(port, admin\_state\_up=Port\_Details['Interface\_Admin\_state\_UP'], fixed\_ips=IP\_A)

break

else:

print("The Router " + RTR['Router\_Name'] + " already exists! We will delete before proceeding!")

#else:

# print("No Configured Routers Other Than Defaults in XML")

#Create VM instances and connect them to the proper network

for IID,Inst in Instances.items():

if not(conn.compute.find\_server(Inst['Name'])):

for X,Y in enumerate(Inst['Interface']):

Port\_Details = Interfaces[Y['ID']]

Network\_Details = Networks[Port\_Details['Network\_ID']]

for A,B in enumerate(Port\_Details['InterfaceSubnet']):

Subnet\_Details = Subnets[B['ID']]

SubnetID = conn.network.find\_subnet(Subnet\_Details['Subnet\_Name']).id

server = conn.compute.create\_server(name=Inst['Name'], image\_id=conn.image.find\_image(Inst['Image']).id, flavor\_id=conn.compute.find\_flavor(Inst['Flavor']).id, networks=[{"uuid": conn.network.find\_network(Network\_Details['Network\_Name']).id}])

server = conn.compute.wait\_for\_server(server)

for port in conn.network.ports():

if port.device\_id == server.id:

IP\_A = [{'subnet\_id':SubnetID, 'ip\_address':Y['IP']}]

conn.network.update\_port(port, admin\_state\_up=Port\_Details['Interface\_Admin\_state\_UP'], fixed\_ips=IP\_A)

break

def prettify(elem):

rough\_string = etree.tostring(elem, 'utf-8')

reparsed = minidom.parseString(rough\_string)

return reparsed.toprettyxml(indent=" ")

#The function update\_hosts\_file is responsible for update the hosts file inside a Linux environment.

#It is included for use to update with the IP of the CTL server in case the OpenStack instance has

#changed or IP has been modified. It first finds and deletes previous definitions of ctl before

#establishing a new one.

#Input varible to this function is the IP address of the CTL, which can be retrieved by a simple

#ifconfig command on the Shell of the CTL.

def update\_hosts\_file(IP):

hosts = Hosts(path='/etc/hosts')

hosts.remove\_all\_matching(name='ctl')

new\_entry = HostsEntry(entry\_type='ipv4', address=str(IP), names=['ctl'])

hosts.add([new\_entry])

hosts.write()

#The function create\_connection is responsible for establishing a new connection with the

#OpenStack instance. This connection is needed by other functions to grab information or create

#newer ones.

#Input variables are the authentication URL, region, project name, username and password.

#These input variables can be grabbed from OpenStack by gooing through Access and Security Tab, under

#Project Menu, and downloading the admin-openrc file.

def create\_connection(URL, region, p\_name, p\_username, p\_password):

prof = profile.Profile()

prof.set\_region(profile.Profile.ALL, region)

return connection.Connection(

profile=prof,

auth\_url=URL,

project\_name = p\_name,

user\_domain\_name = 'default',

project\_domain\_name = 'default',

username = p\_username,

password = p\_password

)

#Input variable is the established connection to the OpenStack instance.

#Output is a list of configured subnets.

def list\_all\_subnets(conn):

for networks in conn.network.networks():

name = networks.name

netid = networks.id

print ("########################################################")

print ("Network Name: "+name+" || ",end="")

for subnets in conn.network.subnets():

if subnets.network\_id == netid:

print("Subnet Name: " + subnets.name + " || CIDR Range: " + subnets.cidr+" ||",end=" ")

print ("########################################################")

print ("########################################################")

#Input variable is the easblished connection to the OpenStack instance.

#Output is a list of the uploaded images.

def list\_all\_images(conn):

print("Images are:")

for image in conn.image.images():

print(image.name)

#The function list\_all\_flavors is tasked with retreiving all the flavors on an instance.

#Input variable is the easblished connection to the OpenStack instance.

#Output is a list of the uploaded images.

def list\_all\_flavors(conn):

print("Flavors are:\n")

for flavor in conn.compute.flavors():

print(flavor.name)

#The function create\_new\_subnet is tasked with creating a new subnet into the OpenStack instance.

#Input variables are the established connection to the OpenStack instance, as well as network name, subnet name,

#CIDR Block, and gateway IP address.

def create\_new\_subnet(conn, net\_name, sub\_name, versionIP, cidr, gatewayIP):

print("Creating New Network Procedure:")

new\_network = conn.network.create\_network(name=net\_name)

new\_subnet = conn.network.create\_subnet(name=sub\_name, network\_id=new\_network.id, ip\_version=versionIP, cidr=str(cidr), gateway\_IP=str(gatewayIP))

#The function create\_new\_subnet is tasked with creating a new subnet into the OpenStack instance.

#Input variables are the established connection to the OpenStack instance, as well as network name, subnet name,

#CIDR Block, and gateway IP address.

def download\_image(conn):

list\_all\_images(conn)

image\_name = input("Please input the image name you want to download")

image = conn.image.find\_image(image\_name)

md5 = hashlib.md5()

url = utils.urljoin('/images', image.id, 'file')

session = conn.session

with open(image\_name, "wb") as local\_image:

response = session.get(url, endpoint\_filter=image.service,stream=True)

#response = conn.image.download\_image(image)

for chunk in response.iter\_content(chunk\_size=1024):

md5.update(chunk)

local\_image.write(chunk)

if response.headers["Content-MD5"] != md5.hexdigest():

raise Exception("Checksum mismatch in downloaded content")

#The function upload\_new\_image is tasked with uploading a new image into the OpenStack instance.

#Assumption is that the user has the image stored locally on the control machine, knows the name and its details.

#Input variables are the established connection to the OpenStack instance, the image name and location, the image format (VDI, ISO, RAW)

def upload\_new\_image (conn):

image\_name = input("Plese enter the name of the image you would like to upload")

container = input("Please enter the Container Format as one of ami, ari, aki, bare,ovf, ova, or docker")

disk = input ("Please enter the Disk Format as one of ami, ari, aki, vhd, vmdk, raw, qcow2, vdi, or iso")

fimage = open(image\_name,'rb')

mmapped\_file\_as\_string = mmap.mmap(fimage.fileno(), 0, access=mmap.ACCESS\_READ)

conn.image.upload\_image(name=image\_name,data=mmapped\_file\_as\_string, container\_format=container, disk\_format=disk)

mmapped\_file\_as\_string.close()

fimage.close()

#The function create\_new\_instance is tasked with instantiating a new VM instance in the OpenStack instnce.

#Input variable are the established connection to the OpenStack instance, the instance name, the image name, the flavor name, the attached network, and number of instances

#with those settings to create.

def create\_new\_instance(conn):

image\_name = input("Please input the image name you would like to use:\n")

flavor\_name = input("Please input the flavor name you would like to use:\n")

network\_name = input("Please input the network name you would like to use:\n")

instance\_name = input("Please input the name you would like to give your instance:\n")

counter = (int(input("Please enter the number of virtual machines you want to create with these settings:\n")))

print (time.strftime("%H:%M:%S"))

print ("Began Creating VMs at " + time.strftime("%H:%M:%S"))

while(counter > 0):

threads = []

t = threading.Thread(target = crThread, args=(conn, counter, image\_name, flavor\_name, network\_name, instance\_name) )

t.start()

threads.append(t)

counter -= 1

print("Starting Main Thread." + time.strftime("%H:%M:%S"))

for t in threads:

t.join()

print("Exiting Main Thread." + time.strftime("%H:%M:%S"))

print ("Finished Creating VMs at " + time.strftime("%H:%M:%S"))

def crThread(conn, count, im, fl, net, nam):

image = conn.compute.find\_image(im)

flavor = conn.compute.find\_flavor(fl)

network = conn.network.find\_network(net)

server = conn.compute.create\_server(name=nam + "-" + str(count), image\_id=image.id, flavor\_id=flavor.id, networks=[{"uuid": network.id}])

server = conn.compute.wait\_for\_server(server)

#The function list\_all\_instances is tasked with returning a list of all the configured VM instances.

#It will provide the list of names, which can be modified to also add a list of IDs, images per VM,...

#The function will also write the list of all configured instances to a file

#Input varibales are the established connection to the OpenStack instance.

def list\_all\_instances(conn):

print("Configured Servers and their states are:")

iptab = open("iptable.txt", "w")

i = 1

for server in conn.compute.servers():

oneline = server.name+"|"+server.status

print(i)

i = i + 1

for ip in conn.compute.server\_ips(server):

oneline = oneline + "|" + str(ip.address)

print(oneline);

iptab.write(oneline + "\n")

iptab.close()

print(" ")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#The function write\_to\_sql is tasked with opening the file that contains the list of all configured instances.

#It also connects to the MySQL databse, CLOUDLAB and inserts the values from the file into the database using queries.

def write\_to\_sql(conn):

db = mysql.connector.connect(user='CloudLab\_admin', password='CloudLab\_2017\_admin', host='localhost', database='CLOUDLAB')

cur = db.cursor()

q1 = 'DELETE FROM CLOUDLAB.ACCOUNT\_HAS\_HOST'

cur.execute(q1)

q2 = 'DELETE FROM CLOUDLAB.ACCOUNT'

cur.execute(q2)

query = "DELETE FROM CLOUDLAB.HOST"

cur.execute(query)

with open("iptable.txt", "r") as iptab:

for content in iptab:

if 'Victim' in content:

private = content.split("|")

query = "INSERT INTO CLOUDLAB.HOST(LOCAL\_IPV4,PUBLIC\_IPV4,ROLE) VALUES (%s, %s, %s)"

private[2] = private[2].strip('\n')

private[2] = private[2].strip('\t')

cur.execute(query, (private[2], 'NULL', 'VICTIM'))

elif 'Attack' in content:

public = content.split("|")

public[3] = public[3].strip('\n')

public[3] = public[3].strip('\t')

query = "INSERT INTO CLOUDLAB.HOST(LOCAL\_IPV4,PUBLIC\_IPV4,ROLE) VALUES (%s, %s, %s)"

cur.execute(query, (public[2], public[3], 'ATTACKER'))

iptab.close()

db.commit()

db.close()

#The function start\_VM\_instance is tasked with turning on a VM.

#Input variables are the established connection to the OpenStack instance and the VM ID or VM name.

def start\_VM\_instance(conn):

list\_all\_instances(conn)

instance\_name = input("Please enter the name of the server you would like to start")

instance = conn.compute.find\_server(instance\_name)

conn.compute.start\_server(instance)

#The function stop\_VM\_instance is tasked with turning off a VM.

#Input variables are the established connection to the OpenStack instance and the VM ID or VM name.

def stop\_VM\_instance(conn):

list\_all\_instances(conn)

instance\_name = input("Please enter the name of the server you would like to stop")

instance = conn.compute.find\_server(instance\_name)

conn.compute.stop\_server(instance)

#The function list\_free\_floating is tasked with returning a list of all configured and unassigned.

#Input varibales are the established connection to the OpenStack instance.

def list\_free\_floating(conn):

print("Free Unassigned Floating IP Addresses are are:")

for ip in conn.network.ips():

print (ip.floating\_ip\_address)

#The function create\_floating\_ip is tasked with creating a new floating IP address from the assigned pool.

#Input varibales are the established connection to the OpenStack instance.

def create\_floating\_ip(conn):

print("Creating New Floating IP.")

external\_network = conn.network.find\_network("ext-net")

conn.network.create\_ip(floating\_network\_id=external\_network.id)

#The function create\_new\_router is tasked with creating a new network router.

#Input varibales are the established connection to the OpenStack instance.

def create\_new\_router(conn):

router\_name = input("Please enter desired router name\n")

conn.network.create\_router(name=router\_name)

#The function create\_new\_router\_interface is tasked with attaching a router to an interface.

#The function will first list the different networks available for the router to hook unto.

#Input varibales are the established connection to the OpenStack instance.

def create\_new\_router\_interface(conn):

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print ("List of available configured routers:")

for routers in conn.network.routers():

print (routers.name)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

router\_name = input("Please enter desired router name\n")

router\_id = conn.network.find\_router(router\_name)

router\_idd = router\_id.id

print ("Selected Router has the following configured IP addresses:")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

for ports in conn.network.ports():

if ports.device\_id == router\_idd:

print (ports.fixed\_ips[0]['ip\_address'])

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

list\_all\_subnets(conn)

new\_interface\_network = input ("Please enter the network name you would like add")

new\_network = conn.network.find\_network(new\_interface\_network)

conn.network.create\_port(admin\_state\_up=True, device\_id=router\_idd, network\_id=new\_network.id)

#The function take\_server\_snapshot is tasked with taken a snapshot of a given VM

#The snapshot will be available at the CTL on the /var/lib/glance/images folder

#We start by listing the configured instances, find the server instance based on name and passing it to the API call

#Input variables are the established connection to the OpenStack instance.

def take\_server\_snapshot(conn):

list\_all\_instances(conn)

instance\_name = input ("Please enter the name of the instance you wish to snapshot")

instance = conn.compute.find\_server(instance\_name)

conn.compute.shelve\_server(instance)

#The function add\_VM\_IP is tasked with adding a Fixed IP Address to a VM of choice

#The function starts by listing all the configured subnets and instances

#User is prompted to enter the desired VM to add an IP to, as well as the desired network

#Inpute variables are the established connection to the OpenStack instance.

def add\_VM\_IP(conn):

list\_all\_subnets(conn)

list\_all\_instances(conn)

instance\_name = input ("Please choose VM to add IP to")

instance = conn.compute.find\_server(instance\_name)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("The Selected Server has the following IP addresses configured")

for ip in conn.compute.server\_ips(instance):

print(ip)

network\_name = input ("Please enter the network name you would like to add")

network = conn.network.find\_network(network\_name)

port = conn.network.create\_port(admin\_state\_up=True, network\_id=network.id)

conn.compute.create\_server\_interface(server=instance, port\_id=port.id)

def Profile\_OpenStack(conn):

Profile = etree.Element('Profile')

Interfaces = etree.SubElement(Profile, 'Interfaces')

for interfaces in conn.network.ports():

Interface\_ID = etree.SubElement(Interfaces,'Interface')

Interface\_ID.attrib['ID'] = interfaces.id

Interface\_Name = etree.SubElement(Interface\_ID,'Interface\_Name')

Interface\_Name.text = interfaces.name

Interface\_Net = etree.SubElement(Interface\_ID,'Network\_ID')

Interface\_Net.text = interfaces.network\_id

Interface\_State = etree.SubElement(Interface\_ID,'Interface\_Admin\_state\_UP')

Interface\_State.text = str(interfaces.is\_admin\_state\_up)

Interface\_Device = etree.SubElement(Interface\_ID,'Connected\_Device\_ID')

Interface\_Device.text = interfaces.device\_id

for i, entry in enumerate(interfaces.fixed\_ips):

Interface\_Sub = etree.SubElement(Interface\_ID,'InterfaceSubnet')

Interface\_Sub.attrib['ID'] = entry['subnet\_id']

Interface\_IP = etree.SubElement(Interface\_Sub,'InterfaceIP')

Interface\_IP.text = entry['ip\_address']

Networks = etree.SubElement(Profile, 'Networks')

for networks in conn.network.networks():

Network\_ID = etree.SubElement(Networks,'Network')

Network\_ID.attrib['ID'] = networks.id

Network\_Name = etree.SubElement(Network\_ID,'Network\_Name')

Network\_Name.text = networks.name

Network\_State = etree.SubElement(Network\_ID,'Network\_Admin\_State\_UP')

Network\_State.text = str(networks.is\_admin\_state\_up)

for subnets in conn.network.subnets():

if subnets.network\_id == networks.id:

Subnet\_Detail\_ID = etree.SubElement(Network\_ID,'Subnet')

Subnet\_Detail\_ID.attrib['ID'] = subnets.id

Subnet\_Details = etree.SubElement(Subnet\_Detail\_ID,'Subnet\_Name')

Subnet\_Details.text = subnets.name

Subnet\_Detail\_CIDR = etree.SubElement(Subnet\_Detail\_ID,'Subnet\_CIDR')

Subnet\_Detail\_CIDR.text = subnets.cidr

Subnet\_Detail\_IP\_Ver = etree.SubElement(Subnet\_Detail\_ID,'Subnet\_IP\_Version')

Subnet\_Detail\_IP\_Ver.text = str(subnets.ip\_version)

Subnet\_Detail\_IP\_GW = etree.SubElement(Subnet\_Detail\_ID,'Subnet\_IP\_GW')

Subnet\_Detail\_IP\_GW.text = str(subnets.gateway\_ip)

Routers = etree.SubElement(Profile,'Routers')

for routers in conn.network.routers():

Router\_ID = etree.SubElement(Routers, 'Router')

Router\_ID.attrib['ID'] = routers.id

Router\_Name = etree.SubElement(Router\_ID,'Router\_Name')

Router\_Name.text = routers.name

Router\_Status = etree.SubElement(Router\_ID,'Router\_Admin\_State\_UP')

Router\_Status.text = str(routers.is\_admin\_state\_up)

for AZ in routers.availability\_zones:

Router\_AvailabilityZone = etree.SubElement(Router\_ID,'Zone')

Router\_AvailabilityZone.text = AZ

for ports in conn.network.ports():

if ports.device\_id == routers.id:

Router\_Interface = etree.SubElement(Router\_ID,'Interface')

Router\_Interface.attrib['ID'] = ports.id

Router\_IP = etree.SubElement(Router\_Interface,'IP')

Router\_IP.text = ports.fixed\_ips[0]['ip\_address']

Images = etree.SubElement(Profile,'Images')

for images in conn.image.images():

Image\_Name = etree.SubElement(Images,'Image')

Image\_Name.attrib['Name'] = images.name

Image\_Container\_Format = etree.SubElement(Image\_Name,'ContainerFormat')

Image\_Container\_Format.text = images.container\_format

Image\_Disk\_Format = etree.SubElement(Image\_Name,'DiskFormat')

Image\_Disk\_Format.text = images.disk\_format

Flavors = etree.SubElement(Profile,'Flavors')

for flavors in conn.compute.flavors():

Flavor\_Name = etree.SubElement(Flavors,'Flavor')

Flavor\_Name.attrib['Name'] = flavors.name

Flavor\_VCPU = etree.SubElement(Flavor\_Name,'VCPU')

Flavor\_VCPU.text = str(flavors.vcpus)

Flavor\_Disk = etree.SubElement(Flavor\_Name,'Disk')

Flavor\_Disk.text = str(flavors.disk)

Flavor\_Ram = etree.SubElement(Flavor\_Name,'RAM')

Flavor\_Ram.text = str(flavors.ram)

Instances = etree.SubElement(Profile,'Instances')

for instances in conn.compute.servers():

Instances\_ID = etree.SubElement(Instances,'Instance')

Instances\_ID.attrib['ID'] = instances.id

Instances\_Name = etree.SubElement(Instances\_ID,'Name')

Instances\_Name.text = instances.name

Instances\_Status = etree.SubElement(Instances\_ID,'Status')

Instances\_Status.text = instances.status

for flavor in conn.compute.flavors():

if flavor.links[1]['href'] == instances.flavor['links'][0]['href']:

Instances\_Flavor = etree.SubElement(Instances\_ID,'Flavor')

Instances\_Flavor.text = flavor.name

for image in conn.image.images():

if image.id == instances.image['id']:

Instances\_Image = etree.SubElement(Instances\_ID,'Image')

Instances\_Image.text = image.name

for ports in conn.network.ports():

if ports.device\_id == instances.id:

Instances\_Interface = etree.SubElement(Instances\_ID,'Interface')

Instances\_Interface.attrib['ID'] = ports.id

Instances\_IP = etree.SubElement(Instances\_Interface,'IP')

Instances\_IP.text = ports.fixed\_ips[0]['ip\_address']

TimeStamp = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d--%H-%M-%S')

Filename = 'OpenStackProfile-' + TimeStamp + ".xml"

Profile\_Output = open(Filename,'w')

print (prettify(Profile),file=Profile\_Output)

Profile\_Output.close()

#The function, create\_multiple\_networks is a function that creates many networks and VMs in parallel.

#The function utilizes the attack and victim images. It also creates 14 floating IP addresses.

#14 floating IP addresses will only be created if a Cloudlab experiment has allocated 16 floating IP's before starting the experiment.

def create\_multiple\_networks(conn):

counter = (int(input("Please enter the number of networks you want to create:\n")))

netname = input("Please input the desired network name\n")

subname = input("Please input the desired subnet name\n")

version = ("4")

image\_name1 = ("Victim")

image\_name2 = ("Attack")

flavor\_name = ("m1.small")

network\_name = (netname)

instance\_name = ("Victim")

instance\_name2 = ("Attacker")

router\_name =("tun0-router")

router\_id = conn.network.find\_router(router\_name)

router\_idd = router\_id.id

new\_interface\_network = (netname)

ipcount = 12

while(counter > 0):

while(ipcount >= 12 and ipcount <= 22):

try:

CIDR = ipaddress.ip\_network("10."+ str(ipcount) + ".0.0/16")

gateway = ipaddress.ip\_address("10." + str(ipcount) + ".0.1")

ipcount += 1

break

except ValueError:

print("Error in your CIDR or gateway, please retry")

threads = []

t = threading.Thread(target = multThread, args=(conn, counter, netname, subname, version, CIDR, gateway, image\_name1, image\_name2, flavor\_name, instance\_name, instance\_name2, router\_name, router\_idd))

t.start()

threads.append(t)

counter -= 1

floatcount = 14

while (floatcount > 0):

external\_network = conn.network.find\_network("ext-net")

conn.network.create\_ip(floating\_network\_id=external\_network.id)

floatcount -= 1

def multThread(conn, count, net, sub, ver, cidr, gate, im1, im2, fl, in1, in2, rname, rid):

new\_network = conn.network.create\_network(name=net + "-" + str(count))

new\_subnet = conn.network.create\_subnet(name=sub + "-" + str(count), network\_id=new\_network.id, ip\_version=ver, cidr=str(cidr), gateway\_IP=str(gate))

image = conn.compute.find\_image(im1)

image2 = conn.compute.find\_image(im2)

flavor = conn.compute.find\_flavor(fl)

network = conn.network.find\_network(net + "-" + str(count))

server1 = conn.compute.create\_server(name=in1 + "-" + str(count), image\_id=image.id, flavor\_id=flavor.id, networks=[{"uuid": network.id}])

server2 = conn.compute.create\_server(name=in2 + "-" + str(count), image\_id=image2.id, flavor\_id=flavor.id, networks=[{"uuid": network.id}])

server1 = conn.compute.wait\_for\_server(server1)

server2 = conn.compute.wait\_for\_server(server2)

# Appendix G: TesterScript with Added Functionality

# import argparse

# import os

# import os\_client\_config

# import MasterScript

# import test

# import ipaddress

# import sys

# from openstack import connection

# from openstack import profile

# from openstack import utils

# #Starting code to update the CTL IP address in the hosts file.

# while True:

# try:

# CTL = ipaddress.ip\_address(input("Please enter the IP address of the CTL\n"))

# break

# except ValueError:

# print("Not a valid IP address")

# MasterScript.update\_hosts\_file(CTL)

# while True:

# print ("1 to Establish a New Connection")

# print ("2 to List All Subnets By CIDR")

# print ("3 to List All Images By Name")

# print ("4 to Create a New Subnet")

# print ("5 to List All Flavors By Name")

# print ("6 to Create New VM Instance")

# print ("7 to List All Configured Instances")

# print ("8 to List All Configured and Unassigned Public IP Addresses")

# print ("9 to Create a New Floating IP Address")

# print ("10 to Create a New Network Router")

# print ("11 to Attach Router to New Network")

# print ("12 to Take a Snapshot of a Server")

# print ("13 to Start a VM")

# print ("14 to Stop a VM")

# print ("15 to add new IP address to a VM")

# print ("16 to Download an Existing Image")

# print ("17 to Profile and Fingerprint Your OpenStack Instance")

# print ("18 to Load you OpenStack Cloud From XML")

# print ("19 to Upload a New Image")

# print ("20 to Create Multiple Networks at Once")

# print ("21 to Upload into SQL")

# print ("Press 999 at anytime to quit")

# Choice = int(input("Please input the value of the operation you would like to perform\n"))

# if Choice == 1:

# #Code to ask for the username and password for the OpenStack instance.

# #Create connection will be used.

# URL = input("Please input the authorization URL from admin-openrc.sh\n")

# region = input("Please enter the region\n")

# p\_name = input("Please enter the project name\n")

# p\_username = input("Please enter the project username\n")

# p\_password = input("Please enter the password\n")

# connection = MasterScript.create\_connection(URL,region,p\_name,p\_username,p\_password)

# #If connection was successful, all of the images will be listed. If not, the program will hang.

# MasterScript.list\_all\_images(connection)

# elif Choice == 2:

# MasterScript.list\_all\_subnets(connection)

# elif Choice == 3:

# MasterScript.list\_all\_images(connection)

# elif Choice == 4:

# MasterScript.create\_new\_subnet(connection)

# elif Choice == 5:

# MasterScript.list\_all\_flavors(connection)

# elif Choice == 6:

# MasterScript.create\_new\_instance(connection)

# elif Choice == 7:

# MasterScript.list\_all\_instances(connection)

# elif Choice == 8:

# MasterScript.list\_free\_floating(connection)

# elif Choice == 9:

# MasterScript.create\_floating\_ip(connection)

# elif Choice == 10:

# MasterScript.create\_new\_router(connection)

# elif Choice == 11:

# MasterScript.create\_new\_router\_interface(connection)

# elif Choice == 12:

# MasterScript.take\_server\_snapshot(connection)

# elif Choice == 13:

# MasterScript.start\_VM\_instance(connection)

# elif Choice == 14:

# MasterScript.stop\_VM\_instance(connection)

# elif Choice == 15:

# MasterScript.add\_VM\_IP(connection)

# elif Choice == 16:

# MasterScript.download\_image(connection)

# 

# elif Choice == 17:

# MasterScript.Profile\_OpenStack(connection)

# elif Choice == 18:

# MasterScript.create\_from\_xml(connection)

# elif Choice == 19:

# MasterScript.upload\_new\_image(connection)

# 

# elif Choice == 20:

# MasterScript.create\_multiple\_networks(connection)

# elif Choice == 21:

# MasterScript.write\_to\_sql(connection)

# 

# elif Choice == 999:

# break

# Appendix H: Documentation For Instructors and Students

Running a Virtual Attack Scenario in CloudLab using Metasploit and Metasploitable

1. **Introduction**

This documentation is split into two parts: The setting up of the experiment, and the lab manual.

Setting up the experiment is to be carried out by the instructor. To set up the experiment, four things are necessary:

* + 1. A CloudLab account **(https://www.cloudlab.us/login.php)**
    2. MobaXterm, an open-source SSH client[1]**(<http://mobaxterm.mobatek.net/>)**
    3. The Metasploitable[2] and Metasploit[3] images, which can be found here:

**[https://archive.org/download/Metasploitable/Metasploitable.vmdk](https://archive.org/download/Metasploitable/Metasploitable.vmdk" \t "_blank)**

**[https://archive.org/download/Metasploit2/Metasploit2](https://archive.org/download/Metasploit2/Metasploit2" \t "_blank)**

(You do not need to download these images, you simply need the links.)

* + 1. The private key to access the Amazon EC2 instance, which can be found from: **https://drive.google.com/open?id=0B-Nx8At6Cvo6Q0hoOER5OXg5akk** (You will need to download this.)

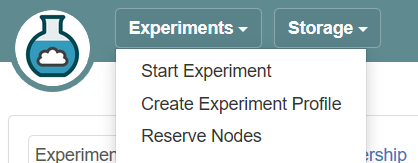
Please note that to create this experiment, there is a time-consuming process that consists of many steps. It can take up to an hour to get everything set up. An experiment on CloudLab automatically expires after *16 hours*, however, the experiment time can be extended. Take this into consideration when setting up your experiment and deciding when experimentation will take place.

The purpose of this experiment is to simulate an attacking scenario on a virtual network. Two virtual images are needed: one for the attacker, one for the victim. The attack image is a Ubuntu image with Metasploit installed. The victim image is Metasploitable, a purposefully vulnerable Linux image. This experiment is for educational purposes only, and is only to be tested in the quarantined virtual environment.

1. **Creating the Experiment (Instructor)**
   1. *Creating a CloudLab Account*
      1. You are required to create a CloudLab account if you do not already have one. If you already have an account, you can skip to section *b: Creating the Experiment.*
      2. Navigate to <https://www.cloudlab.us/>.
      3. Click “Request an account”.

*Figure 1*

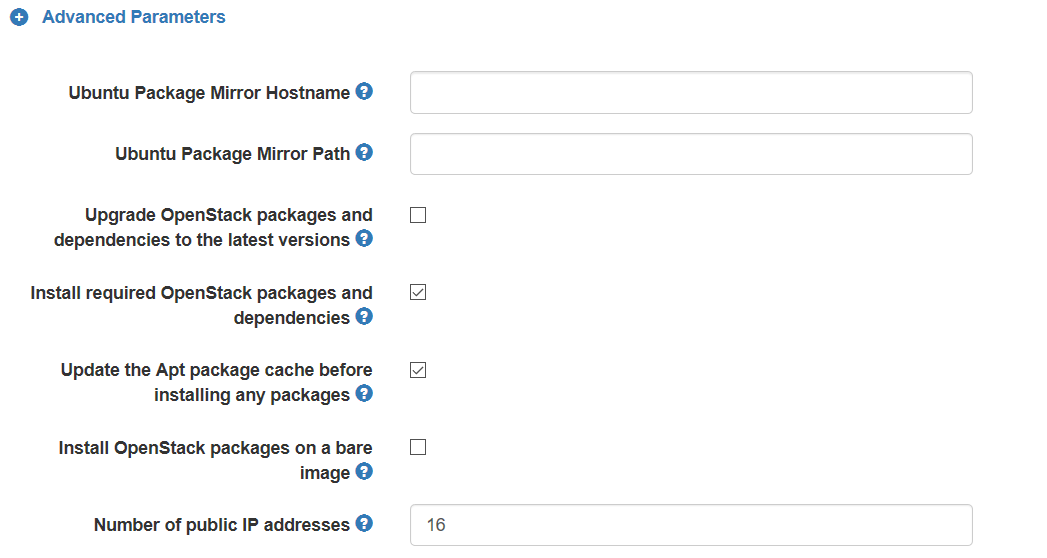
* + 1. Input your personal information. You do not need to upload a public key.
    2. For project information, choose *“Start new project”.* Input some simple information, such as the class you are using this experiment for, and an explanation of the experiment. For example, *“The purpose of this project is to use CloudLab to create a virtual environment for an educational penetration testing experiment.”*
    3. You should receive an e-mail when your account is approved, and then you will be able to login to CloudLab.
  1. *Creating the Experiment*
     1. Login to your CloudLab account*.*
     2. Click on the tab labeled “Experiments” and choose Start Experiment.

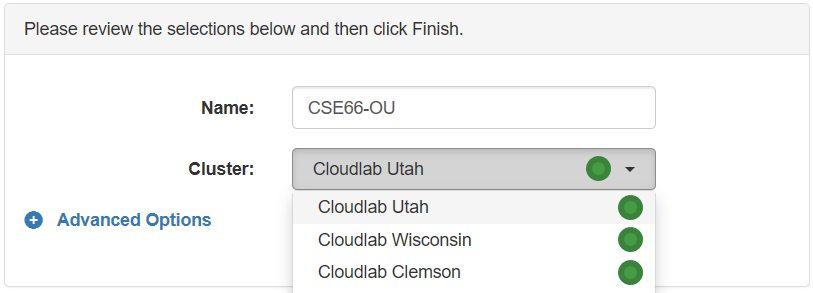


*Figure 2*

* + 1. On the page titled *“Select a Profile”,* click next. So far, there are no settings you need to change from here.
    2. On the page titled *“Parameterize”*, click *“Advanced parameters”*.
    3. In the box next to *“Number of Public IP Addresses”,* change this number to **16**. This will give you **14** public IP addresses you can use for your publicly accessible attack machines, while the other **2** public addresses are used for network configuration. By default, this number cannot be higher than 16. If you are using this experiment with more than 14 public IPs, you will have to create two or more

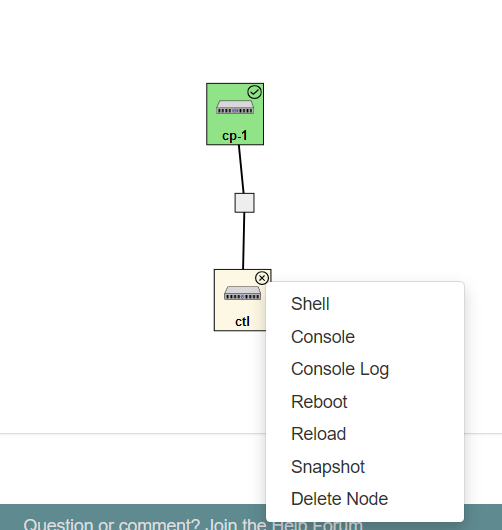
*Figure 3*

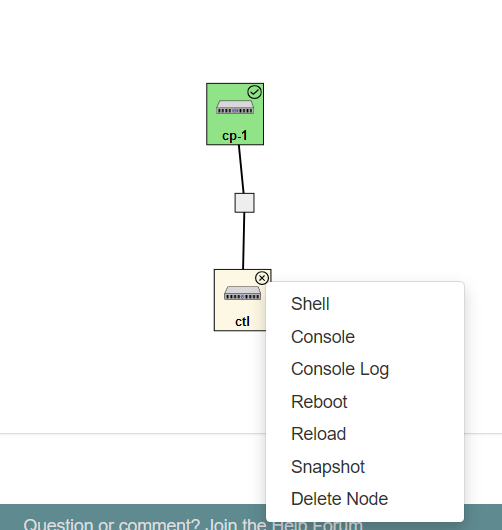
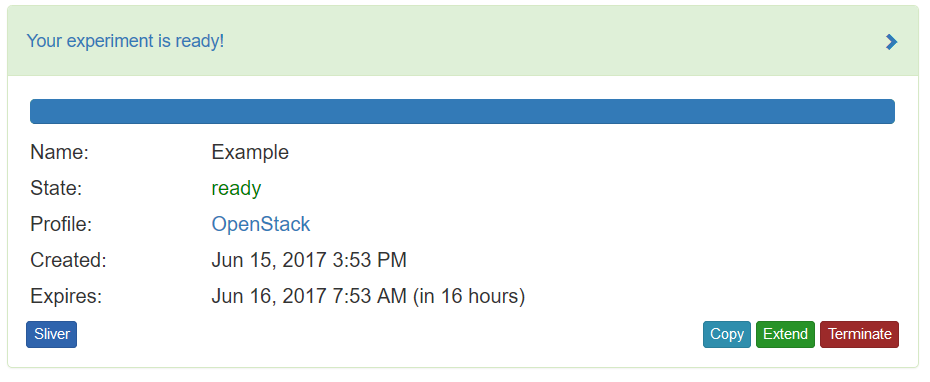
CloudLab experiments.

* + 1. Click next.
    2. Give your experiment a name. For example, **CSE660-OU.** For cluster, choose the *CloudLab Utah cluster.*

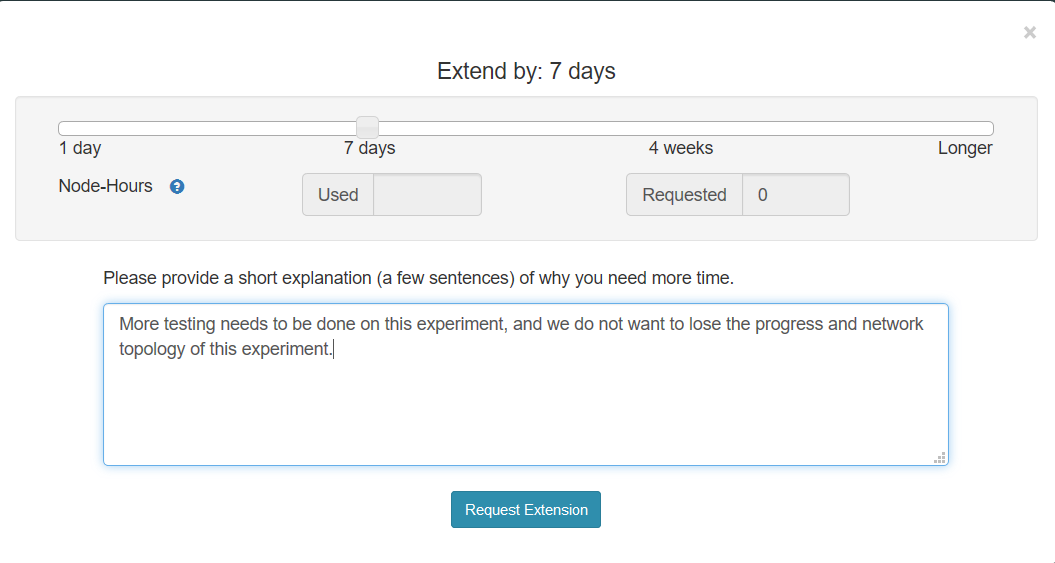
*Figure 4*

* + 1. Click *“Finish”* to create your experiment. It can take 10-15 minutes to create the experiment. When the green tab says “Your experiment is ready!”, and the state says “ready”, your experiment can be accessed via the OpenStack dashboard. You will also receive an e-mail when your experiment is ready. **Do not try to access the OpenStack dashboard until the experiment state says Ready.**
    2. **Operational issues:** It is possible that, sometimes, your experiment cannot be loaded properly. If there is an **x** on any of your nodes, as seen in Figure 6 (below), you need to reload this node to fix the problem. Once you see the green check mark on your nodes, the experiment is fully loaded.

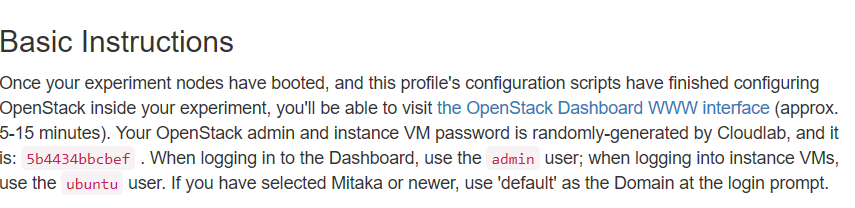
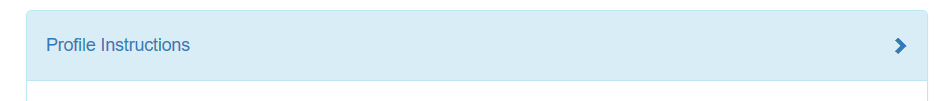
*Figure 5*



*Figure 6*

* + 1. **Note:** By default, your experiment will automatically expire in 16 hours. However, you can extend this when needed. Click *“Extend”* and drag the slider to the amount of days you wish to extend the experiment by. You will be required to enter a reason of why you want to extend your experiment. Most requests are granted automatically depending on how many days you wish to extend it by.

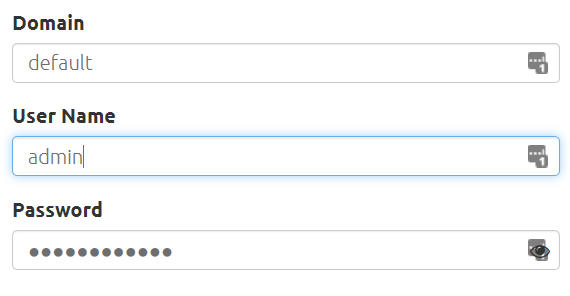
*Figure 7*

* 1. *Uploading Virtual Machine images to Experiment*
     1. Opening the OpenStack Dashboard
        1. **Extend the blue box entitled *“Profile Instructions”.* You will see a link to the OpenStack dashboard, and a randomly generated password you will need to login to the dashboard. You can copy this password to your clipboard for future usage.

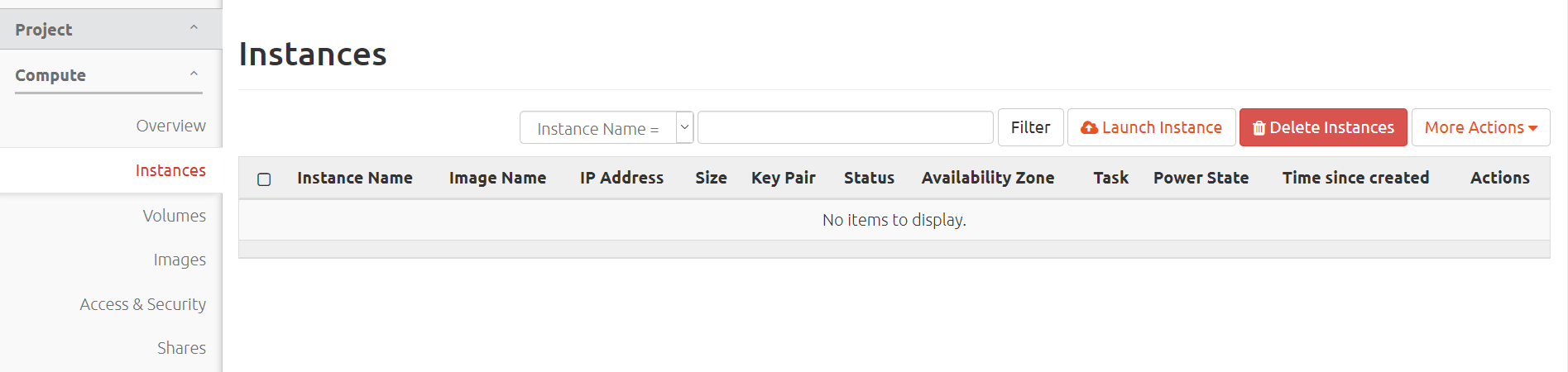
*Figures 8 and 9*

2. Click the link to open the OpenStack Dashboard.

* + - 1. The Domain is *“default”*. The user *is “admin”.* The password is the randomly generated password generated for your experiment.. (See Figure 9) After entering this information, click *“Connect”.*

**

*Figure 10*

* + - 1. **The first page that will open is the list of instances. Because this experiment is newly created, there will be no instances listed here. On the left side, click *“Images*”. (Shown below)

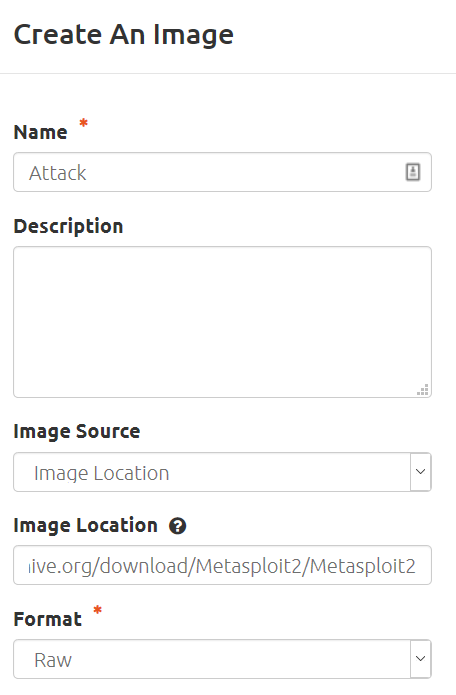
*Figure 11*

* + - 1. On the Images page, click *“Create Image”.*

*Figure 12*

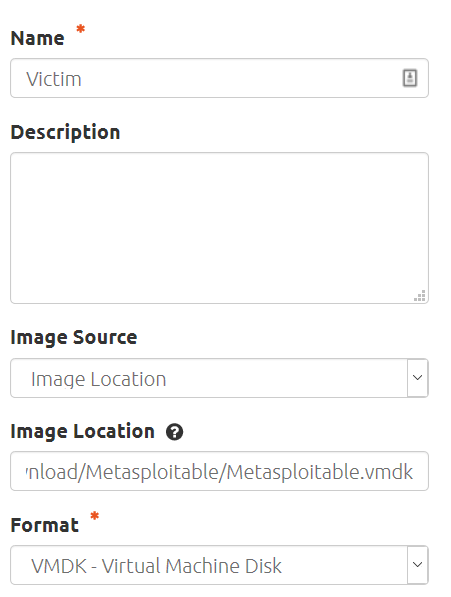
**

* + - 1. We will upload the Attack image first since it is the largest. For the name, type *“Attack”* without quotations. **(It is very important you give it this exact name, or else the experiment will not upload properly in later steps.)** For image location, copy and paste this address: **[https://archive.org/download/Metasploit2/Metasploit2](https://archive.org/download/Metasploit2/Metasploit2" \t "_blank)** For image format, select “Raw”.

**

*Figure 13*

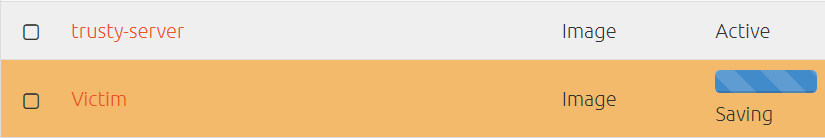
* + - 1. Once you have confirmed these settings are correct, scroll down and click *“Create Image”.*
      2. Now we will import the Victim image. Click *“Create Image”* once more.
      3. For the name, type *“Victim”* without the quotations. **(It is very important you give it this exact name, or else the experiment will not upload properly in later steps.)** For image location, copy and paste this address: **[https://archive.org/download/Metasploitable/Metasploitable.vmdk](https://archive.org/download/Metasploitable/Metasploitable.vmdk" \t "_blank)** Confirm that VMDK is the image format selected.



*Figure 14*

* + - 1. Once you have confirmed these settings are correct, scroll down and click *“Create Image”.*

*Figure 13*

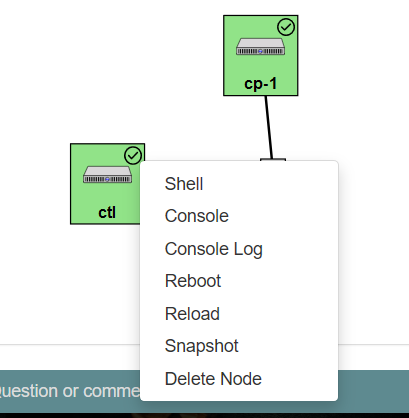
* + - 1. **It will take a while for these images to upload. When the image status says *“Saving”*, the image has not finished uploading. When the image status says *“Active”,* the image is finished uploading. **You MUST wait until both images show “Active” as their status before you can move onto Section *d.***

*Figure 15*

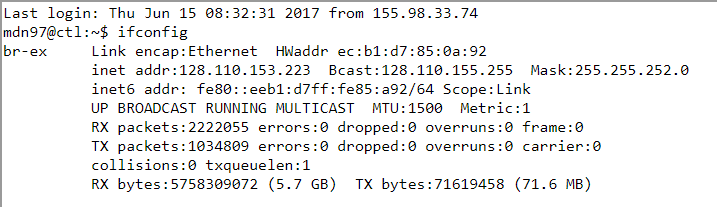
Figure 15 shows an image that has finished uploading (trusty-server) and an image that has not finished uploading (Victim).

* 1. *Importing Experiment Profile using Amazon EC2*
     1. Required software:
        1. MobaXterm (<http://mobaxterm.mobatek.net/>)
     2. Steps
        1. You will need to know the IP address of the control node for your CloudLab experiment, and the password to access the control node. The randomly generated password is already provided to you, as illustrated in Figure 9.

To get the IP address, you can use the shell on CloudLab. To find the IP, scroll down to the bottom of your experiment page on CloudLab to the topology of the nodes. Click on the check mark next to the **ctl** node and choose **Shell** from the list of options.



*Figure 16*

Once the shell loads, type **ifconfig.** The first IP that is listed is the IP of the ctl node. **(After inet addr:)**

*Figure 17*

To avoid typing all the information needed to connect to your experiment using the Tester Script, copy the text below **replacing the IP and password with the IP and password for your experiment.**

**Experiment IP**

1

http://ctl:5000/v3

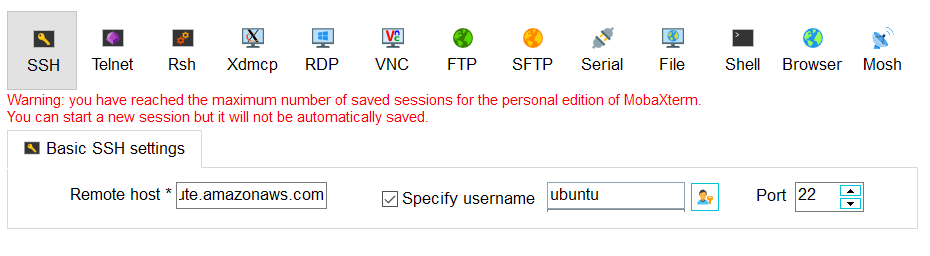
RegionOne

admin

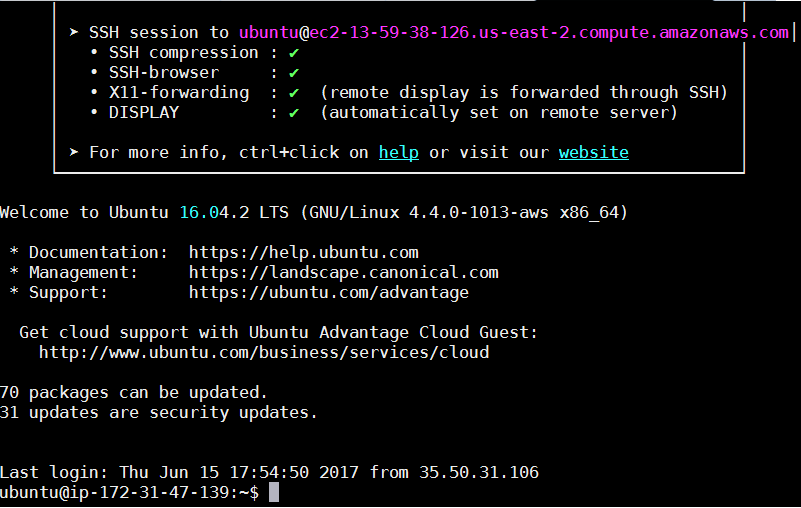
admin

**Experiment password**

* + 1. An Amazon EC2 image has already been created for you, but you will need to import the private key to access this image. The XML document to upload the experiment and scripts to upload the XML document are already on this Amazon EC2 image. Download the private key from **https://drive.google.com/open?id=0B-Nx8At6Cvo6Q0hoOER5OXg5akk**
    2. The public DNS to access this EC2 VM is **ec2-13-59-38-126.us-east-2.compute.amazonaws.com.** Copy this address.
    3. Launch MobaXterm. Click Session, and then SSH. You will be presented with basic SSH settings. In the box labeled *“Remote Host”,* enter **ec2-13-59-38-126.us-east-2.compute.amazonaws.com.** Ensure that **22** is the port being used. Check the box that says *“Specify username”,* and enter ***“ubuntu”***in the box. After you have entered this information, click “*Advanced SSH settings”.*



*Figure 18*

* + 1. Check the box that says *“Use private key”,* and import the private key that you saved to your computer earlier. Click “OK” after you have done this. If successful, you will be presented with the screen present in Figure 19.

*Figure 19*

* + 1. Enter the following command: **sudo python3 TesterScript.py**. This will launch the Tester Script, the Python program used to import the experiment framework.
    2. Copy and paste the text from earlier (where you input your experiment IP and password) into the script.

**Experiment IP**

1

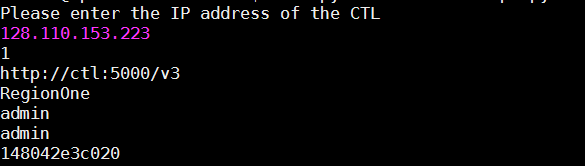
http://ctl:5000/v3

RegionOne

admin

admin

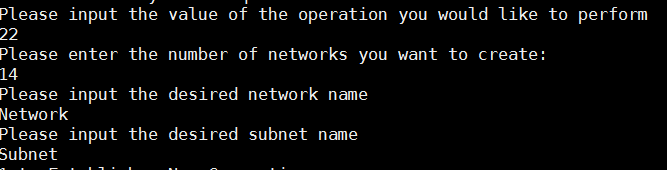
**Experiment password**

****

*Figure 20*

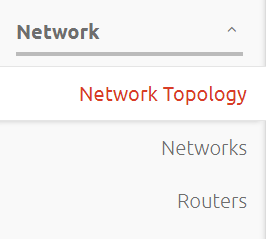
If the connection is established, a list of virtual machine images in that experiment will be listed, and you will be presented with an options menu. **You should see the Attack and Victim images listed. If you do not see these image names listed, confirm that they have finished uploading before continuing.**

* + 1. The only option you will need to use is Option 20. Enter **20** and press enter. You will be prompted for the number of networks you want to create, the name for the network, and the name for the subnets. You can simply choose **“Network”** and **“Subnet”** for the names. (Each network, subnet, and machine will be appended with a number for identification purposes.) If you want to have **14** pairs of attack and victim machines, you will enter **14** for the number of networks. Give the script a minute to completely run before proceeding.

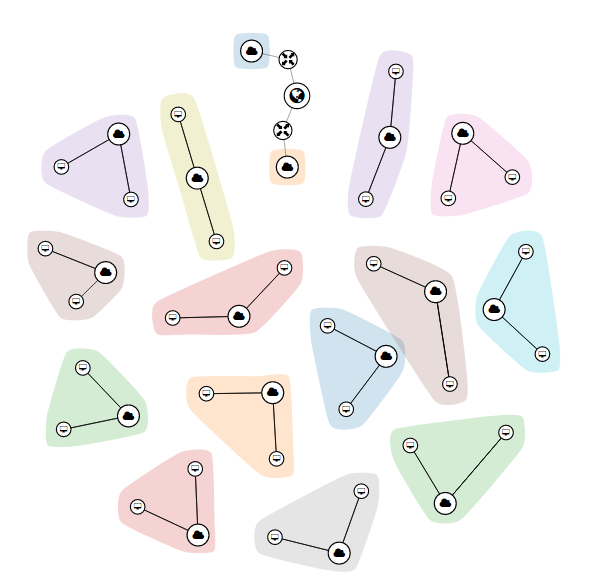


*Figure 21*

* + 1. Switch back to the OpenStack Dashboard, and on the left side, expand the **“Network”** tab and click on **“Network Topology”.**

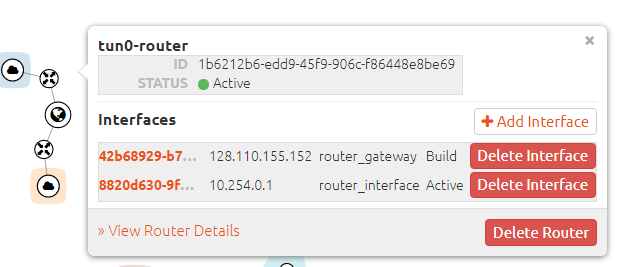


*Figure 22*



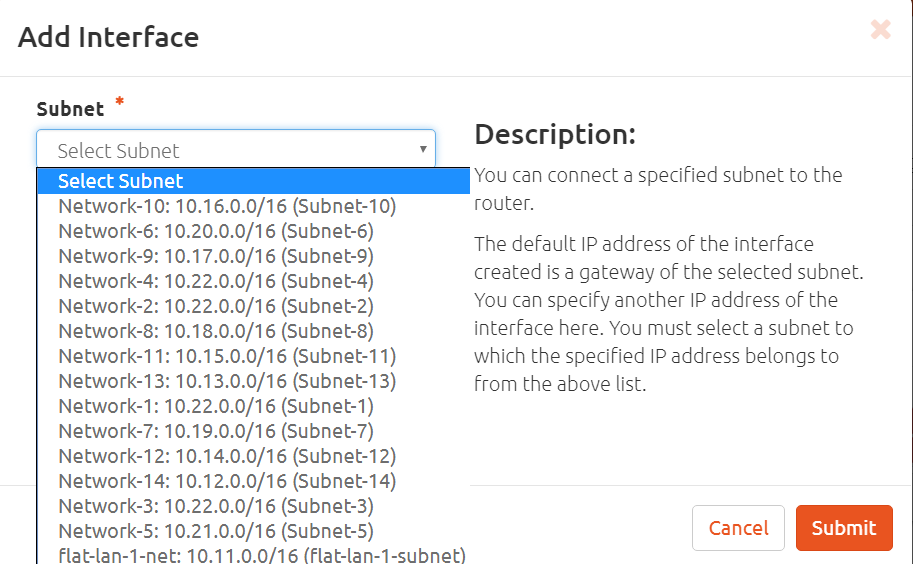
*Figure 23*

* + 1. You will see all the networks with their respective Victim and Attack machines. Currently, they are not connected to a router, and you will need to connect them before proceeding. You may choose either of the default routers. Click on one of the routers, and choose **“Add Interface”.**



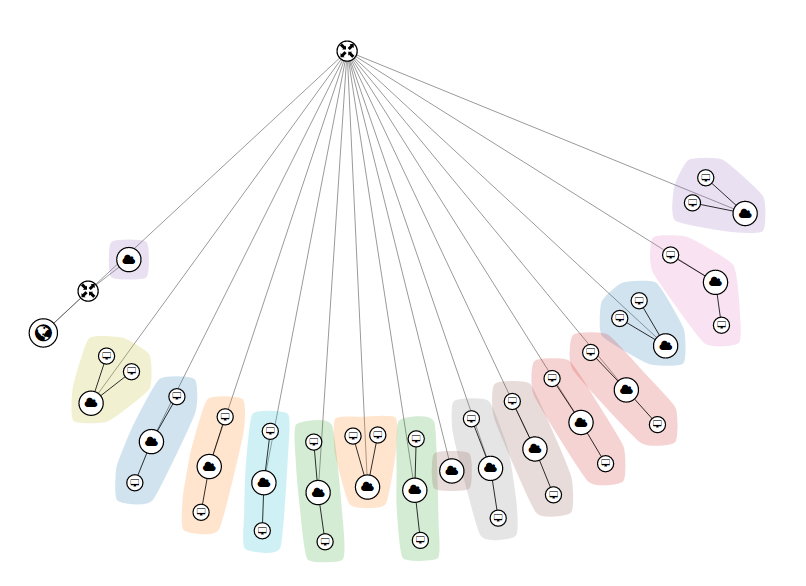
*Figure 24*

* + 1. A new window will appear. Click on the arrow next to **“Select Subnet”.** You will be presented with a list of all the networks. Click on the first network shown, and then click **“Submit”.** The network you chose will be connected to the router.



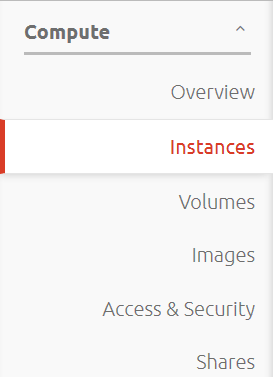
*Figure 25*

* + 1. Repeat Steps XI and XII for each network in the experiment. This process should not take more than a few minutes. After all the networks have been connected to a router, it will look similar to the figure below.



*Figure 26*

* + 1. When you ran the previous menu item, it also automatically created **14** floating IP addresses. You will need to associate each Attack machine with a floating IP. On the left side of the OpenStack Dashboard, expand the **“Compute”** tab then click on **“Instances”.**

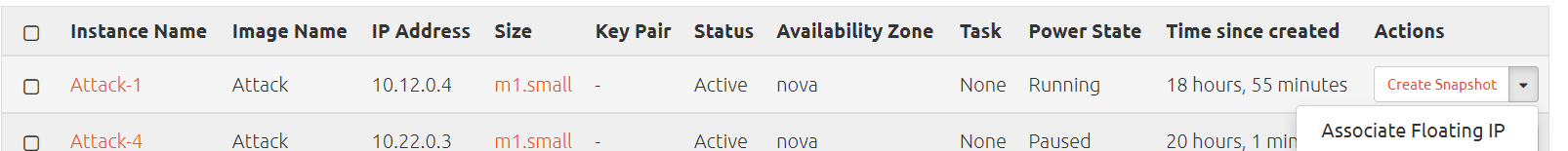


*Figure 27*

* + 1. At the top of the Instances page, there will be a search bar. Enter “**Attack”** and click **“Filter”** to show only Attack virtual machines.

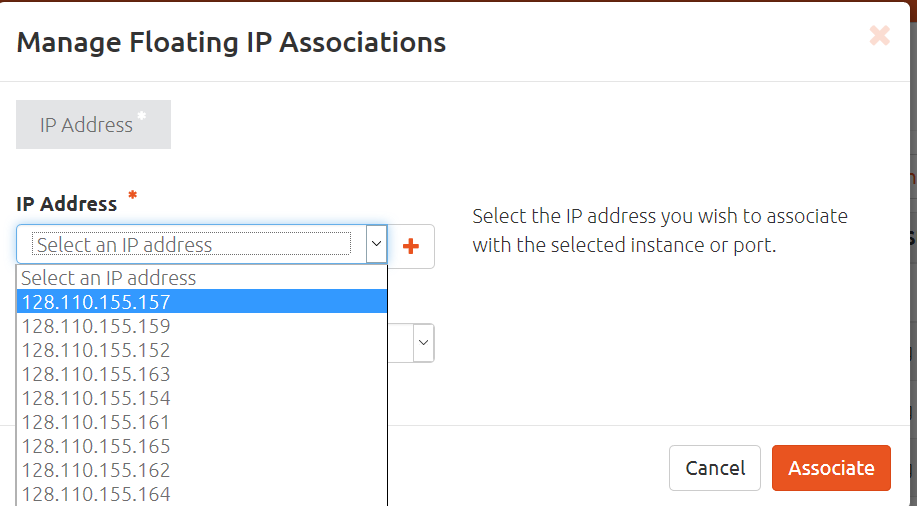


*Figure 28*

* + 1. Next to the first instance, click on the **arrow** next to **“Create Snapshot”**. A drop-down menu will appear. On this menu, click **“Associate Floating IP”.**

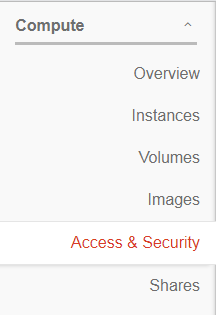
*Figure 29*

* + 1. A new window will appear. Click on “**Select an IP address”** and click the first IP in the list. Once you have selected an IP, click **“Associate”.**



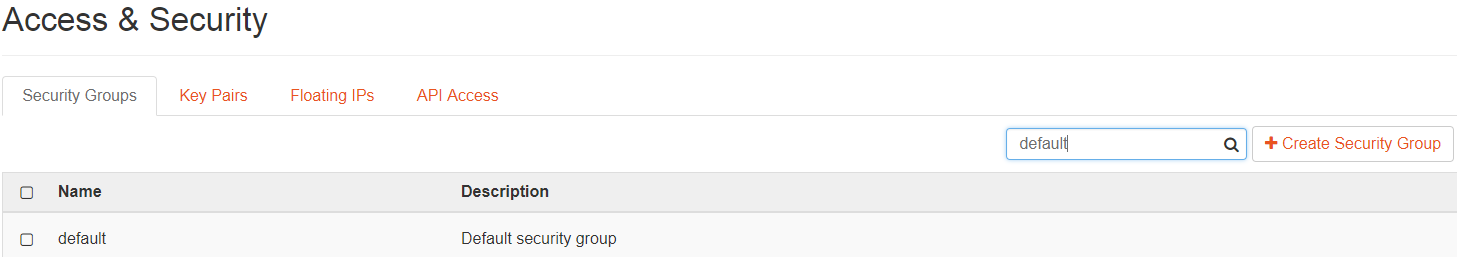
*Figure 30*

* + 1. Repeat steps xvi and xvii for each Attack virtual machine until each Attack virtual machine has a floating IP address associated with it.
    2. After floating IP addresses have been associated, a security group must be created so that each instance can be accessed from the internet.
    3. Click on the **“Access and Security”** tab under the **“Compute”** tab in the OpenStack Dashboard.



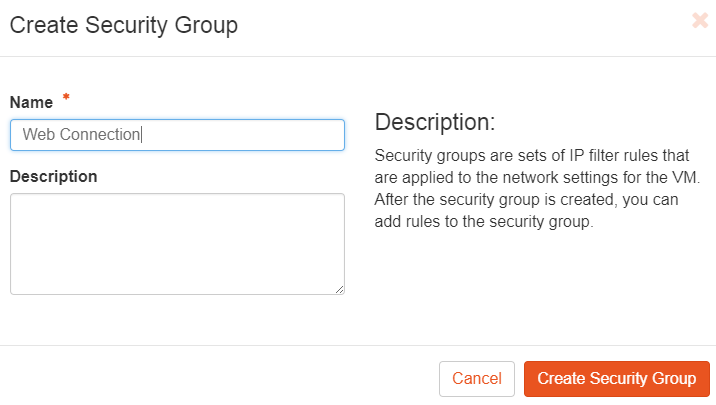
*Figure 31*

* + 1. Navigate to and click a button that says **“Create Security Group”** on the right side of the page.



*Figure 32*

* + 1. Fill in the name of your choice for the security group, then click **“Create Security Group”**.



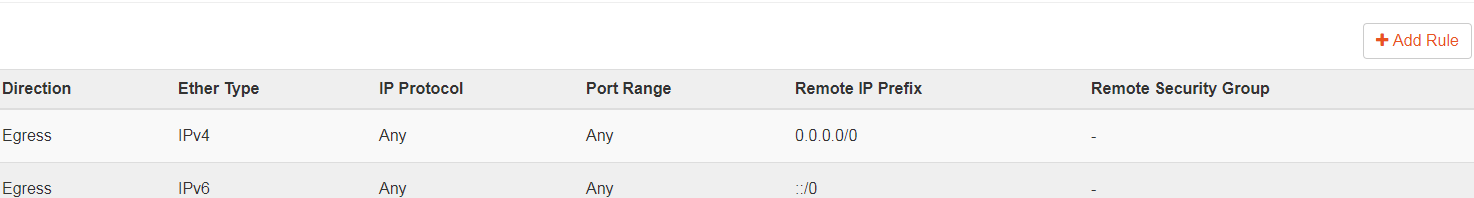
*Figure 33*

* + 1. After the security group is created, you will see it displayed on the page. On the right side of the security group, there is a button called **“Manage Rules”**, click that button.

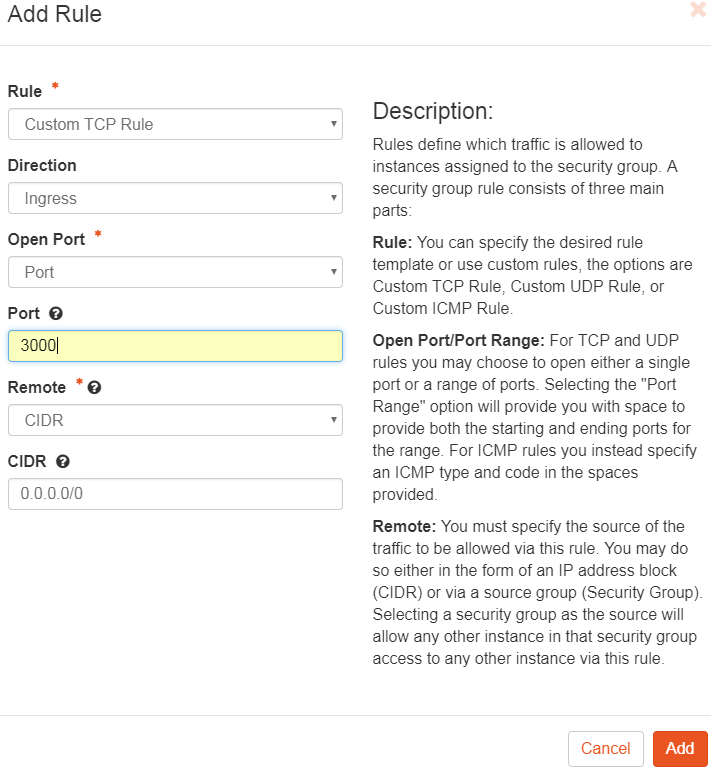
*Figure 34*

* + 1. The attack instance uses a software called “Wetty” to allow the user to interact with the terminal online. The TCP port that is used is 3000. Under manage rules, you can create a rule to allow the instances to access that port using the **“Add Rule”** button on the right side of the page.



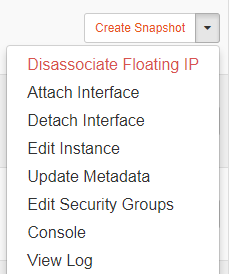
*Figure 35*

* + 1. After the **“Add Rule”** button is clicked, you will be relocated to a page which allows you to set your rules. In this case, the only line you will have to edit is the **“Port Number”** line, all other parameters are already set. Here, you will set the port to 3000 and click the **“Add”** button.



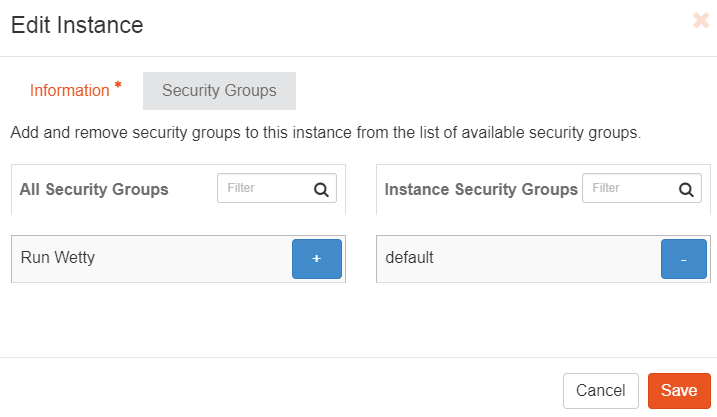
*Figure 36*

* + 1. After the rule is created, it must be applied to each Attack instance containing a floating IP address. Click the arrow next to the **“Create Snapshot”** button. A drop down menu will appear and you will need to click on the **“Edit Security Groups”** option.



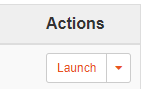
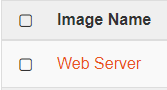
*Figure 37*

* + 1. A new window will appear showing that the **“default”** security rule is already applied to the instance. Click the **“+”** button on the rule that you have created, in this case, the rule is called “Run Wetty”. This will apply the new security rule to the instance.



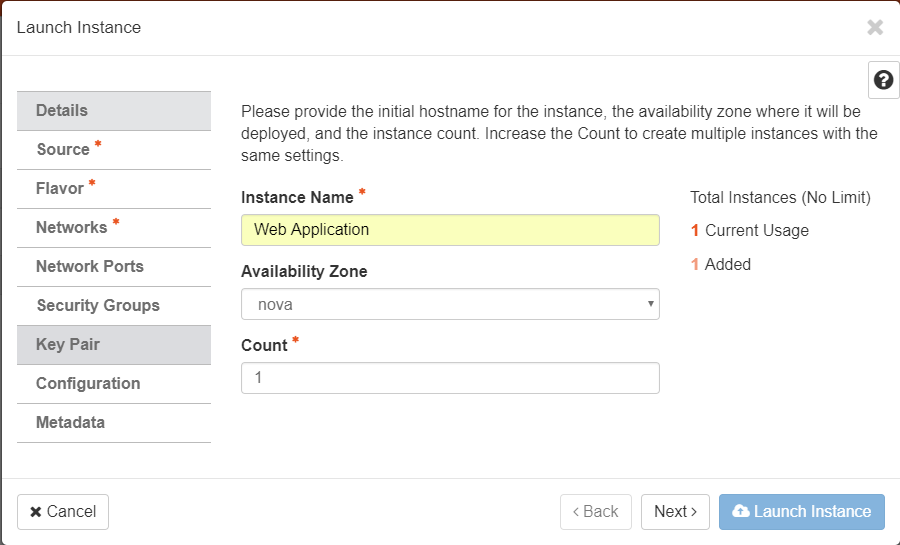
*Figure 38*

* + 1. The next step is to upload an image of a Web Server to an experiment, the link can be found here: When the image is uploaded, manually create an instance from it and allocate a floating IP address.
    2. To manually upload an instance from an image, click the launch button next to the name of the image. In this case the name of the image is Web Server. Click the launch button on the right side of the image name.

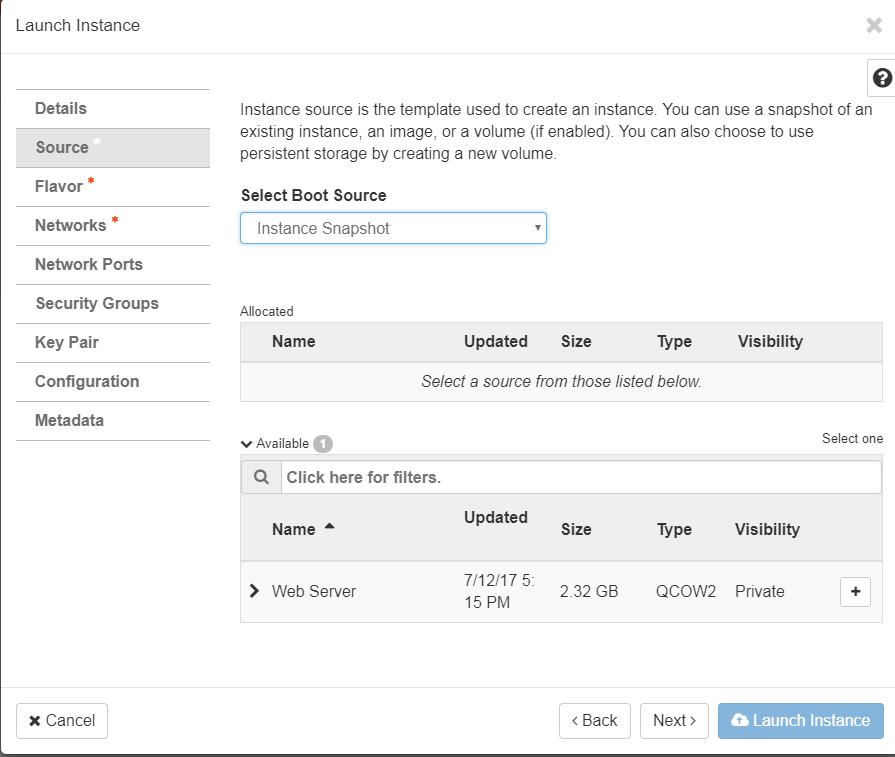


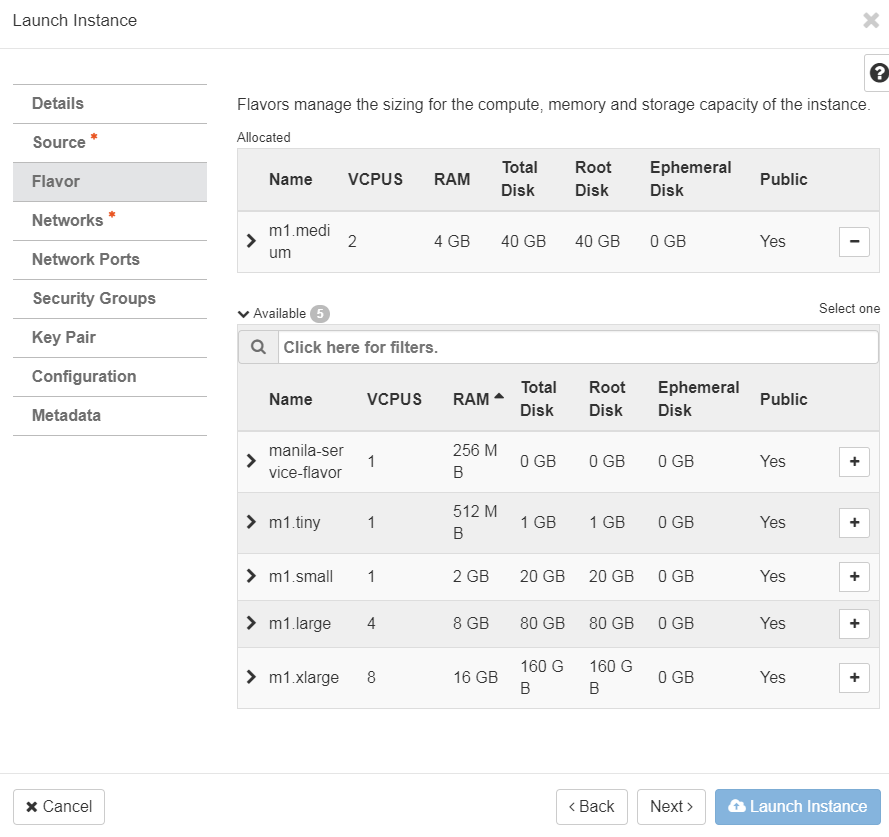
*Figure 39*

* + 1. You will get to a pop-up window where you need to set the parameters of your virtual machine including the name, image, flavor, and network.

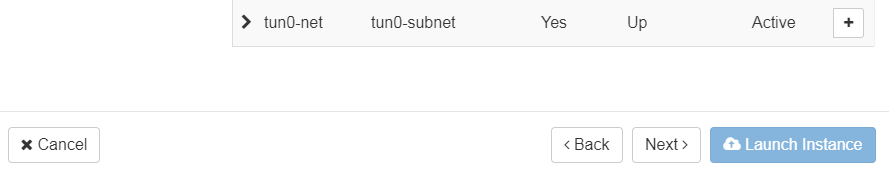


*Figure 40a*





*Figure 40b-c*



*Figure 40d*

* + 1. Connect to this instance using “Moba X-term” or any SSH client. When prompted for a password, please enter the following: **“7770ddc5d198”**. This image contains both the Master Script and Tester Script to allow you to connect to the control node of the experiment as well as a MySQL database which will store the IP addresses from the experiment.
    2. To obtain all of the IP addresses and machines from the experiment, run the Tester Script and copy and paste the text from earlier.

**Experiment IP**

1

http://ctl:5000/v3

RegionOne

admin

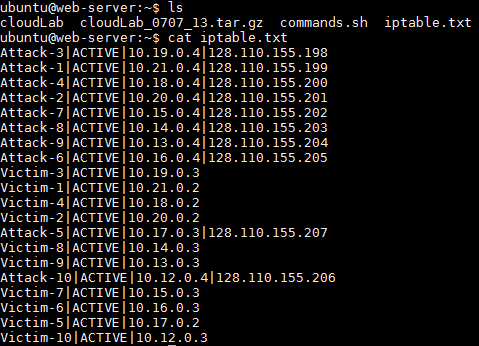
admin

**Experiment password**

* + 1. When the connection is made to the experiment, select option 7. Option 7 will display the all instances of the experiment including the including the public and private IP address for each instance. 

*Figure 41*

* + 1. The output will then be redirected to a file called **“iptable.txt”.**



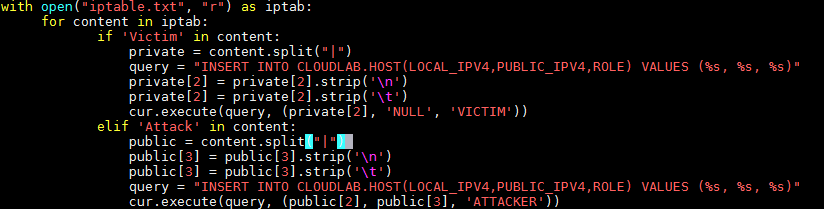
*Figure 42*

* + 1. Run option 21 in the Tester Script using the same IP address and password as above. This option will connect to the database in the server.



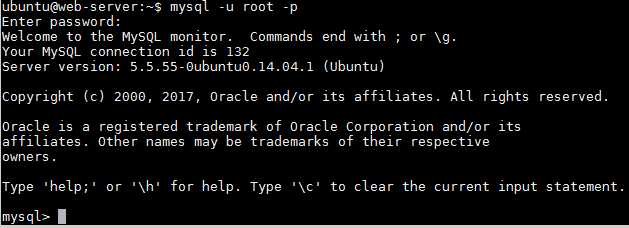
*Figure 43*

* + 1. After the connection is made, the content in the iptable.txt will be read to the **HOST** table in the database. At each character “|”, the content will be split into a list for each line. Some of the text will be altered; for example, if a list position contains the word “Attack”, the position of the list will be changed to “ATTACKER”. The same thing will happen if a list position contained the word “Victim”, it will be changed to “VICTIM”. Using queries, the text from the file will be inserted into the database.



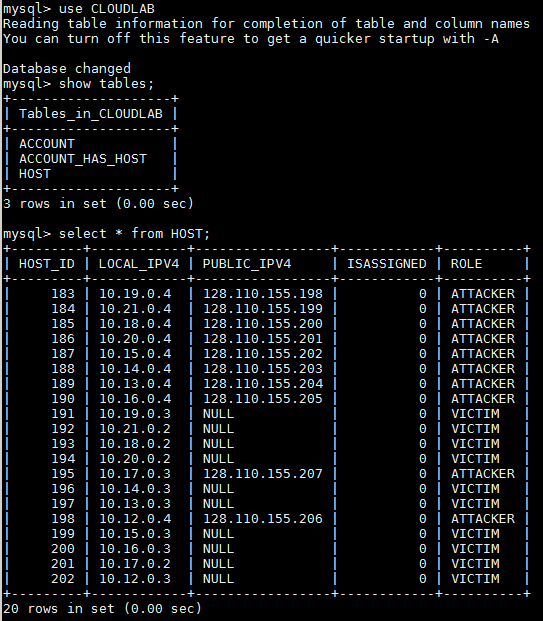
*Figure 44*

* + 1. All of the information regarding the IP addresses is now in the database. To view this information, type the command **mysql –u root –p**. You will then be prompted for a password, the password is **“root”.**



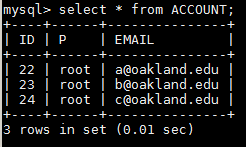
*Figure 45*

* + 1. Now that you are in MySQL, you must select the correct database. Type the command **use CLOUDLAB**. You are now connected to the database. By using the command **show tables;** you will see that there are three tables. The one that contains the information about the IP addresses is called **“HOST”**. To view this database, use the command **select \* from HOST;** the table will now be displayed and you can see all of the contents.

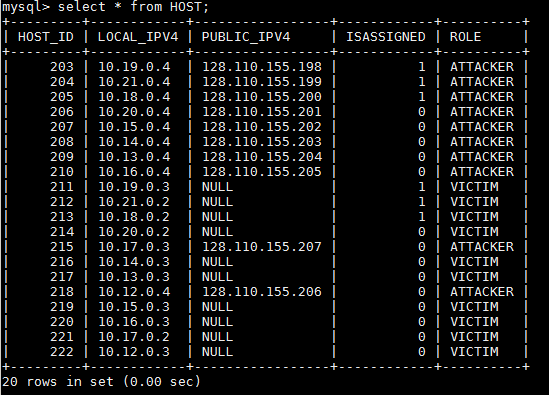


*Figure 46*

* + 1. The database is now occupied by the IP Addresses of the attack and victim machines. When a student signs up online, their email address and password will be stored to the database in the “**ACCOUNT**” table. At the same time, two “**ISASSIGNED**” attributes will be changed to a “**1**” in the “**HOST**” table to indicate that the student has been assigned a victim and attack virtual machine.

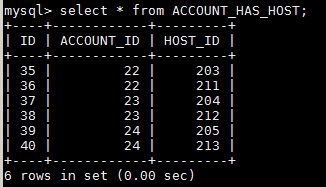


*Figure 47a*



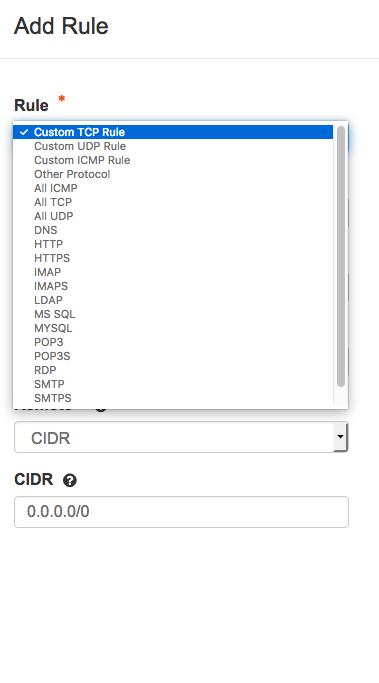
*Figure 47b*

* + 1. In the database, there is a third table called “**ACCOUNT\_HAS\_HOST**”. This table stores the primary keys of the “**HOST**” and “**ACCOUNT**” tables. This is done so that the user gets assigned the attack and victim nodes. As you can see, between the figures above and the figure below, there is a one to many relationship between the user and the host.



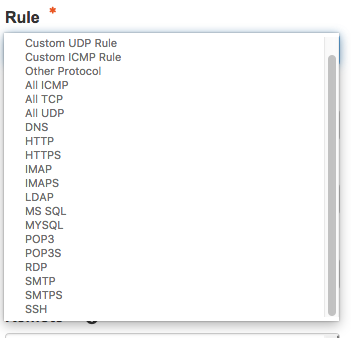
*Figure 47c*

* + 1. Before students can begin using the web server to obtain IP address that they can use, there are a couple of security rules that will need to be put in place. Refer to steps xx-xxv to create a security group.
    2. In your new rule, port 80 will need to be enabled so that the instance can make a connection to the internet via http. As shown in the following figure, under the “**Add Rule**” pop up window, there are many different options. To enable http, select the “**HTTP**” menu item and press add in the lower right hand corner.



*Figure 48*

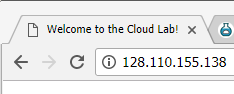
* + 1. The same thing must be done to continue to allow SSH access. Click the button to add another rule and in the pop up window, bring up the drop down menu for “**Rule**” and scroll to the bottom of the drop down menu. Select “**SSH**” and enable it by clicking “**Add**” in the lower right hand corner.



*Figure 49*

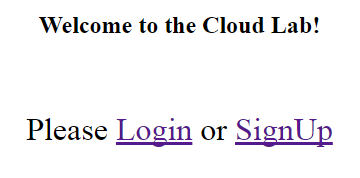
* + 1. Repeat step xxvii to add your new rule to the Web Server Virtual Machine.
    2. The web client should now be ready for students to use.

1. **Running the Experiment (Student)**
   * 1. To begin running an experiment, please use your browser and enter the floating IP address of the web server in your search bar.



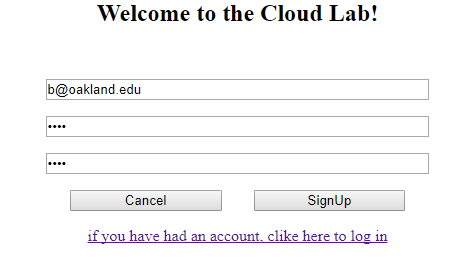
*Figure 50*

* + 1. You will then be directed to a page containing options to login or sign up. Since you are a new user, you will be required to sign up.



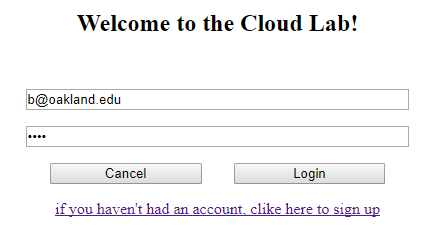
*Figure 51*

* + 1. After hitting the sign up button, you will be required to enter your email address as well as designate a password for your account.

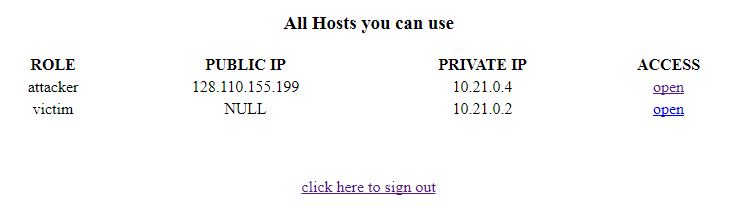


*Figure 52*

* + 1. After the sign up button is clicked, you will be redirected to the login page. At the login page, you will enter the credentials that you created during the sign up process.

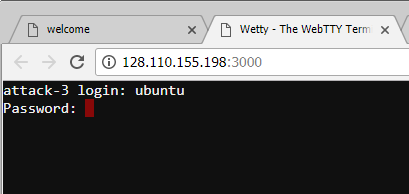


*Figure 53*

* + 1. After the login button is clicked, you will be redirected to a page containing some information: The Attack Node’s public and private IP address as well as the private IP address of the victim machine.

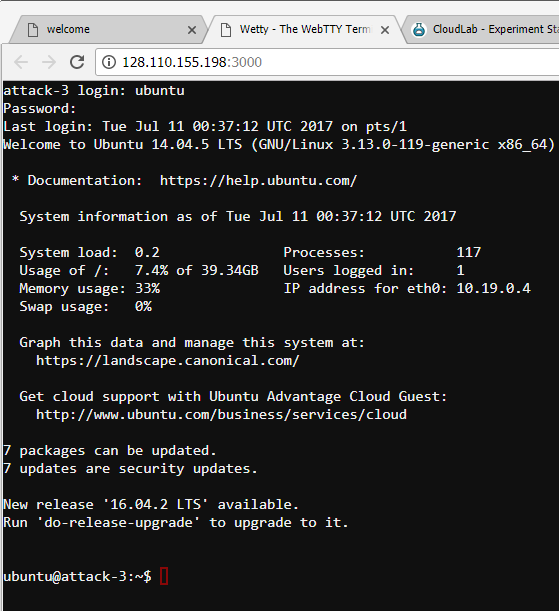
*Figure 54*

* + 1. In order to access the attack machine, which is the machine with the only public IP address, you must click the link to open the VM.
    2. You will be directed to a web terminal of the attacker node. The link address will reflect that with the IP address of the attacker.



*Figure 55*

* + 1. You will be prompted with a username and password. Enter **“ubuntu”** as the username and “**ce17e6285b09**” as the password (without quotations) (The attack image used the password for a previously generated experiment. This password will always stay the same.) You can choose to save this password so that you do not have to enter it again if you get disconnected. If successful, you will see the login screen shown in Figure 23:



*Figure 56*

* + 1. You need to create the database Metasploit uses before you can run it. PostgreSQL is the database server used to create this. Before you can create the PostgreSQL database, you need to run this command to create the database for logs: **mkdir -p ~/.msf4/logs/**
    2. After running the previous command, enter **sudo service postgresql start**. To verify that the service is running, enter **service postgresql status**.

1. Enter **sudo** **msfconsole** to launch Metasploit.

The following exploits come from hdmoore’s documentation found on Rapid7’s website. We thank hdmoore for this documentation.[4]

**Exploit 1: UnrealIRCD IRC Daemon**

**Exploit information:** <https://www.rapid7.com/db/modules/exploit/unix/irc/unreal_ircd_3281_backdoor> [5]

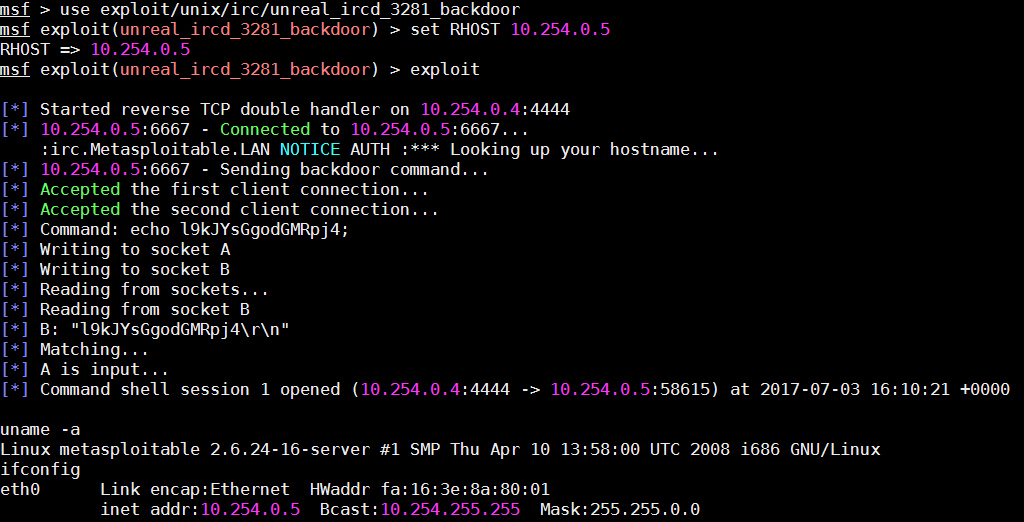
1. Type **use exploit/unix/irc/unreal\_ircd\_3281\_backdoor into the Metasploit console and press enter.**
2. **Find the IP address for your victim machine. This should be present in the web client.**
3. **Type set RHOST followed by the IP address of your victim machine, as seen in the figure below. Press enter.**
4. **Type exploit and press enter.**
5. To verify the exploit was successful, you will run some system identification commands. Type **uname -a.** You should see that **Metasploitable** will be the image name shown. Then, type **ifconfig.**You should see the IP address for the Victim machine shown next to **inet addr.** See the figure below.
6. When you are ready to quit, press **CTRL + C** and enter **y** to exit and enter back into the Metasploit console.
7. Type **back** and press enter to quit using the current exploit.

**Exploit 2: Distccd**

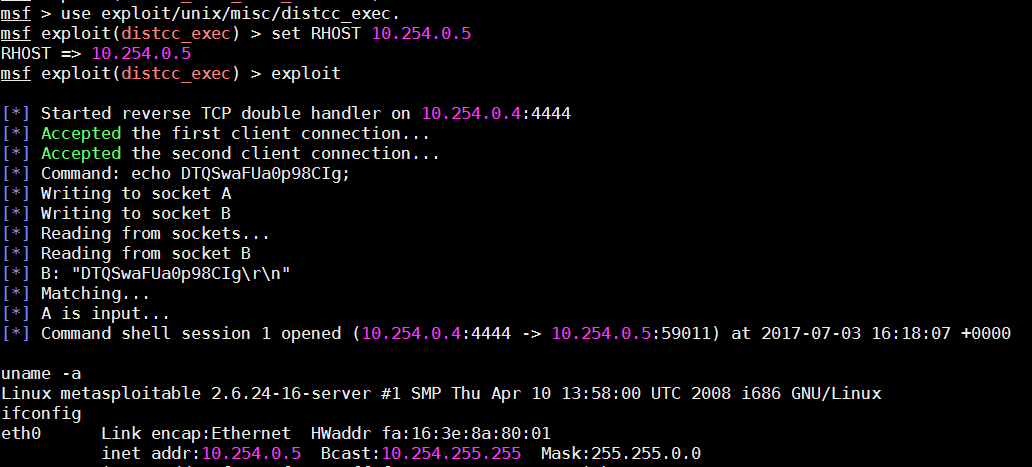
**Exploit information:** <https://www.rapid7.com/db/modules/exploit/unix/misc/distcc_exec> [6]

1. Type **use exploit/unix/misc/distcc\_exec.**
2. Find the IP address for your victim machine. This should be present in the web client.
3. Type **set RHOST** followed by the IP address of your victim machine, then press enter.
4. Type **exploit** and press enter.
5. To verify the exploit was successful, you will run some system identification commands. Type **uname -a.** You should see that **Metasploitable** will be

the image name shown. Then, type **ifconfig.**You should see the IP address for the Victim machine shown next to **inet addr.** See the figure below.

1. When you are ready to quit, press **CTRL + C** and enter **y** to exit and enter back into the Metasploit console.
2. Type **back** and press enter to quit using the current exploit.

*Figure 56*

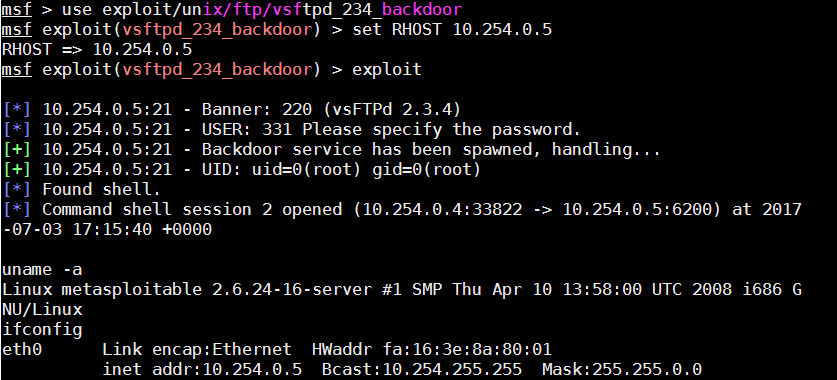
****

*Figure 57*

**Exploit 3: vstfpd Backdoor**

**Exploit information:**

<https://www.rapid7.com/db/modules/exploit/unix/ftp/vsftpd_234_backdoor> [7]

1. Type **use exploit/unix/ftp/vsftpd\_234\_backdoor.**
2. Find the IP address for your victim machine. This should be present in the web client.
3. Type **set RHOST** followed by the IP address of your victim machine, then press enter.
4. Type **exploit** and press enter.
5. ****To verify the exploit was successful, you will run some system identification commands. Type **uname -a.** You should see that **Metasploitable** will be the image name shown. Then, type **ifconfig.**You should see the IP address for the Victim machine shown next to **inet addr.** See the figure below.
6. When you are ready to quit, press **CTRL + C** and enter **y** to exit and enter back into the Metasploit console.

*Figure 58*

1. Type **back** and press enter to quit using the current exploit.

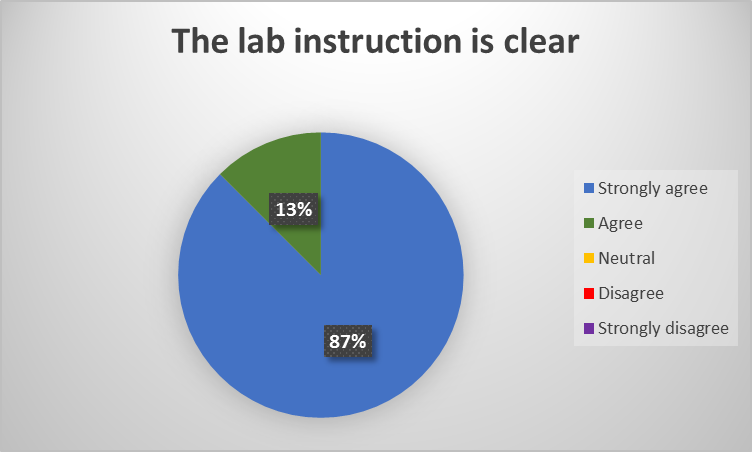
More exploits can be found on Metasploit’s exploit database, found at <https://www.rapid7.com/db/modules/>. [8]

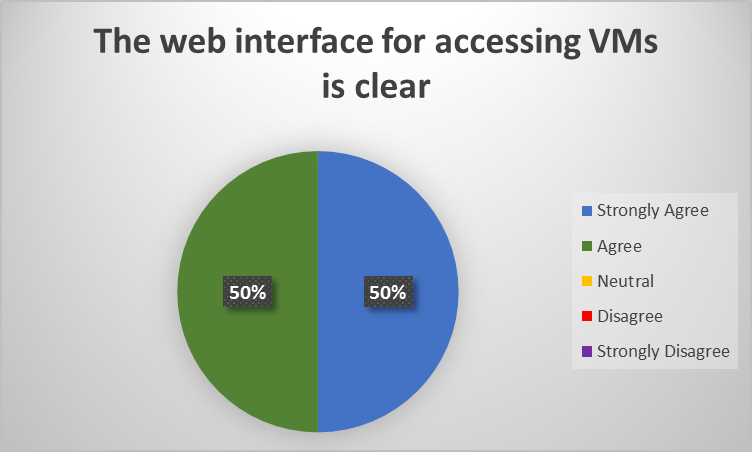
1. **Conclusion**

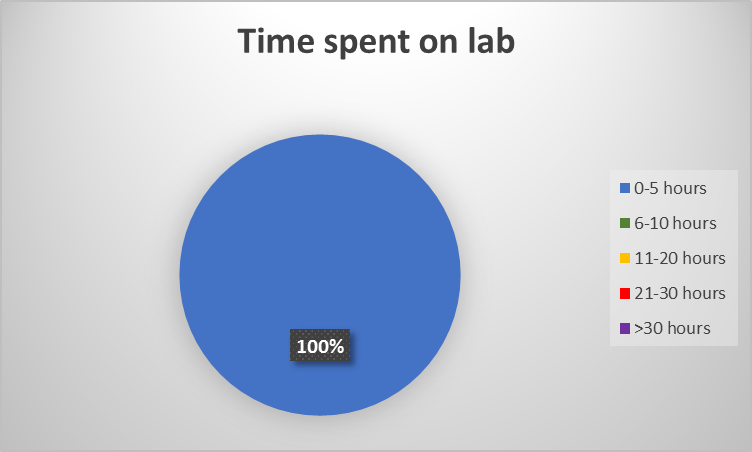
We tested our implementation with ten REU (Research Experience for Undergraduates) and received overall high satisfaction with our work. We were able to create a virtual laboratory using virtual images and a web client with an in-browser SSH terminal. By doing this, we were able to streamline the process of creating new laboratory environments, and created a framework for future experiments. After the test was completed, we conducted a voluntary survey which provided us with valuable feedback on students experience using the virtual laboratory. The data from nine multiple choice questions was compiled and plotted for each question. The survey also contained two open ended questions where students were allowed to give their feedback and opinion on the virtual laboratory.

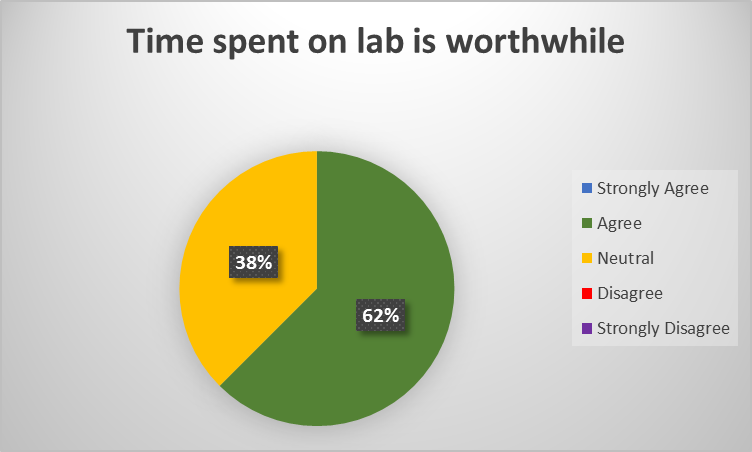
1.

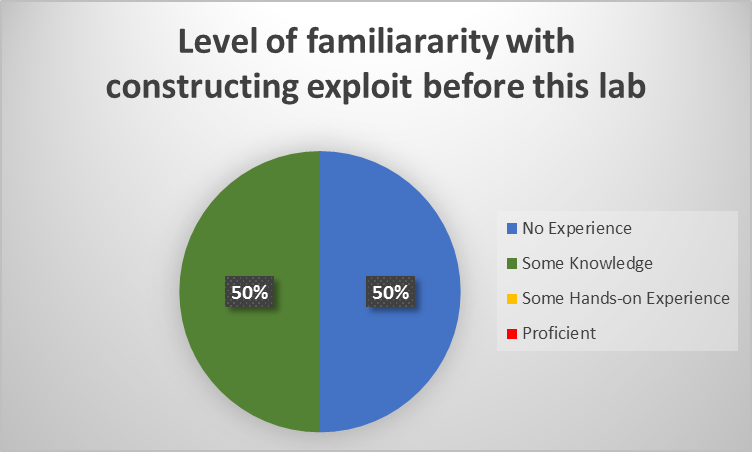


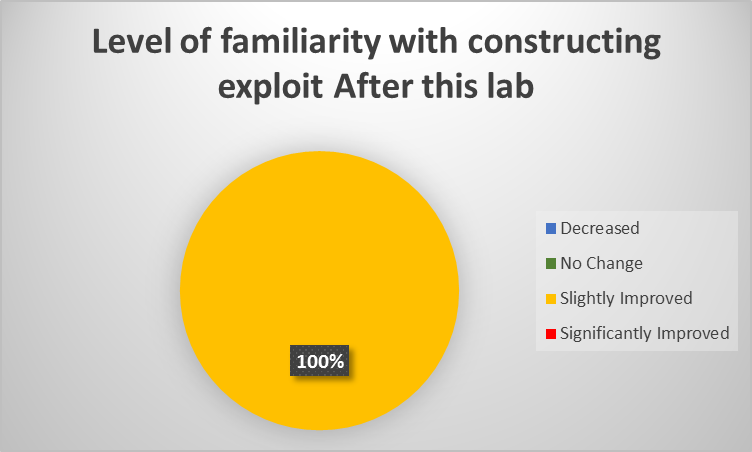


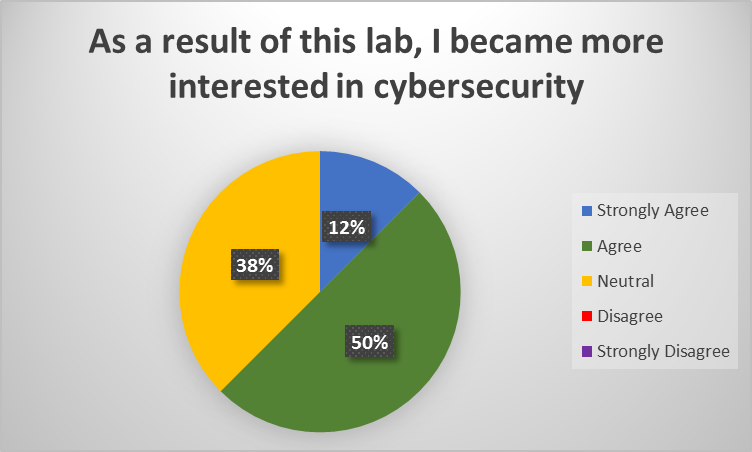


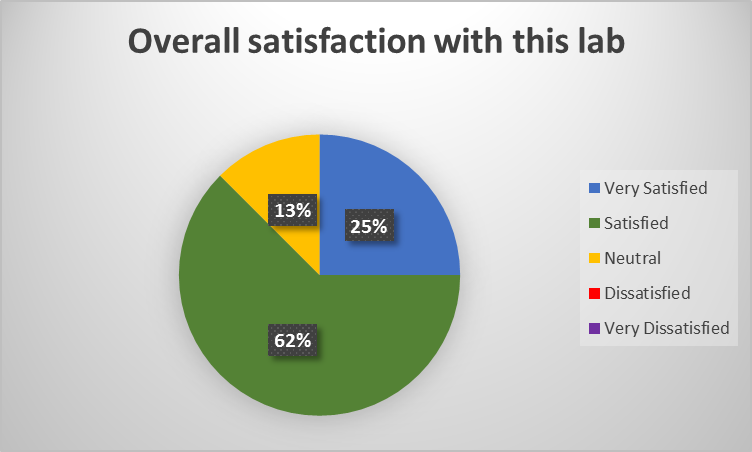












1. What part of this lab could be improved?

Login screen typos. If you type wrong password, it goes to weird page, should have a warning pop-up.

More participation required

Providing more information on what is being done when running the exploits. What vulnerabilities are being exploited?

1. Which part of this virtual lab environment could be improved?

Spell Check

Email authentication

If possible, create required directories before experiment to reduce setup time.

Initial authentication, when differing passwords are input when creating an account (into the password and password verification fields), neither password will work for logging in, and no error is presented.

1. **References**

**[1]** Mobatek. (n.d.). *MobaXterm free Xserver and tabbed SSH client for Windows.* Retrieved from http://mobaxterm.mobatek.net/

**[2]** rapid7user. (n.d.). *Metasploitable.* Retrieved from https://sourceforge.net/projects/metasploitable/

**[3]** Rapid7. (n.d.). *Penetration Testing Software, Pen Testing Security*. Retrieved from https://www.metasploit.com/

**[4]** Hdmoore. (2017, April 20). *Metasploitable 2 Exploitability Guide.* Retrieved from https://community.rapid7.com/docs/DOC-1875

**[5]** Rapid7. (n.d) *Vulnerability & Exploit Database.* (n.d.). Retrieved from https://www.rapid7.com/db/modules/exploit/unix/irc/unreal\_ircd\_3281\_backdoor

**[6]** Rapid7. (n.d) *Vulnerability & Exploit Database.* (n.d.). Retrieved from https://www.rapid7.com/db/modules/exploit/unix/misc/distcc\_exec

**[7]** Rapid7. (n.d) *Vulnerability & Exploit Database.* (n.d.). Retrieved from https://www.rapid7.com/db/modules/exploit/unix/ftp/vsftpd\_234\_backdoor

**[8]** Rapid7. (n.d.). *Exploit Database.* Retrieved from https://www.rapid7.com/db/modules/

1. The reason is that when a new OpenStack instance is created, the UUIDs of each artifacts are unique and cannot be reused again. Therefore, the old IDs cannot be used again during reconstruction [↑](#footnote-ref-1)
2. Paramiko package - http://www.paramiko.org/ [↑](#footnote-ref-2)
3. [↑](#footnote-ref-3)
4. [Caution]: Large images might cause the memory leak while mapping the file into the memory. [↑](#footnote-ref-4)
5. Keystone service is the identity service that OpenStack uses for the authentication and authorization an instance. When our program calls the authentication API to establish a connection, our program interacts with the Keystone service. This is similar to the process that we interact with the Neutron service when we modify network parameters. More information about the Keystone service can be found from <https://docs.openstack.org/developer/keystone/> [↑](#footnote-ref-5)
6. Every time you start a new OpenStack instance, you are required to check the update information from admin-openrc.sh [↑](#footnote-ref-6)
7. https://en.wikipedia.org/wiki/Topological\_sorting [↑](#footnote-ref-7)