James Madison University

College of Science & Engineering (CISE)

Information Technology (IT)

IT 445 - Capstone Project Implementation

OpenStack Private Cloud Computing

Friday 7th November, 2025

Student: Casey Alexander	Student: Katherine Botticelli
Signature	*Signature*
Date	*Date*

Advisor:
Dr. Emil H. Salib

Acknowledgments

We would like to thank first and foremost, Dr. Emil Salib, as without him this project would not have gotten off the ground. He not only helped us with OpenStack and Docker execution, but also keeping us motivated and pushing us when we needed it.

We would also like to thank Jenny Chang for assisting in hardware needs and being quick to solve hardware issues. We would have struggled longer than necessary without having access to what felt like unlimited hardware resources.

We lastly would like to thank Kagen Roberts for pilot testing our lab instructions and giving perspective as a student with no OpenStack experience.

Contents

List of Figures

List of Tables

List of Attachments

.
Finals
Finals/Lab 1
1 /Capstone-LAB01.zip
2 /Capstone-LAB01.pdf
3 /LAB01.docx
4 /LAB01.pdf
Finals/Lab 2
5 /Capstone-LAB02.zip
6 /Capstone-LAB02.pdf
7 /LAB02.docx
8 /LAB02.pdf
Finals/Lab 3
9 /LAB03.docx
10 /LAB03.pdf
Finals/Symposium
11 /OpenStack.pdf

Todo list

1 Executive Summary

1.1 Introduction

Many Information Technology students, including us, have experienced feeling overwhelmed with the amount of hardware or software needed for an assignment. Sometimes it could be four access points or even four virtual machines, either way, it is a lot to keep track of. To go along with all these materials, you aren't able to do any of the work from home, so you have to break it down and take it home with you at the end of each day. Now, thinking from the point of view of an educator, it is clear to see the program is growing while the amount of resources is not growing quite as fast. As the number of students increase in the program, the amount of hardware needs to as well, along with testing each piece that comes in, it is near impossible. The solution to this, is an alternative in the cloud.

Cloud computing is a solution we have found to cover almost every base of the problem, turning physical access points with wired access to the internet all virtual, and ridding the need of expensive external drives to store VMs and other things. The approach we chose was with OpenStack; an open source private cloud computing platform allowing the creation of working networks with internet access, virtual machine instance storage, storage for containers, the ability to run these services rather than just store them, and best of all, access from anywhere [1]. OpenStack provides ease of use access and usage as well as low cost, requiring a one-time hardware cost for the server setup, rather than an individual cost for each student's account on other platforms.

Another considerably important piece behind our decision was the customization available within each OpenStack deployment. If a user just wants to use their project to create a network and store and run their instances, they can do so with no modification. But, to add features, there are just small amounts of alterations needed to be done for each additional service, making the platform entirely personalized to your needs. Kolla-ansible is what allows this customization [2], which is a deployment style we chose for the reasons stated. Kolla, itself, is pre-defined containers, similar to Docker containers, that ease deployment, doing a lot of the heavy lifting for you, and being able to be modified easily to match the needs and wants of a user. Ansible adds automated configuration ability that will run through the deployment, stopping at errors with messages to guide you towards a resolution, and complete a working deployment ready for further configuration and eventually usage.

Running an OpenStack private cloud deployment with customization and a low cost solves the issues regarding limited access to work as it allows access from anywhere, solves the issue with excessive hardware with the need for storage for it by automating network setups and virtual machine storage, and solves the increasing student issue as the deployment is a one-cost shareable platform able to be used by the whole department.

2 Project Objectives and Deliverables

2.1 Opportunity Statement

In the IT department we are faced with an on going issue of lab space and growing class sizes. As the classes grow bigger and need more equipment for every student it is also getting harder for students to find space in a lab to complete their labs. With the need for students to use lab computers run multiple VMs and the growing lack of lab space, we started to brain storm ideas on how to fix these issues.

- Lack of space in labs
- Finite amount of hardware to provide students with
- Inaccessible remotely
- Growing class sizes
- Excessive need of hardware for network setup
- Excessive need of VMs, difficult for a laptop to run

2.2 Proposed Solution

Our proposed solution to these ongoing issues is a cloud computing platform that would allow students to load their virtual machine instances on to their own deployment projects to be accessed on their own computers no matter the operating systems and it would not hinder their work based on available lab computers. This is also in hopes of making collaborating on projects and being able to work from any where more possible for IT students.

An OpenStack private cloud deployment using kolla-ansible is our solution to this problem. We've created a private cloud instance through OpenStack that is completely customized and low in cost. OpenStack would allow us to customize the cloud instance to what the IT students would need for that year, project, or class, and the administers of the server would have complete control of what is available in the cloud and how to students can interact within openstack, such as assignments to groups, group projects, and access to networks and instances.

This would also be more cost effective then some cloud services such as AWS where you would pay for subscriptions for each student. Instead, the cost would just be the hardware components needed to run the OpenStack server.

2.3 Project Goal

Use Cases

- Create OpenStack deployment using hardware
- Create networks (internal, external)
- Create instances
- Create containers
- Create volumes
- Create multiple users
- Give users access to their accounts
- Create projects
- Assign users to the proper projects
- Create OpenStack Deployment using VMs to enable teach-ability
- Research and test Wireshark capturing within deployment
- Allow access from anywhere, home, around campus
- Create Docker Mongo/Mongo-express lab
- Create Docker routing lab
- Create OpenStack Deployment lab
- Create lab for after deployment, users, projects
- Create glance images from scratch

The project goal is to complete the use cases listed and have an OpenStack deployment running that would be used for students to learn from and to benefit from in future labs. They would be able to access their own accounts, setup their own networks, instances, containers, all while having internet access, while the admin still has the higher control of each user.
Most of these were completed, some were only played with or looked at a little, and some were just not done entirely. Creating the deployment, both in hardware and in VM, creating networks, instances, volumes, containers, many users, projects,

were all completed. As well as creating docker mongo lab [3] and creating OpenStack deployment lab were completed. Wireshark was something we looked into a bit and never ended up really diving into it or using it for analysis. The Docker routing lab was not included or finished, as well as the lab after deployment using the users and projects, was not done. We were never able to create glance images from scratch also [4]. We feel we have done enough to display deep understanding of OpenStack although not as far as we wished to have gotten.

2.4 Objectives & Deliverables

The deliverables of this project are to have two completed labs for students to learn about cloud computing from. The labs will focus on Docker, deployment of OpenStack in a VM, and the use of a deployed platform created by students, to undeterred more of the capabilities. We have the students creating the deployment in a VM just for the ability to do the lab, because otherwise, the hardware needs for a deployment straight in OS, is far too great for each student to do. It is valuable for the students to learn what the deployment looks like, to troubleshoot it, and to mess around in it before using it for themselves in practical use. The third lab is more so just instructions than a full lab at this point, as we ran out of time to finalize this into a workable lab with questions and all the like.

Deliverables:

- Lab 1 - Docker Lab
- Lab 2 - OpenStack Deployment
- Lab 3 - Users and Administrators of OpenStack

3 Solution Design Architecture

3.1 Design Phase Semester

During our first semester of our capstone, we focused solely on Docker. We were new to Docker and the learning process was long and slow for us. We used an online video [3] to jumpstart our knowledge on the topic to go from not knowing anything to knowing how to create containers, how to bring containers up and down, how to use docker compose to treat multiple containers as one for a project, how to create our own docker images, and how to add data persistence to our setup with docker volumes. This video allowed us to create functional projects after just a few weeks learning the topics. After this, we ventured off to do some of our own work through ideas about routing and how to connect containers to each other. Following this video, we created instructions and ended up turning them into Lab 1. We learned about docker networks through the video, but not about real networking in docker. Because of this, we went on to create a container with modified routing rules and multiple subnets, allowing us to use it as a middle man between two containers with different subnets. This brought us near the end of our semester, thinking we would be going into the next with a plan to migrate current labs in the major into labs with the same concept, but using docker containers rather than virtual machines. This was not the case.

3.2 Implementation Phase Semester

Although we started this semester thinking we would be using Docker to migrate labs and make them more accessible to students, not forcing them to be in the building when working. This changed quickly when we were presented with the idea of OpenStack and took it and ran. Professor Salib gave us the option to continue in Docker, or pivot to focus on cloud computing in OpenStack. We took the latter and began devising our plan for the semester using a new platform we had no experience in. In the beginning of the semester, we did a lot of reading, research, and practice using OpenStack before ever creating a deployment. This came with many errors and much troubleshooting. When attempting to deploy OpenStack, we first tried using Linux based computers in our respective homes, which was unsuccessful quickly. We both did not know what exactly we were getting into, and were not setting up our hardware correctly for this to be successful. We moved on to working in the lab, but trying to deploy inside a VM, which was also wildly unsuccessful at the time. This was due to many different reasons, but the main being that we were using a VM that had remnants of a previous deployment still inside, that was heavily interfering with our attempts to deploy a new platform.

Finally, we moved ourselves into another room, on computers dedicated just for our deployments, which is where we first had success and began implementing our ideas.

4 Solution Implementation

In this section, we will describe our process from the beginning of implementation of a private cloud platform to now. As we began, we passed some troubles that, while being unfamiliar with the platform, did not know how to solve. We got setup in a room dedicated to capstone implementation and began our the deployment of our OpenStack environment. We decided to forego the use of virtual machines for the time being as we were running into errors we later solved. We decided to instead create the deployment straight on the OS of an Ubuntu 20.04 Server. This method worked, of course there were still many errors faced as is to be expected, but we were able to combat them. Creating a working deployment was the largest struggle for our capstone and almost took until the middle of the semester to complete. This was because we treated a few weeks of the semester as a secondary design phase since the plan devised during the previous semester was not the one we were going with. However, once a deployment was up and running with lots of help from Professor Salib, we were underway.

With a now running and stable deployment, it was time for us to begin exploration of the platform and what it had to offer. Thankfully since we did research beforehand, we were not going in blind, but of course there was a lot we were unfamiliar with. We then began running a script manually that would create a public network, router, private network, flavors (sizes), security group rules, a cirros image, and a cirros instance. This transforms an empty deployment environment, to one with internet access, security, a private network, and an instance. Cirros is a Linux image that was created by Red Hat with minimal use for the testing of cloud platforms [5]. This allowed us to do a lot of learning and testing without running into image specific issues. All while doing these things, we were writing instructions on each thing we did, as well as writing weekly status updates on our progress so we could go back and reflect.

Now with a deployment up and running, a VM instance able for use, and a greater understanding of the platform, we could begin customization by modifying which services are enabled to increase function of the deployment. The first began researching and implementing creating new users, assigning them to new projects, and setting up instances, networks, etc. in those projects [6]. This helped us visualize how a deployment in actual function for the department would be like, where each students would have their own user account and their own project. Users can also be assigned to multiple projects, which is helpful while students are in multiple classes, or are working on more than one project at once, because this allows them to have a new network setup, new instances, and new containers

for each project to keep them straight and not mixed up. This completes what is within Lab 2. To go along with creating a deployment to mimic how it would be used practically for the department, we began implementing port forwarding rules that would allow access to the server from outside the network of the deployment and on the network of JMU. Ideally, there would be rules for allowing students access from anywhere, but for the moment, we setup access from anywhere on the JMU network. This includes dorm rooms, libraries, all school buildings, and more around campus. This is a good start to fixing the problem regarding the need to be in our IT labs to work on projects, but not yet solves the problem by allowing access from anywhere.

We next began working on capturing Wireshark packets to see data flow, although we never followed through on the topic unfortunately. We then worked on containers. This was a full circle moment since we started our capstone doing Docker containers, and now we'd be adding the same into our OpenStack deployment [7] [8]. We decided that the instructions we had made on the process following the referenced video during the design semester [3], we would migrate to attempt in the containers in OpenStack. Following the creation of containers and implementing the same process from Lab 1 into them, we implemented volumes [9] [10] [11] [12]. This was not the same process as in regular docker containers, so we followed a new process and added volumes to our deployment through more customization, before realizing that in OpenStack, the need for volumes we expected is not the same in practice. In Docker, we needed volumes added to our Mongo and Mongo-express container setup to have data persistence within the MongoDB through the containers going down and being put back up, because upon default, the database is reset to default each time this happens. In OpenStack, we did not experience the same thing, as the containers could be put down and up and the database remained the same. This highlights one of the features of OpenStack that applies to many things, where what needs to be done manually on other platforms, happens inherently in OpenStack.

At this point, we had done a considerable amount within our deployment, especially since each thing we did was usually done more than once because of issues or user error. We began enabling one of the last services before we were done progressing forward. This was swift, which is known as Object Store within the platform [13] [14], and is labeled as containers, but acts like folders to store things in the deployment. We did this as a stepping stone to allow us to create our own images using glance (OpenStack's image service) but because of time, we never were able to finish this.

At this point, we had to start migrating everything we had done, including the instructions we had written, into VM implementation instead of in hardware as we had been doing to this point. We did this because it allows students to each be able to create their own deployments, which is not for practical use, but for learning and understanding not just how the platform works, but how to create, setup, and be the admin of a deployment before they were just added as users to a deployment. This brings us to now, where we write to wrap up our semester.

5 Results

5.1 Overview

For this project, our results consist of 2 labs and 1 pilot trial. The pilots trial was completed by 1 volunteer student who was a senior level Information technology student with no prior knowledge in OpenStack specifically, but some experience in cloud instances such as AWS and Docker. Our pilot trial was conducted on our Lab 2 completed lab instructions, which has much edits after the run through. The first lab was focused in Docker and container creation in Docker. The second lab and pilot trial was focused on the deployment of openstack and administrator view of creating images, instances, and networks. The third lab was focused in the user perspective of OpenStack and how a user would complete networking and other activities in openstack, though this is not complete.

5.2 Pilot Studies Procedure and Results

The pilot trial we conducted was done by Kagen Roberts, a senior IT student with no OpenStack experience. He was very helpful to the edits of our lab instructions, because when creating a lab and going over these topics a million times beforehand, it is easy to overlook instructions that may be hard to follow by a student new to the topic. His photos going through the lab (Lab 2) can be seen in 2-img/Pilot Trial. The edits to the instructions were what seemed to be minimal to us, since no content was changed, but the order of things, and reminders to click save, etc. are definitely helpful when completing a lab, so of course important to include in a lab. We found that all his changes stemmed from things we failed to migrate from our hardware version of instructions to instructions for the same processed, but within the VMs. His edits helped us create a finer version of our instructions, that later faced much more edit while being ran through again.

5.2.1 Lab 01

In the first lab we focused only on the use of Docker. We wanted to create a lab to allow students to start to gain experience into the world of cloud computing by first understanding Docker and how it uses containerized cloud computing. This lab works in Docker teaching students how to use Docker by pulling images, creating containers, docker-compose, and how to customize Docker Files and creating images off of them. This lab is an overview of docker and skills we will later use through the same concepts just different ways of executing, in OpenStack. This is so students understand and see what docker is like so that using containers in

OpenStack is not as new to them.

Exercises

No pilot trial was executed on this lab because of a lack of time and not wanting to burden other students with the task during finals week. This lab has one exercise and two steps.

Step 1, Task 1 gets right into it by pulling images from docker hub that will be used in the lab [15]. Step 1, Task 2 continues by running these images separately and checking their logs to become familiar with them. Step 1, Task 3 then accesses the express webpage and the javascript page to begin making edits and seeing records in the database.

Step 2, Task 1 begins important work within docker-compose which allows multi-container projects to run as one [16]. Step 2, Task 2 begins with the creation of a Dockerfile we will build as an image to have a customized image that can run as a container [17]. Step 2, Task 3 works mainly in the console of the containers to show usage and what you can do in the Dockerfile. And Step 2, Task 4, includes docker volumes to the compose file to include data persistence [18].

5.2.2 Lab 02 & Pilot Trial

Lab 02 was focused on openstack deployment for individual students using Ubuntu 22.04 VMs to deploy openstack. This lab has a student using an iso of an Ubuntu 22.04 VM to create a OpenStack deployment filled with networks, instances, routers, security rules [19], and more.
This lab has:

- the hardware set up needed to deploy
- the VM interface setup
- the IP address static assignment
- working with kolla customizable containers
- the configuration of the globals file (kolla)
- OpenStack post deployment including networks, instance creation, etc.
- how to access your OpenStack GUI
- Creating networks, images, and instance from an admin viewpoint

This lab is a brief overview of what an admin would be doing in OpenStack with the hands on experience of how to create a customized private cloud deployment.

Exercises

This lab contains one exercise, two steps, the first step has four tasks, and the second step has three tasks.

Step 1, Task 1 covers the setup including a network diagram, requirements of two APs, and importantly setting up network adapters to create the IP configuration desired for the deployment. Step 1, Task 2 continues with setup and features many package installs that are required for the deployment. Step 1, Task 3 contains very important commands that setup the globals file which is vital for deployment. Step 1, Task 4 finishes configuration and deploys OpenStack.

Step 2, Task 1 begins the "post deploy" actions that include spawning a file that will contain the admin password needed to access the deployment via GUI. Step 2, Task 2 begins the work of the "init-runonce" file, that is meant to be ran as a script, but is broken down into individual commands for better understanding of the configurations. Finally, Step 2, Task 3 continues the runonce file, finishing with access instances from the console, sending pings to internet, accessing the console from an external terminal by floating IP [20], and other things to show the full range of usage of an instance in the deployment.

5.2.3 Lab 3

This lab contains topics such as creating new users, new projects, assigning users to projects, adding port forwarding rules to the AP to allow access from other places off the direct network, adding containers rather than just instances, and more. It is displayed differently than the others because it is not complete, and not something we would call a finished lab, but the topics are relevant to include for future use.

6 Discussion

6.1 Discussion & Reflection

In our beginning stages of this project, our goal was to introduce students to cloud computing. The intention was always to lessen the amount of VMs used in labs, but our main goal was to teach students and ourselves more about cloud computing earlier in their/our college careers. We started by using Docker containers and images as a substitute for VMs and thought we would use Docker in the terminal instead of vmware workstation. We wanted to migrate labs from a VM environment to a Docker environment, with the idea that it would make things more accessible and virtualize the storage. The more we thought about our problem and how we would like to solve it, we came to a different conclusion.

We started looking into AWS and OpenStack cloud computing instances instead of focusing just on Docker. We thought it might be better to use a service that could have Docker capabilities, but also could be used as a platform and would allow students to do their work for labs via the cloud. We started by looking into AWS. We were familiar with this service due to using it in other classes, so we knew a little more about how AWS worked, but mainly we were inclined to use a cloud instance that was already deployed and ready for use. We thought OpenStack was less accessible to us for use, but the more we learned about OpenStack, we thought it would fit our project best. We decided on this due to the fact that OpenStack is customizable and we would be able to choose what we use to deploy it, what services are available on it, and the cost it would accrue.

We were able to get working OpenStack instances up and running on both hardware set up (how it would be if JMU had it running for students) and a VM set up deployment (how students will learn how to deploy and manage a cloud deployment). We were able to create 2 full labs that would teach students about Docker, OpenStack and how to use the different features of OpenStack as a student (user) and being professor (administer).

Our expectations for this project was to have a completely working OpenStack instance that we would be able to create multiple labs from and eventually expand our OpenStack instance with more hardware to be able to run a deployment with enough storage and power for the whole department. In reality, we found the deployment and customization of OpenStack rather challenging. This caused us many set backs, specifically attempting to better our OpenStack deployment to allow customized images to be created so a user would be able to create a in-

stance from a image we provided them that had modifications to make it ready for use during a specific lab. We also had set backs changing our environment from a hardware to a VM perspective as this took lots of time for us to grasp. Some things are different in OpenStack depending on the environment, and just the function in general is not the same when using native OS vs a VM.

Overall, we could say we expected more, but those expectations were coming from students in the beginning of the semester that had no idea the magnitude of an OpenStack deployment or what it would entail. We also would have had more time if we had used a VM from the start as we would not have needed over a week of transitioning the setups and changing our instructions to match. If we could change anything, one thing would be to have two meetings a week from the very start of the semester, because having one gives the impression of a laid back setup that a capstone never is. We'd change the few weeks in the beginning of the semester that we let pass by being stuck without a worry, because we were at the beginning of the semester and weren't thinking too much about the semester ahead.

6.2 Conclusion

We had many different ways during this project of solving the big issue of wanting to move from a VM based lab system to a cloud based system, and we came about it from different angles, multiple different times, and found a solid solution in the end. The issues we faced in OpenStack were less than ideal and made this project very tedious at times. These problems are also what helped us become so familiar with the platform. We accomplished most of what we set out to do, with the biggest one being the deployment. We wanted to create containers, instances, public and private networks, working setups for users, multiple projects, and all was accomplished. There were a few topics that we either were proposed to look into by Professor Salib, told we needed to implement, or researched and were interested in applying to our deployment, that did not make it to fruition. These include Wireshark, swift [13], custom glace images, and extra pilot trials. Other than these things, we accomplished what we set out to do for the most part and have tried to capture it all here. In the end, we were able to accomplish the main goal we set out to do which was find a solution for students and OpenStack is definitely the best solution with lots of benefits for students in the future.

6.3 Next Steps

There are many potential next steps for this project. What we were able to accomplish, although we are proud of it, does not scratch the surface of what one is able to do in an OpenStack deployment, especially with the ability of kolla-ansible. We would have liked to do pilot trials on all instructions that we submitted since there was considerable alterations on the one set of instructions that we had tested by a pilot student. We had intentions to also have labs on custom glance images and the mongo/express setup that we did in Docker migrated to OpenStack, which is contained in Lab 3 instructions, but not complete. A very large goal as well was the take labs from IT classes, specifically labs with a lot of hardware or software (many VMs or many APs), and migrate that into OpenStack. We feel that this would be the best way to truly show the impact that the deployment can make by seeing the difference in accessibility once it is transitioned. To go along with the lab we wanted to make using glance images, we would have liked to pick out a specific lab, see the packages, installations, and anything it needed, configure an image file to have those things, then build an image from it. From there, we could launch an instance from that and go on with the lab. This is all that would need to be provided to a student before a lab if working in OpenStack, which could ease lab creation for professors. We would have liked to setup a deployment on hardware, with multiple computers so as to have enough storage, and then create some users, some projects, and give logins to a few students and have them access, create their setups using instructions, and do work within the deployment so both them and us could see real usage. This section could go on forever, but to wrap up, we would have liked to explore much more of the services offered within the globals file, as they are right at our fingertips, but we did not experiment with them. There is so much to be done within OpenStack, but we were thankful to be able to learn and do what we could this semester.

7 References

Appendix A - Timeline/Milestones

Status 7 - Oct 26

Port forwarding instructions

Lab completed, container running with an image created by us, routing between two containers lab

Status 8 - Nov 2

Volumes

Status 9 - Nov 9

Finished mongodb + mongo-express lab, wireshark, instructions, questions

Status 10 - Nov 16

Routing lab

Dry Run 1 - Nov 28

Start to finish demo, labs, containers, instructions

Symposium - Dec 2

Dry Run 2 - Dec 5

Start to finish demo, labs, containers, instructions

Final - Dec 12

Final paper, final report, published paper?

Appendix B - Weekly Status Updates

Status #-1 To-Do List - 8/29/2023

(a) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. read about kolla ansible

ii. demo open stack AIO in its rudimentary form

iii. setup chrome remote desktop + practice

Status #0 To-Do List - 9/5/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. both attempted hardware implementation, stuck

ii. resorted to VM implementation, first the baseline, next the implemented VM, stuck

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Attempt deployment on previously implemented VM

ii. Attempt deployment on baseline VM

iii. Attempt deployment on at-home hardware

Status #1 (1) To-Do List - 9/12/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Katie - able to reconfigure VM previously deployed

ii. Casey - unsuccessful with both

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. set up new hardware environment in new lab

ii. meeting twice more this week, finishing deployment on hardware setup

Status #1 (2) To-Do List - 9/14/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Lab hardware set up complete with 2 workstations
ii. 2 meetings a week implemented for the semester
iii. Casey and Katie's instance have both completed open stack deployment successfully
iv. Modified Salib's given instructions to accurately represent our set ups needs

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Run init-runonce manually on Katie's environment
ii. Write up detailed instructions on the process.
iii. Network Diagram rough draft on lab setup.

Status #2 To-Do List - 9/19/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Ran init-runoce file manually on Katie's environment
ii. Wrote up instructions in detail on how the process works on terminal manually
iii. We completed network diagrams and network setup instructions drafts
iv. Completed a rough brainstorm of topics

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Research + brainstorm ideas for labs involving docker, openstack, possible others, rough proposal
ii. Create a scenario, work with users, permissions, passwords, AWS comparison, write instructions on it

Status #3 To-Do List - 9/28/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. experimentation with admin, users, projects, roles
ii. creating second fully functioning project
iii. adding users to the deployment
iv. creating internet access for the new projects interface

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Wireshark working and capturing on openstack instance connected directly to public network
ii. Experiment with port forwarding to ssh from separate network
iii. Volumes, data persistence

Status #4 To-Do List - 10/5/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Katie - Wireshark installed, running, capturing
ii. Enabled volumes by enabling Cinder in gloabls.yml + reconfigure
iii. Completed first lab instructions + submitted (might need some extra polishing in some sections)

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Scenarios of how the system will be made use of by users

Status #5 To-Do List - 10/12/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Created multiple scenarios

ii. Focused research on containers and volumes

iii. Demo creating containers and volumes - without usability

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Demo containers with a working console + instructions

Status #6 To-Do List - 10/17/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Accomplished running and opening console in multiple container with little current function

ii. Containers ran through object-store and container sections, both with terminals, little function

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Port forwarding setup + instructions

ii. Routing lab

Status #7 To-Do List - 10/27/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Attempted to connect to Mongo containers with ssh and connecting Mongo container to Mongo server running locally. Neither worked.

ii. Worked on understanding how containers work and worked in containers to see what they are capably of doing.

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Choose whether to continue in OpenStack or change to a Docker setup and continue with the capstone from there.

ii. Demo a Mongo and a Mongo-Express Container that can run a HTML code on a web page.

Status #8 To-Do List - 11/2/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Volumes

ii. Instrutions on ssh into containers (write-up 7)

iii. Instructions on "Connect the OpenStack Mongodb Container To a Local Mongodb Server" (failed) (write-up 8)

iv. Wrote beginning of mongo + mongo-express lab (write-up 8) (will continue by Tuesday)

v. wrote volumes instructions (write-up 8)

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. MongoDB + Mongo-express lab instructions, questions, wireshark evidence, screenshots

ii. Finalized plan/timeline

Status #9 To-Do List - 11/9/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Glance research + execution until stopped by not having swift
ii. Beginning enabling process of swift
iii. Recruited + started draft of pilot testing

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Pilot testing session
ii. Glance image completed + instructions
iii. Start transferring instructions to VMs, take photos, solidify instructions

Status #10 To-Do List - 11/17/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. Decided on no longer continuing with swift + glance to create glance images because we had unforeseen errors that caused one of our openstack instances to completely break.
ii. Started transferring openstack instructions to VM's, screenshots and solidified network set up for VM use of openstack. solidified

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. Pilot trials started
ii. Symposium presentation draft
iii. Complete a lab with screenshots, questions and refined, working instructions.

Status #11 To-Do List - 11/30/2023

(a) **Pre-Meeting - Status of last week's Action Items and/or Milestone due this week**

i. SYMPOSIUM REPORT FINISHED (Draft 1)

ii. LAB 1 will be last semester's Docker lab, refined with questions, screenshots, and formatting. - draft

iii. LAB 2 will be openstack deployment and initrunonce formatted as lab (separated to exercises, steps, tasks). Screenshots and added questions. - draft

(b) **Post-Meeting - Action Items for next week and/or Milestones due next week**

i. LAB 3 Users and admin controls in OpenStack with changing what we need in globals, enable volumes.

ii. LAB 4 Volumes and mongo, mongo-express and connecting them in openstack with

iii. LAB 5 glance, swift (work in progress)

iv. Overleaf Draft 1

1. HIGH PRIORITY
   (a) Pilot session + recruiting
   (b) Volumes write up
   (c) Glance image container
   (d) Wireshark
   (e) List of labs we will report on + exercises
2. LOW PRIORITY
   (a) Symposium slides
   (b) Networking in mongo, routing containers

Appendix C - System/Network Architecture

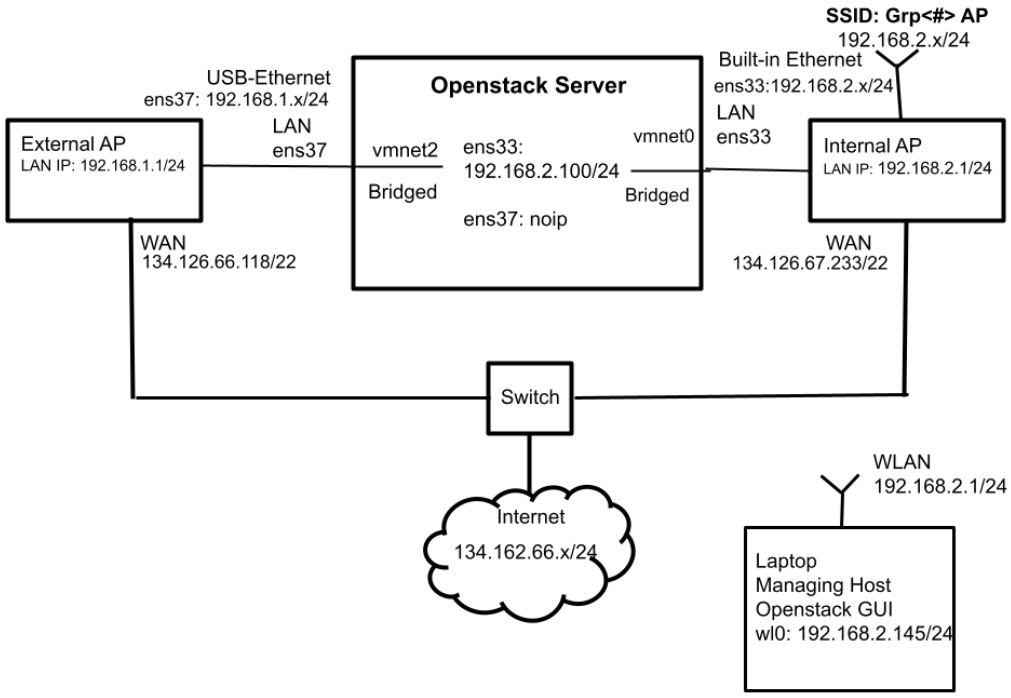


Figure 1: Network Architecture - Hardware Setup

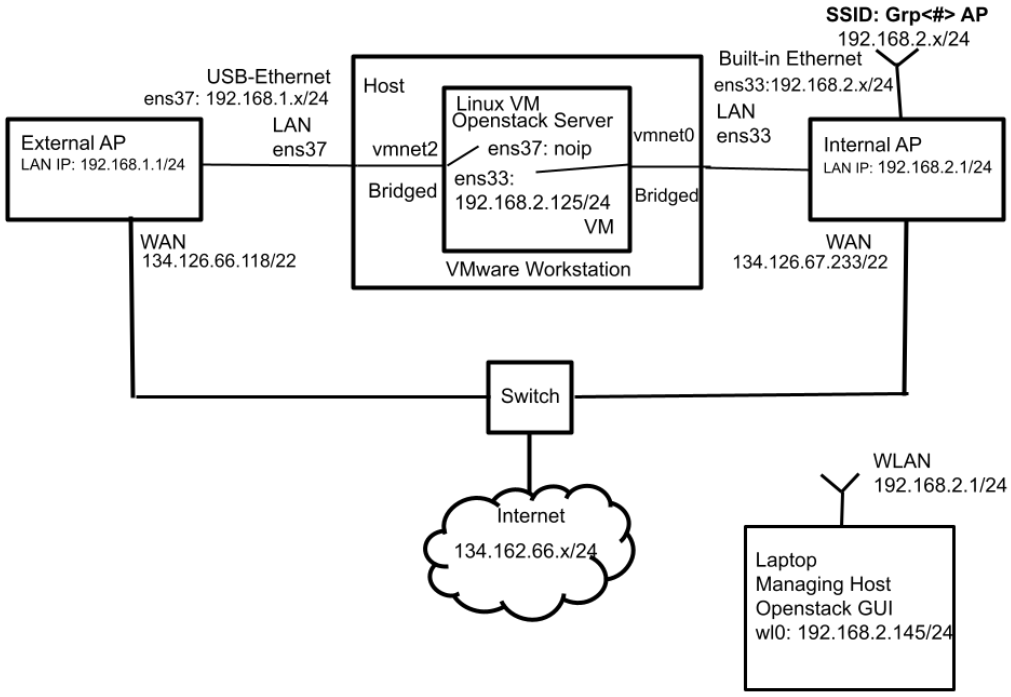


Figure 2: Network Architecture - VM Setup

Appendix D - Weekly Write-Up

Status #1 Write-Up - 9/14/2023

(a) **Katie's setup implementation and deployment Network Diagram used for instructions 3**

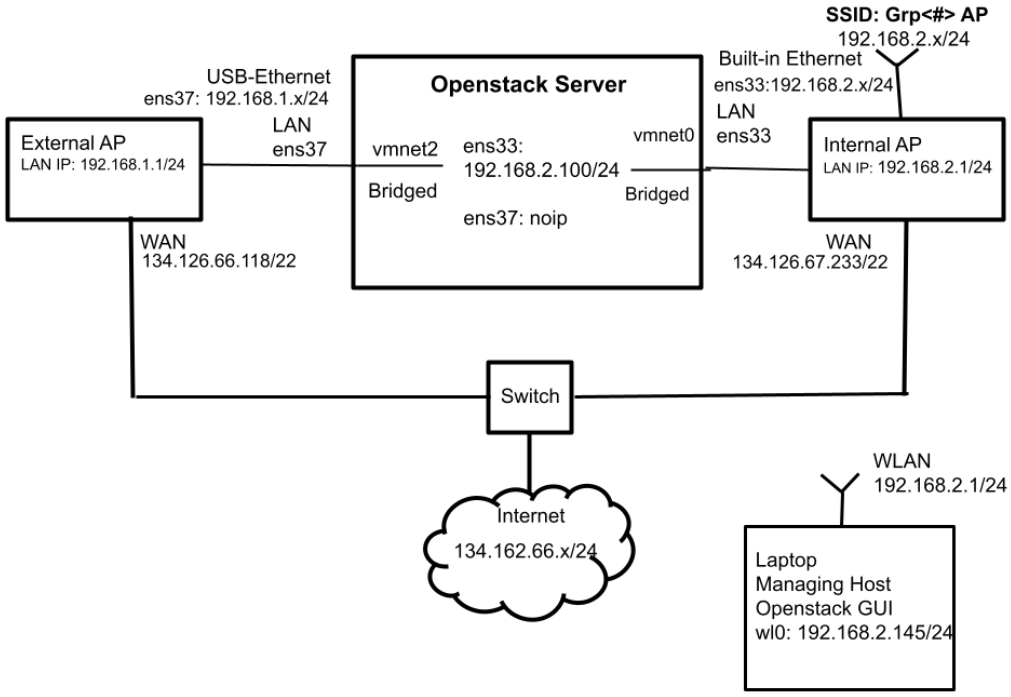


Figure 3: Network Diagram for setup

```
sudo apt update
sudo apt upgrade
sudo reboot
ping 8.8.8.8
ip a (has set static IP address 192.168.2.135)
edited netplan to mimic Appendix A (has diff interface names)
sudo apt install git python3-dev libffi-dev gcc libssl-dev
sudo apt install python3-venv
```

```
python3 -m venv  wallaby
source wallaby/bin/activate
pip install -U pip
pip install 'ansible>=6,<8'
pip install \
git+https://opendev.org/openstack/kolla-ansible\@stable/2023.1
sudo mkdir -p /etc/kolla
sudo chown $USER:$USER /etc/kolla
cd
ls -al (check permissions)
cd /etc/kolla
cp -r ~/wallaby/share/kolla-ansible/etc_examples/kolla/*
/etc/kolla
cp ~/wallaby/share/kolla-ansible/ansible/
inventory/all-in-one .
ls (should have all-in-one, globals, passwords)
sudo vim passwords.yml (to check empty)
to get globals in correct form, scp from Casey,
set both to same router for connectivity to do so, ping first
change IP in globals + interface names if necessary
cd
kolla-ansible install-deps
kolla-genpwd
sudo vim /etc/kolla/password.yml (check for full)
sudo apt update
sudo apt install timeshift
sudo timeshift {create
cd /
cd run
cd timeshift
cd backup
sudo su
cd timeshift
cd snapshots
cd 2023..
ls -al
exit
cd /etc/kolla
kolla-ansible -i ./all-in-one bootstrap-servers
kolla-ansible -i ./all-in-one prechecks
```

```
kolla-ansible -i ./all-in-one deploy
ls
pip install python-openstackclient \
-c https://releases.openstack.org/
constraints/upper/2023.1
kolla-ansible post-deploy (to generate clouds, admin)
ls
sudo vim admin-openrc.sh
c+p password
private browser, enter 192.168.2.135 (in my case 192.168.2.100)
openstack webpage appears
username admin
password paste
appear

Appendix A

‘‘‘

# This is the network config written by ‘subiquity’
network:
    version: 2
    renderer: networkd
    ethernets:
        ens33:
            dhcp4: true
            match:
                macaddress: xx:xx:xx:xx:xx:xx
                ^ EUI-48 addresses of your internal interface (2.x)
            set-name: ens33
        ens37:
            dhcp4: no
            addresses: []
            match:
                macaddress: xx:xx:xx:xx:xx:xx
                ^ EUI-48 addresses of your external interface (1.x)
            set-name: ens37

‘‘‘
```

```
xxx = only if your default interface names were not ens33, ens37,
or something of the sort


Appendix B

‘‘‘

workaround_ansible_issue_8743: yes
kolla_base_distro: "ubuntu"
openstack_release: "2023.1"
kolla_internal_vip_address: "192.168.2.x" ← set to your static IP

docker_configure_for_zun: "yes"
containerd_configure_for_zun: "yes"
docker_apt_package_pin: "5:20.*"
docker_yum_package_pin: "20.*"

network_interface: "ens33" ← might be different if you
did not change your interface names
neutron_external_interface: "ens37" ← might be different
enable_haproxy: "no"
enable_etcd: "yes"
enable_horizon_zun: "{{ enable_zun | bool }}"
enable_kuryr: "yes"
enable_zun: "yes"
enable_magnum: "no"
nova_compute_virt_type: "qemu"

‘‘‘
```

(b) **Katie's setup implementation and deployment**

```
source wallaby/bin/activate
cd /etc/kolla
source admin-openrc.sh

# check that this is empty

openstack image list
```

```
# on webpage open openstack page and make sure everything is
empty and no instances exists (except security group)

KOLLA_DEBUG=${KOLLA_DEBUG:-0}
KOLLA_CONFIG_PATH=${KOLLA_CONFIG_PATH:-/etc/kolla}
KOLLA_OPENSTACK_COMMAND=openstack

# setting variables

ARCH=$(uname -m)
CIRROS_RELEASE=${CIRROS_RELEASE:-0.6.1}
IMAGE_PATH=/opt/cache/files/
IMAGE_URL=
https://github.com/cirros-dev/cirros/releases/download/
${CIRROS_RELEASE}/
IMAGE=cirros-${CIRROS_RELEASE}-${ARCH}-disk.img
IMAGE_NAME=cirros
IMAGE_TYPE=linux

IP_VERSION=${IP_VERSION:-4}
DEMO_NET_CIDR=${DEMO_NET_CIDR:-'10.0.0.0/24'}
DEMO_NET_GATEWAY=${DEMO_NET_GATEWAY:-'10.0.0.1'}
DEMO_NET_DNS=${DEMO_NET_DNS:-'8.8.8.8'}

ENABLE_EXT_NET=${ENABLE_EXT_NET:-1}
EXT_NET_CIDR=${EXT_NET_CIDR:-'192.168.1.0/24'}
EXT_NET_RANGE ={EXT_NET_RANGE:-'start=192.168.1.150,
end=192.168.1.199'}
EXT_NET_GATEWAY=${EXT_NET_GATEWAY:-'192.168.1.1'}

export OS_CLIENT_CONFIG_FILE=${KOLLA_CONFIG_PATH}/clouds.yaml
export OS_CLOUD=kolla-admin

# downloading image

curl --fail -L -o ${IMAGE_PATH}/${IMAGE} ${IMAGE_URL}/${IMAGE}
openstack image list

# creating glance image
```

```
$KOLLA_OPENSTACK_COMMAND image create --disk-format qcow2 \
--container-format bare --public --property os_type=${IMAGE_TYPE} \
--file ${IMAGE_PATH}/${IMAGE} ${IMAGE_NAME}

# Create router

$KOLLA_OPENSTACK_COMMAND router create demo-router
SUBNET_CREATE_EXTRA=""

# Create network, subnet, router, external network,
ext subnet, ext router

$KOLLA_OPENSTACK_COMMAND network create demo-net

$KOLLA_OPENSTACK_COMMAND subnet create --ip-version ${IP_VERSION} \
--subnet-range ${DEMO_NET_CIDR} --network demo-net \
--gateway ${DEMO_NET_GATEWAY} --dns-nameserver ${DEMO_NET_DNS} \
${SUBNET_CREATE_EXTRA} demo-subnet

$KOLLA_OPENSTACK_COMMAND router add subnet demo-router demo-subnet
if [[ $ENABLE_EXT_NET -eq 1 ]]; then
$KOLLA_OPENSTACK_COMMAND network create --external \
--provider-physical-network physnet1 \
--provider-network-type flat public1

$KOLLA_OPENSTACK_COMMAND subnet create --no-dhcp \
--ip-version ${IP_VERSION} \
--allocation-pool ${EXT_NET_RANGE} --network public1 \
--subnet-range ${EXT_NET_CIDR} \
--gateway ${EXT_NET_GATEWAY} public1-subnet

if [[ $IP_VERSION -eq 4 ]]; then
$KOLLA_OPENSTACK_COMMAND router set --external-gateway public1
demo-router
else
$KOLLA_OPENSTACK_COMMAND router set --external-gateway public1 \
--fixed-ip subnet=public1-subnet,ip-
address=${EXT_NET_DEMO_ROUTER_ADDR} \
demo-router
```

```
fi
fi

# project list, security group, security group rules

ADMIN_PROJECT_ID=$($KOLLA_OPENSTACK_COMMAND project list \
| awk '/ admin / {print $2}')

ADMIN_SEC_GROUP=$($KOLLA_OPENSTACK_COMMAND security group list \
--project ${ADMIN_PROJECT_ID} | awk '/ default / {print $2}')

$KOLLA_OPENSTACK_COMMAND security group rule create --ingress \
--ethertype IPv${IP_VERSION} \
--protocol icmp ${ADMIN_SEC_GROUP}

$KOLLA_OPENSTACK_COMMAND security group rule create --ingress \
--ethertype IPv${IP_VERSION} \
--protocol tcp --dst-port 22 ${ADMIN_SEC_GROUP}

$KOLLA_OPENSTACK_COMMAND security group rule create --ingress \
--ethertype IPv${IP_VERSION} \
--protocol tcp --dst-port 8000 ${ADMIN_SEC_GROUP}

$KOLLA_OPENSTACK_COMMAND security group rule create --ingress \
--ethertype IPv${IP_VERSION} \
--protocol tcp --dst-port 8080 ${ADMIN_SEC_GROUP}

# ssh key + quotas

if [ ! -f ~/.ssh/id_ecdsa.pub ]; then
echo Generating ssh key.
ssh-keygen -t ecdsa -N '' -f ~/.ssh/id_ecdsa
fi

if [ -r ~/.ssh/id_ecdsa.pub ]; then
echo Configuring nova public key and quotas.
$KOLLA_OPENSTACK_COMMAND keypair create --public-key \
~/.ssh/id_ecdsa.pub mykey
fi
```

```
# alter optional amounts

$KOLLA_OPENSTACK_COMMAND quota set --instances 40 ${ADMIN_PROJECT_ID}
$KOLLA_OPENSTACK_COMMAND quota set --cores 40 ${ADMIN_PROJECT_ID}
$KOLLA_OPENSTACK_COMMAND quota set --ram 96000 ${ADMIN_PROJECT_ID}

# creating flavors

if ! $KOLLA_OPENSTACK_COMMAND flavor list | grep -q m1.tiny; then
$KOLLA_OPENSTACK_COMMAND flavor create --id 1 --ram 512 \
--disk 1 --vcpus 1 m1.tiny
$KOLLA_OPENSTACK_COMMAND flavor create --id 2 --ram 2048 \
--disk 20 --vcpus 1 m1.small
$KOLLA_OPENSTACK_COMMAND flavor create --id 3 --ram 4096 \
--disk 40 --vcpus 2 m1.medium
$KOLLA_OPENSTACK_COMMAND flavor create --id 4 --ram 8192 \
--disk 80 --vcpus 4 m1.large
$KOLLA_OPENSTACK_COMMAND flavor create --id 5 --ram 16384 \
--disk 160 --vcpus 8 m1.xlarge
fi

% to finish
cat << EOF

# to create instance
openstack server create --image cirros --flavor m1.tiny \
--key-name mykey --network demo-net demo1
```

Status #3 Write-Up - 9/28/2023

(a) **Implementing and Understanding: Admin, Users, Passwords, Projects**

To begin exploration, we created a new project. We were unaware at the time, but this was the cornerstone of this section and essentially creates a whole new openstack dashboard. We then added a new user. This contained a username, email, password, a primary project assignment, and a role. The different roles also became an important part of the section of the project as they entirely control what each user can do. We added the project we created previously

as the primary project for this user, and added the role "_member_". We were unsure what the difference was between the "member" role and the "_member_" role at the start but shortly discovered there was not much difference besides the "member" role had the image from our admin project already loaded but the other had nothing. All new projects came loaded with a public network (with no subnet attached and no internet), as well as the security groups that also came loaded in our original admin project. We eventually changed the role of this user in this project to the "member" role and used that in all future users. At this point, we logged in as a new user for the first time and saw the new dashboard. This showed us a different view than we were used to as now we were members and not admins/heat stack owners. We lost access to the admin section in the left side navigation bar, and a few other tabs in the Identity section on the navigation bar. All of the instances we had created in the admin project were gone, as well as the floating IPs, router, networks, subnets, and assigned floating IPs. We eventually made another project and another user with this second project as its primary project. This user had a member role on this project. Now that we have explored logging in as different users, seen new projects, experimented with roles, we wanted to make the new projects functional. We also experimented with adding new admin users and reader users. As a reader, you are able to see all that a member can, yet you cannot add or delete anything (router, subnet, network, instance). An admin user has admin access to the project and regains the admin section on the navigation bar. Admin also has the ability to delete instances, Routers, subnets and networks from the system as well as see what other users have access to the project. A _heat_stack_user_ is the highest amount of permissions a user can have. Heat_stack_users have all the same permissions of an admin but also have the ability to assign members to a project, see any project happening on the openstack deployment and change members roles including an admins role.. We focused now on one project in order to bring functionality. We added another network to act as an internal network and added a subnet to it. We then added an instance to this network. We added a router in between the public and internal networks. From here, we began to test connectivity from inside the instance. We now needed to create a subnet attached to our public network. On a project that's only user has a member role, we are unable to do this. We found that a project needs a user with the admin role in order to do this. To combat this, we added the admin user to

the project as an admin role, and this added the subnet (associated with our 1.x AP) that was configured in the admin project. During setup, it is smart to have a user with admin permissions in order to have full access to the project, and it can be removed/demoted to member after setup is finished. We tried to ping 8.8.8.8, and were unsuccessful. We were not sure why, and after a lot of troubleshooting, we found that once the four additional security rules were added, we were able to access the internet. Another important thing to check was the IPs of the internal network's subnet to be sure the gateway ends with .1.

Status #5 Write-Up - 10/12/2023

(a) **User Scenarios**

- An admin creates a working network for the user to use
- A user wants to create and launch a website using docker containers in openstack so they are able to access their API server that is running the website from anywhere.
- A user wants to have an instance with internet connection to ping google
- A user wants to access their instance from anywhere, not just in the lab, to have access to their files, via floating IP
- A user wants to create and launch docker containers using docker images that are/are not already in the environment
- A user wants to create a Kubernetes type setup with their containers
- A user wants to launch an instance from a particular image
- A user wants to modify security/access for their instance
- A user wants to trace where packets go to and from in regards to their instance

Status #7 Write-Up - 10/27/2023

CONTAINERS

(a) First go into your ssh openstack client

- Source wallaby/bin/activate

- cd /etc/kolla
- Source admin-openrc.sh

(b) Now that you are in your openstack deployment you need to access the globals.yml. One inside uncomment

(c) **OpenStack options**

- enable_openstack_core: "yes"
- enable_glance: " enable_openstack_core — bool "
- enable_haproxy: "no"
- enable_keystone: " enable_openstack_core — bool "
- enable_neutron: " enable_openstack_core — bool "
- enable_nova: " enable_openstack_core — bool "

(d) Then save the document and reconfigure the openstack deployment.

- kolla-ansible -i ./all-in-one reconfigure

(e) Once this command has worked successfully log on to your openstack webpage. You should see a new tab in the side bar that is labeled "container infra" and "Object store". We will now create a container and start messing around with openstacks container capabilities.

SHH INTO A CONTAINER

(a) We now have containers that are able to run. Since the consoles even when they work do very very little. But we are able to allocate floating Ips to them so our thought process naturally brought us to the idea of ssh, since most of the containers do not have a terminal prompt already installed. We started this experiment with a container made from a Ubuntu image. We did this because this container without us having to change any default setting already came with a terminal. We then used the allocated ip address to this container, went into our wireless client and executed the command "ssh (image name of container)@floatingIP". We were able to connect to the container and were given a prompt to enter the password to complete the connection but then we were met with a realization that we didn't know the password to the container. We then went back into the ubuntu container and found that there was no way to find the current password of the user. We reset the password so that we knew what it was and made sure the ssh server was running. Then we went back to the wireless client and the prompt popped up. We entered the new password and it told us the wrong password on all three prompts and after the last

attempt said private key authentication needed. We figured out that the container will not allow public key authentication and we have no way of using private key authentication so ssh was not a liable option.

PORT FORWARDING

(a) First, connect to your 2.x AP wired. Access the webpage and find the WAN IP address (134.126..) through Status, Sys-Info. Click the Security tab, and scroll to the Block WAN Requests section. Uncheck "Block Anonymous WAN Requests (ping)".

(b) Go to the terminal and ping your WAN IP address. Click the NAT/QoS tab, and hit "Add" below the Forwards section. You will be entering two entries for port forwarding to work by webpage and by terminal ssh.

(c) In the first entry; Application: "OpenStack1", Protocol: Both, Source Net: 0.0.0.0/0, Port from: 2200, IP Address: <the address used to access your deployment, 2.x static IP>, Port to: 22, Enable: checked in.

(d) In the second entry; Application: "OpenStack2", Protocol: Both, Source Net: 134.126.0.0/0, Port from: 80, IP Address: <the address used to access your deployment, 2.x static IP>, Port to: 80, Enable: checked in. Hit save. Hit apply settings.

(e) Remove the wired connection to your AP2. Try to ssh from the terminal with command "ssh checkout@134.126.x.x -p 2200". This should be successful. Now, try to access OpenStack from a browser by entering your WAN IP address. This may take a moment, but should be successful.

(f) Finally, to allow remote access to the AP webpage without wired/wireless connection, you must go to the Administration tab, Remote Access section, select Enable

Status #8 Write-Up - 11/2/2023

Connect the OpenStack MongoDB Container To a Local MongoDB Server

`We are using a client that is not running or sshed into the openstack server but is in the same network as the opentack server. Since we are on Ubuntu 22.04 we cant run mongodb server on this system because of`

the updates in ubuntu no longer match up with mongodb. To combat this we instead download and run a mongod server.
We start with installation commands to prep the system
sudo apt install software-properties-common gnupg apt-transport-https ca-certificates -y

echo "deb [arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-7.0.list

wget [arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/7.0 multiverse

sudo apt update
After everything has updated properly we the install mongod
sudo apt install mongodb-org -y
Once it is installed we now get to start the server.
sudo systemctl start mongod
sudo systemctl enable mongod
To get a Mongod instance to start on the terminal you use the command Mongosh
Now that we have a working mongod server we want to connect the mongod server to the mongo container running on openstack as a container we do this by first finding the directory that stairs the mongod information.
cd /etc
Then we want to open the mongodb configuration file
Sudo vim mongod.conf
Onec insid ethe file we want to add the internal ip address for the mongo container off of openstack it shoudl ook lik ethis.

net:
port: 27017
bindIp: 127.0.0.1,"internal ip"
And allow access to go through
security:
   authorization: enable
Now you are all set up and able to access the mongodb of the container

```
from your local command line using the command.
mongosh 'mongodb://user@127.0.0.1/?authSource=admin'
This was in an attempt to access the openstack containers through the
local host and attempt connecting through the local hosts mongod
server. The connection of mongod running on the local server from
openstack  to mongo-express is very complicated and we have yet to have
a successful attempt. We were trying to run the mongo-express on docker
through run commands but the location of the mongodb server made us
stop attempting to connect to them.
```

MONGO + MONGO EXPRESS

(a) Add these security rules to your default security group 4

Figure 4: Security Groups on OpenStack

(b) Ssh to openstack terminal from laptop

(c) On openstack webpage create mongo container 5

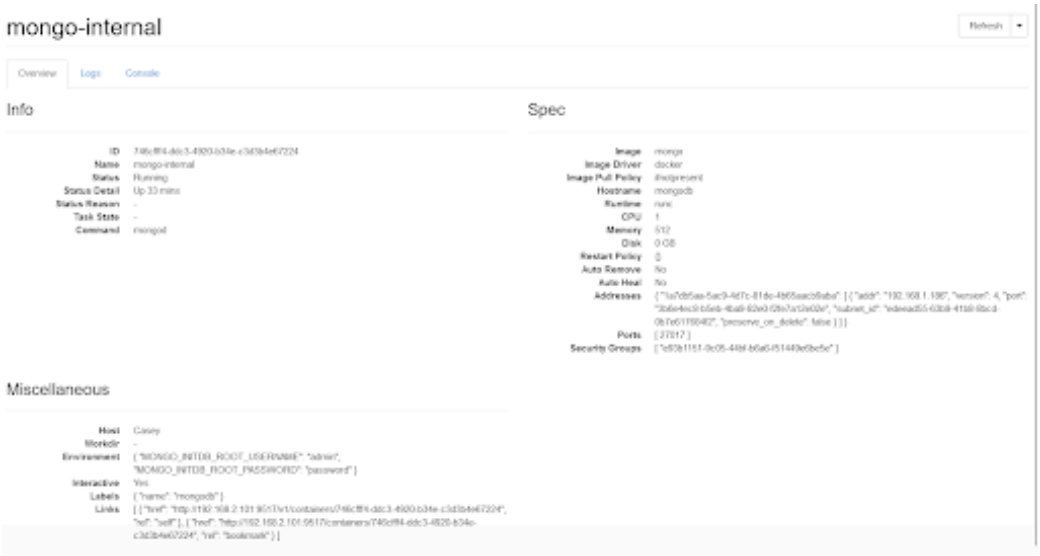


Figure 5: Created Container For Mongodb Contents

(d) On openstack webpage create mongo-express container 6

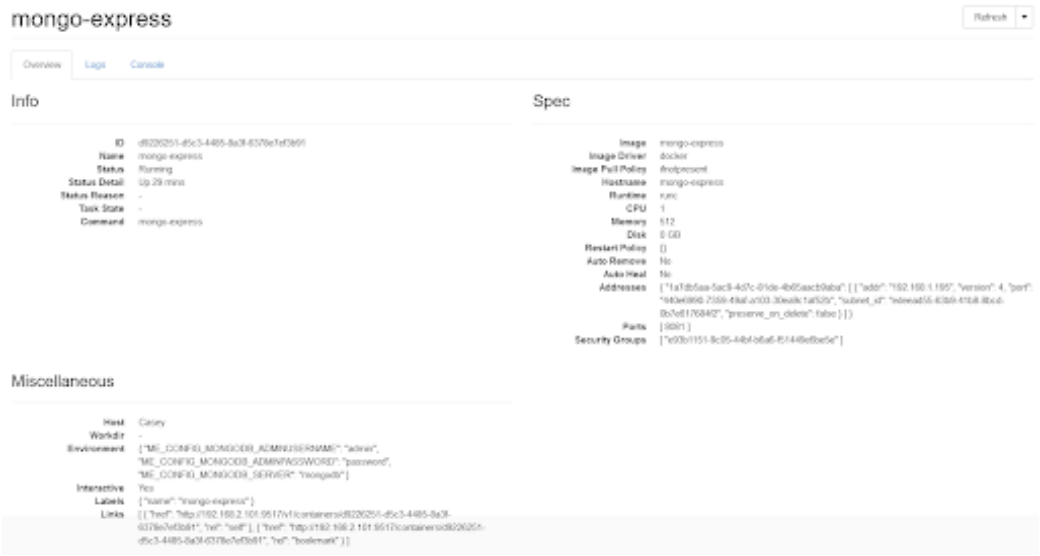


Figure 6: Created Container For Mongo-express Contents

(e) Important to note, passwords and usernames match, hostnames are set in spec, connected to public network

(f) In a webpage, enter <express IP address>:8081 to see webpage, you will be asked for user and pass

(g) This is not the same as the user and pass set as environmental variables

(h) Go to ssh terminal

(i) Enter docker ps

(j) Copy the container ID of the express container created in openstack (not created by docker in terminal)

(k) Enter docker logs <container ID>

(l) Credentials should be shown in red (admin:pass)Enter on express webpage, should be logged in and see databases

VOLUMES

```
Begins with adding a second hard disk into the box of the computer
You can check for the new disk by entering "sudo fdisk -l"  before and
after putting it in

Check which /dev/sdx is new <-- x = a, b, c
    In this case, we are using /dev/sdb
Enter "sudo pvcreate /dev/sdb"
    Be sure the disk is unmounted. If mounted enter "sudo su"
Enter "mount /dev/sdb1 /vols" <-- vols is just a created directory
    If response is "mount: /vols: /dev/sdb1 already mounted on /vols."
    Enter "umount /dev/sdb1 /vols"
Enter "sudo vgcreate cinder-volumes /dev/sdb1"
Enter "y" to the wipe it? question

In globals.yml
Uncomment...

    cinder_volume_group: "cinder-volumes"
    enable_cinder: "yes"
    enable_cinder_backend_lvm: "yes"
    cinder_backend_ceph: "no"

Enter "kolla-ansible -i ./all-in-one reconfigure"
Enter "sudo reboot" once reconfiguration is finished

On OpenStack webpage
```

```
Click Volumes
Click Create Volume
Enter name, leave the rest blank

Click Compute, click Instances
Drop down on a running instance
Click Attach Volume
Select the volume just made

Check for "in-use" on Volumes page

In SSH Terminal
Enter "lsblk"
Should see lines regarding "cinder-volumes" underneath
```

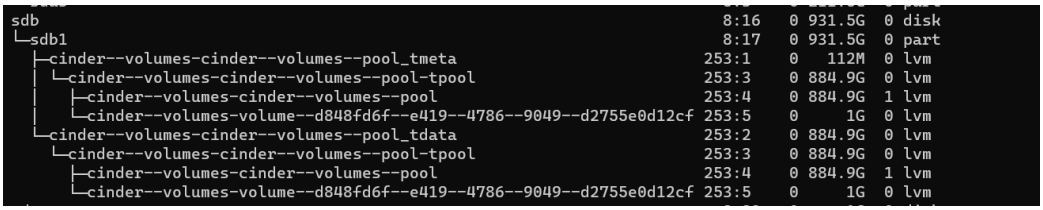


Figure 7: Volumes

Status #9 Write-Up - 11/9/2023

1. **Glance**
   - make sure "enable_opentsack_core= "yes" is uncommented in the gloabls.yml. If not uncomment it and reconfigure openstack
   - Now go into /etc/kolla and ls to find the glance.conf file ours is in /etc/kolla/glance-api/glance-api.conf
   - execute the command mkdir /etc/kolla/config/glance-api.conf (we are replicating the already existing conf file for glance in our instance yours might be under a differnet name)
   - In this new conf file all we need to add these lines.

```
[DEFAULT]
container_formats = ami,ari,aki,bare,ovf,docker
```

- Now we reconfigure the instance
- After re-configuring we look into the original glance-api.conf file is /etc/kolla/glance-api/glance-api.conf and we should see the line we added in our new file in this one meaning that the instance saw the new like and wrote it in to the image will re-configuring.
- In Docker make solidified sure you have a mongo image pulled already for this next part
- Now we attempt to make a glance image in the command line using the command "docker save mongo — openstack image create –container-format docker mongoimage".
- This would work for us for a moment and give us a "Queued" image on our openstack web page with the name we gave our image but after a moment the terminal would give use a "HTTP 5000" error and the image would disappear from the terminal view as well. We assumed this problem was because we lacked a Swift deployment so we started to enable swift to see if that fixed the issue.

2. **Enabling Swift**

```
Swift - Glance Images

add 3 20 G disk (usb drives will do, sdc, sdd, sde).

sudo reboot

(wallaby) checkout@checkout:/etc/kolla$ sudo fdisk -l
[sudo] password for checkout:

    Disk /dev/loop0: 63.45 MiB, 66531328 bytes, 129944 sectors
    Units: sectors of 1 * 512 = 512 bytes
    Sector size (logical/physical): 512 bytes / 512 bytes
    I/O size (minimum/optimal): 512 bytes / 512 bytes


    Disk /dev/loop1: 63.46 MiB, 66547712 bytes, 129976 sectors
```

```
        Units: sectors of 1 * 512 = 512 bytes
        Sector size\customfigure{2-img/lsblk-vols}{Volumes}{lsblk-vols}
(logical/physical): 512 bytes / 512 bytes
        I/O size (minimum/optimal): 512 bytes / 512 bytes


        Disk /dev/loop2: 111.95 MiB, 117387264 bytes, 229272 sectors
        Units: sectors of 1 * 512 = 512 bytes
        Sector size (logical/physical): 512 bytes / 512 bytes
        I/O size (minimum/optimal): 512 bytes / 512 bytes


        Disk /dev/loop3: 43.07 MiB, 45158400 bytes, 88200 sectors
        Units: sectors of 1 * 512 = 512 bytes
        Sector size (logical/physical): 512 bytes / 512 bytes
        I/O size (minimum/optimal): 512 bytes / 512 bytes


        Disk /dev/loop4: 40.84 MiB, 42827776 bytes, 83648 sectors
        Units: sectors of 1 * 512 = 512 bytes
        Sector size (logical/physical): 512 bytes / 512 bytes
        I/O size (minimum/optimal): 512 bytes / 512 bytes


        Disk /dev/loop5: 40.86 MiB, 42840064 bytes, 83672 sectors
        Units: sectors of 1 * 512 = 512 bytes
        Sector size (logical/physical): 512 bytes / 512 bytes
        I/O size (minimum/optimal): 512 bytes / 512 bytes


        Disk /dev/nvme0n1: 238.47 GiB, 256060514304 bytes, 500118192 sectors
        Disk model: KXG60ZNV256G NVMe TOSHIBA 256GB
        Units: sectors of 1 * 512 = 512 bytes
        Sector size (logical/physical): 512 bytes / 512 bytes
        I/O size (minimum/optimal): 512 bytes / 512 bytes
        Disklabel type: gpt
        Disk identifier: 02DB2DA1-484E-45FB-B6B3-0B5DC9BB7D2F

        Device           Start       End   Sectors   Size Type
        /dev/nvme0n1p1    2048   2203647   2201600     1G EFI System
```

```
        /dev/nvme0n1p2 2203648   6397951   4194304     2G Linux filesystem
        /dev/nvme0n1p3 6397952 500115455 493717504 235.4G Linux filesystem


        Disk /dev/mapper/ubuntu--vg-ubuntu--lv: 100 GiB, 107374182400 bytes,
        209715200 sectors
        Units: sectors of 1 * 512 = 512 bytes
        Sector size (logical/physical): 512 bytes /\customfigure{2-img/lsblk-vols}-
512 bytes
        I/O size (minimum/optimal): 512 bytes / 512 bytes


        Disk /dev/sda: 238.47 GiB, 256060514304 bytes, 500118192 sectors
        Disk model: TOSHIBA THNSNK25
        Units: sectors of 1 * 512 = 512 bytes
        Sector size (logical/physical): 512 bytes / 4096 bytes
        I/O size (minimum/optimal): 4096 bytes / 4096 bytes
        Disklabel type: dos
        Disk identifier: 0x72c2160f

        Device     Boot    Start       End   Sectors   Size Id Type
        /dev/sda1  *        2048  10487807  10485760     5G  7 HPFS/NTFS/exFAT
        /dev/sda2       10487808  56625151  46137344    22G  7 HPFS/NTFS/exFAT
        /dev/sda3       56625152 500115455 443490304 211.5G  7 HPFS/NTFS/exFAT


        Disk /dev/sdb: 931.51 GiB, 1000204886016 bytes, 1953525168 sectors
        Disk model: HGST HTS721010A9
        Units: sectors of 1 * 512 = 512 bytes
        Sector size (logical/physical): 512 bytes / 4096 bytes
        I/O size (minimum/optimal): 4096 bytes / 4096 bytes
        Disklabel type: gpt
        Disk identifier: 09ED9D10-9477-4CA5-AE56-704E75F1F91F

        Device     Start        End    Sectors   Size Type
        /dev/sdb1   2048 1953523711 1953521664 931.5G Linux LVM


        Disk /dev/mapper/cinder--volumes-volume--d848fd6f--e419--4786--9049
        --d2755e0d12cf: 1 GiB, 1073741824 bytes, 2097152 sectors
```

```
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 524288 bytes / 524288 bytes


Disk /dev/mapper/cinder--volumes-volume--bd23141b--0e0f--4c9f--998b
--694afe1a5e9e: 1 GiB, 1073741824 bytes, 2097152 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 524288 bytes / 524288 bytes


Disk /dev/sdc: 238.47 GiB, 256060514304 bytes, 500118192 sectors
Disk model: Generic
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: gpt
Disk identifier: DE83B048-0A1D-4961-96DE-5EF139BD48A8

Device     Start       End   Sectors   Size Type
/dev/sdc1   2048 500115455 500113408 238.5G Linux filesystem


Disk /dev/sdd: 238.47 GiB, 256060514304 bytes, 500118192 sectors
Disk model: Angelbird
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 33553920 bytes
Disklabel type: gpt
Disk identifier: 316FB73F-F265-4466-9818-F957745CF16F

Device     Start       End   Sectors   Size Type
/dev/sdd1   1953 500116239 500114287 238.5G Linux filesystem


Disk /dev/sde: 238.47 GiB, 256060514304 bytes, 500118192 sectors
Disk model:  LCS-256M6S 2.5
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
    I/O size (minimum/optimal): 512 bytes / 33553920 bytes
    Disklabel type: gpt
    Disk identifier: D8264236-1247-4D7F-B734-11A23A0FE0CC

    Device     Start       End   Sectors   Size Type
    /dev/sde1   1953 500116239 500114287 238.5G Linux filesystem

(wallaby) checkout@checkout:/etc/kolla$ sudo parted /dev/sdc -s
-- mklabel gpt mkpart KOLLA_SWIFT_DATA 1 -1
(wallaby) checkout@checkout:/etc/kolla$ sudo parted /dev/sdd -s
-- mklabel gpt mkpart KOLLA_SWIFT_DATA 1 -1
(wallaby) checkout@checkout:/etc/kolla$ sudo parted /dev/sde -s
-- mklabel gpt mkpart KOLLA_SWIFT_DATA 1 -1

Potential response: warning about the use of this, continue

(wallaby) checkout@checkout:/etc/kolla$ sudo mkfs.xfs -f -L d0 /dev/sdc1
    meta-data=/dev/sdc1              isize=512    agcount=4,
    agsize=1310592 blks
             =                       sectsz=512   attr=2,
             projid32bit=1
             =                       crc=1        finobt=1,
             sparse=1, rmapbt=0
             =                       reflink=1    bigtime=0
             inobtcount=0
    data     =                       bsize=4096   blocks=5242368,
    imaxpct=25
             =                       sunit=0      swidth=0 blks
    naming   =version 2              bsize=4096   ascii-ci=0,
    ftype=1
    log      =internal log           bsize=4096   blocks=2560,
    version=2
             =                       sectsz=512   sunit=0 blks,
             lazy-count=1
    realtime =none                   extsz=4096   blocks=0, rtextents=0

(wallaby) checkout@checkout:/etc/kolla$ sudo mkfs.xfs -f -L d1 /dev/sdd1
    meta-data=/dev/sdd1              isize=512    agcount=4,
    agsize=1310592 blks
             =                       sectsz=512   attr=2,
```

```
             projid32bit=1
             =                       crc=1        finobt=1,
             sparse=1, rmapbt=0
             =                       reflink=1    bigtime=0
             inobtcount=0
    data     =                       bsize=4096   blocks=5242368,
    imaxpct=25
             =                       sunit=0      swidth=0 blks
    naming   =version 2              bsize=4096   ascii-ci=0,
    ftype=1
    log      =internal log           bsize=4096   blocks=2560,
    version=2
             =                       sectsz=512   sunit=0 blks,
             lazy-count=1
    realtime =none                   extsz=4096   blocks=0,
    rtextents=0

(wallaby) checkout@checkout:/etc/kolla$ sudo mkfs.xfs -f -L d2 /dev/sde1
    meta-data=/dev/sde1              isize=512    agcount=4,
    agsize=1310592 blks
             =                       sectsz=512   attr=2,
             projid32bit=1
             =                       crc=1        finobt=1,
             sparse=1, rmapbt=0
             =                       reflink=1    bigtime=0
             inobtcount=0
    data     =                       bsize=4096   blocks=5242368,
    imaxpct=25
             =                       sunit=0      swidth=0 blks
    naming   =version 2              bsize=4096   ascii-ci=0,
    ftype=1
    log      =internal log           bsize=4096   blocks=2560,
    version=2
             =                       sectsz=512   sunit=0 blks,
             lazy-count=1
    realtime =none                   extsz=4096   blocks=0,
    rtextents=0

(wallaby) checkout@checkout:/etc/kolla$
```

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo vim /etc/sysctl.conf
    Ensure "net.ipv4.ip_forward=1" is listed in list at bottom
sudo sysctl -p
    Ensure "net.ipv4.ip_forward=1" is one of the returned lines

sudo vim init-swift-ring # see appendix, or copy from
https://blog.inkubate.io/configure-swift-on-openstack-
ocata-standalone-with-kolla/

sudo chmod 755 init-swift-ring

(wallaby) checkout@checkout:/etc/kolla$ sudo ./init-swift-ring
 Needs to return with no errors and balanced

(wallaby) checkout@checkout:/etc/kolla$ sudo blkid

    /dev/nvme0n1p3: UUID="NiUgvl-Xx6R-1Txj-hPN7-c3T4-YLS2-yk12Kf"
    TYPE="LVM2_member" PARTUUID="222683e7-d062-4944-ac0a-48a288b55c2d"

    /dev/nvme0n1p1: UUID="AF6C-DA23" BLOCK_SIZE="512" TYPE="vfat"
    PARTUUID="bd42c69c-4f9c-4422-98ea-156d79ee295d"

    /dev/nvme0n1p2: UUID="e754c0b8-a996-4437-8521-52e90a4d2d5f"
    BLOCK_SIZE="4096" TYPE="ext4" PARTUUID="44b7c697-3555-4608
    -a4c8-270ddd693ad8"

    /dev/sdd1: LABEL="d1" UUID="c0069d3e-5886-4f63-939c-4aab0f66fd90"
    BLOCK_SIZE="512" TYPE="xfs" PARTLABEL="KOLLA_SWIFT_DATA"
    PARTUUID="3f311851-5a70-445e-80cf-23410496f453"

    /dev/sdb1: UUID="Zwny0M-w9rh-BsFW-KBvf-LLjs-1vT3-qqCXls"
    TYPE="LVM2_member" PARTLABEL="primary" PARTUUID="86f19ca8-ff00
    -4b09-b45c-29b2a96145b5"

    /dev/mapper/ubuntu--vg-ubuntu--lv: UUID="17e66adc-3c5c-4c2e-baaa
    -05696fe67196" BLOCK_SIZE="4096" TYPE="ext4"

    /dev/sde1: LABEL="d2" UUID="5a4cb383-2606-4ba9-a5ea-fae9d8ffad06"
    BLOCK_SIZE="512" TYPE="xfs" PARTLABEL="KOLLA_SWIFT_DATA"
```

```
    PARTUUID="cc77c514-c019-4c27-bb42-c185d32be11e"

    /dev/sdc1: LABEL="d0" UUID="a23aa370-4fbb-4845-b77c-4d09ed072364"
    BLOCK_SIZE="4096" TYPE="xfs" PARTLABEL="KOLLA_SWIFT_DATA"
    PARTUUID="baadb75d-1712-4998-b551-41a190bef223"

    /dev/sda2: LABEL="Thaw" BLOCK_SIZE="512" UUID="D6A84DC3A84DA2BB"
    TYPE="ntfs" PARTUUID="72c2160f-02"

    /dev/sda3: LABEL="PE_Data" BLOCK_SIZE="512" UUID="9AF25287F2526791"
    TYPE="ntfs" PARTUUID="72c2160f-03"

    /dev/sda1: LABEL="LabDashPE" BLOCK_SIZE="512" UUID="30DE4CB4DE4C73DE"
    TYPE="ntfs" PARTUUID="72c2160f-01"

    /dev/loop1: TYPE="squashfs"
    /dev/loop4: TYPE="squashfs"
    /dev/loop2: TYPE="squashfs"
    /dev/loop0: TYPE="squashfs"
    /dev/loop5: TYPE="squashfs"
    /dev/loop3: TYPE="squashfs"

Open /etc/fstab and add:

UUID="a23aa370-4fbb-4845-b77c-4d09ed072364" /mnt/sdc1 xfs noatime 0 0
UUID="c0069d3e-5886-4f63-939c-4aab0f66fd90" /mnt/sdd1 xfs noatime 0 0
UUID="5a4cb383-2606-4ba9-a5ea-fae9d8ffad06" /mnt/sde1 xfs noatime 0 0

# If these drives are removed for any reason, be sure to remove their
corresponding UUID record before rebooting, reconfiguring

Create the Swift data mount point and test that mounting works:

sudo mkdir /mnt/sdc1
sudo mkdir /mnt/sdd1
sudo mkdir /mnt/sde1

sudo mount -a

sudo reboot
```

```
In globals.yml
WSREP
Uncomment
    swift_storage_interface: "{{ network_interface }}"
    swift_replication_interface: "{{ swift_storage_interface }}"
    enable_swift: "yes"

sudo vim /etc/kola/config/swift.conf
Enter:

    [storage-policy:0]
    name = Policy-0
    default = true

    [swift-constraints]
    max_header_size=32768

sudo vim /etc/kolla/config/swift/proxy-server.conf
Enter:

    [filter:authtoken]
    delay_auth_decision = true

(wallaby) checkout@checkout:/etc/kolla$ kolla-ansible -i ./all-in-one deploy
#Must be deploy, not reconfigure

sudo reboot

# More to continue

Appendix

#!/bin/bash
export KOLLA_INTERNAL_ADDRESS=<>
export KOLLA_SWIFT_BASE_IMAGE="kolla/centos-source-swift-base:4.0.0"
mkdir -p /etc/kolla/config/swift
# Object ring
docker run \
  --rm \
```

```
  -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
  $KOLLA_SWIFT_BASE_IMAGE \
  swift-ring-builder \
    /etc/kolla/config/swift/object.builder create 10 3 1
for i in {0..2}; doWSREP
  docker run \
    --rm \
    -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
    $KOLLA_SWIFT_BASE_IMAGE \
    swift-ring-builder \
      /etc/kolla/config/swift/object.builder add r1z1-
      ${KOLLA_INTERNAL_ADDRESS}:6000/d${i} 1;
done
# Account ring
docker run \
  --rm \
  -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
  $KOLLA_SWIFT_BASE_IMAGE \
  swift-ring-builder \
    /etc/kolla/config/swift/account.builder create 10 3 1
for i in {0..2}; do
  docker run \
    --rm \
    -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
    $KOLLA_SWIFT_BASE_IMAGE \
    swift-ring-builder \
      /etc/kolla/config/swift/account.builder add r1z1-
      ${KOLLA_INTERNAL_ADDRESS}:6001/d${i} 1;
done
# Container ring
docker run \
  --rm \
  -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
  $KOLLA_SWIFT_BASE_IMAGE \
  swift-ring-builder \
    /etc/kolla/config/swift/container.builder create 10 3 1
for i in {0..2}; do
  docker run \
    --rm \
    -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
```

```
    $KOLLA_SWIFT_BASE_IMAGE \
    swift-ring-builder \
      etc/kolla/config/swift/container.builder add r1z1-
      ${KOLLA_INTERNAL_ADDRESS}:6002/d${i} 1;
done
for ring in object account container; do
  docker run \
    --rm \
    -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
    $KOLLA_SWIFT_BASE_IMAGE \
    swift-ring-builder \
      /etc/kolla/config/swift/${ring}.builder rebalance;
done
```

3. **Pilot Experiment Proposal**

- Anthony - Docker lab not using openstack that uses docker compose, and bringing up mongo-express and mongodb
- Kagen - OpenStack Deployment instructions and initrunonce commands
- Phil - Mongo lab:Admin,users, password,projects,containers, connection of the mongo and mongo-express containers in OpenStack

Status #10 Write-Up - 11/17/2023

OpenStack Deployment on VM

All images for this lab can be seen in 2-img/LAB02 folder.

```
Setup your host running the VM with an ethernet cable (direct ethernet port
Perform an ip a to verify you have two interfaces on the host (find the int
Click Edit, Virtual Network Editor. We selected vmnet0 (bridged). Next to "
Click VM, click settings. Change Network Adapter "Network Connection" to Br
Open /etc/netplan/00-installer* and make sure it mimics Appendix A. Enter s

        Appendix A
        ‘‘‘
```

```
        # This is the network config written by 'subiquity'
        network:
          version: 2
          renderer: networkd
          ethernets:
            ens33:
              dhcp4: true
            ens37:
              dhcp4: no
              addresses: []
        ```

We will be statically assigning an IP address to your 2.x interface. To do
Now, you should have an ens33 interface with a static IP, ens37 bridged int
ping 8.8.8.8
sudo apt update
sudo apt upgrade -y
sudo reboot
ping 8.8.8.8 ensure internet connection
ip a
sudo apt install git python3-dev libffi-dev gcc libssl-dev -y
sudo apt install python3-venv -y
python3 -m venv wallaby
source wallaby/bin/activate
pip install -U pip
pip install 'ansible$>$=6,$<$8'
pip install git\+https://opendev.org/openstack/kolla-ansible\@stable/2023.1
sudo mkdir -p /etc/kolla
cd /etc
ls -al | grep kolla
If root root, sudo chown \$USER:\$USER kolla
ls -al | grep kolla
cd kolla
cp -r /wallaby/share/kolla-ansible/etc_examples/kolla/* .
ls
cp /wallaby/share/kolla-ansible/ansible/inventory/all-in-one .
ls
sudo vim passwords.yml
sudo vim globals.yml
Modify globals.yml to mimic Appendix B

```
\*\*Ansible options\*\*
workaround\_ansible\_issue\_8743: yesGlance
\*\*Kolla options\*\*
kolla\_base\_distro: "ubuntu"
openstack\_release: "2023.1"
kolla\_internal\_vip\_address: "192.168.2.x(125)" (set to your stat
\*\*Neutron - Networking options\*\*
network\_interface: "ens33"  ← might be different if you did not ch
neutron\_external\_interface: "ens37"  ← might be different
\*\*OpenStack options\*\*
enable\_haproxy: "no"
enable\_etcd: "yes"
enable\_kuryr: "yes"
\*\*Nova - Compute Options\*\*
nova\_compute\_virt\_type: "qemu"

cd
kolla-ansible install-deps
kolla-genpwd
sudo vim /etc/kolla/passwords.yml
sudo apt update
sudo apt install timeshift -y
sudo timeshift --create (dash dash) (will take a minute)
cd /run/timeshift/backup
sudo su
cd timeshift/snapshots/2023 (hit tab to populate name)
ls -al (checking size of snapshot (info.json file (264)))
exit

Now for deployment
cd /etc/kolla
kolla-ansible -i ./all-in-one bootstrap-servers
Should see, localhost: ok = x failed = 0
kolla-ansible -i ./all-in-one prechecks
Should see, localhost: ok = x failed = 0
kolla-ansible -i ./all-in-one deploy
Should see, localhost: ok = x failed = 0
```

We never got our deployment working w ran into an error that you can see in Figure 8 we tried our very best to fix this error but nothing we found worked. Our globals.yml has all the right interface names and ip addresses the files for WSREP and MariaDB are all the same as our working instance as well as online resources. We are listening on port 4568 which is WSREP port and port 3306 for MariaDB. We are at a complete and total loss. We tried to change scripts connect WSREP to a running MariaDB database on the VM. We could not get past this error. The also occurred for both of the VM instance we had running and any new instance we tried to deploy on the same VM baseline given to us on canvas.

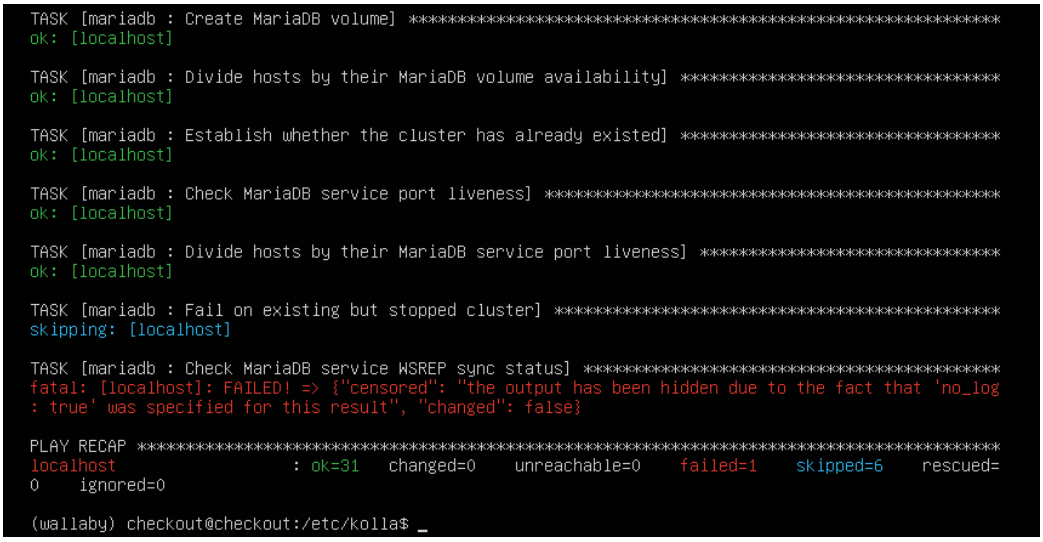


Figure 8: WSREP Error during Deployment

Status #11 Write-Up - 11/30/2023

All Drafts for Capstone Final

1. Refer to atts/CAPSTONE LABS folder for Draft 1 of CAP Lab 1, Draft 1 of Cap Lab 2, Draft 1 of Powerpoint Slides

Status #12 Write-Up - 12/5/2023

1. We completed a Pilot trial for the first exercise in lab 02. We were able to make lots of changes to the instructions using this pilot trial to understand where we needed better descriptions and where we needed to fix errors within the commands. See 2-img/Pilot Trial to see all the screenshots form the pilot trial.

Appendix E - Lab Instructions

Lab 1 Instructions

Docker creation + exploration lab

1. **Learning Objectives**
   - What is Docker?
   - What is a docker container?
   - What is a DockerFile?
   - What is Docker Compose?
   - What are Volumes?
   - What is nodejs?

2. **Equipment & Tools**
   - Ubuntu 22.04 Base Line VM (60GB)

3. **Exercises**

Assumption: The baseline VM has the following downloaded to it (Appendix A);
- Docker
- Docker-compose
- Nodejs
- Npm
- Visual Studio Code
- Techworld-js-docker-demo-app-master [5]

3.1 **Exercise 1 - Docker container setup**

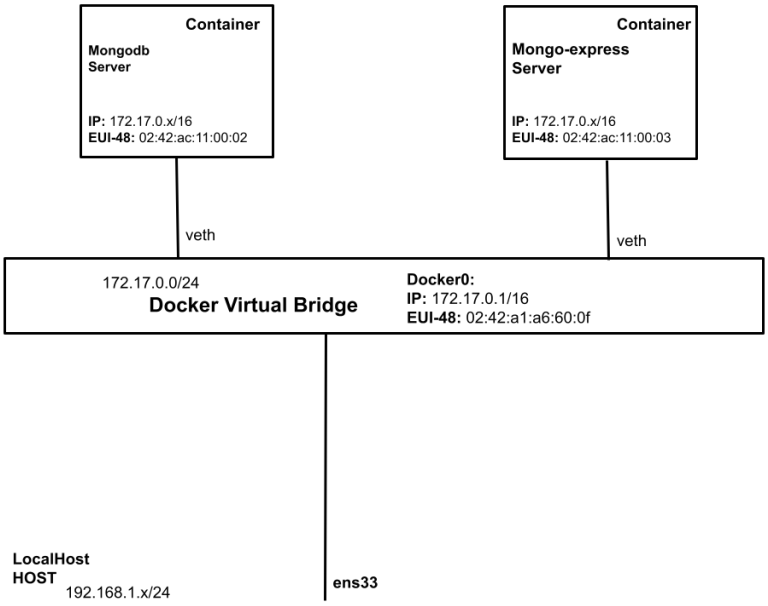


Figure 1: Docker Network Diagram

Step 1 - MongoDB and Mongo-Express setup

Prepare for MongoDB and Mongo-Express Containers
T1. Create a clone of the provided VM.

- **sudo apt update**
- **sudo apt upgrade**

Adding the user to the docker group. [1]

sudo groupadd docker
sudo gpasswd -a $USER docker
sudo service docker restart
docker context use default

docker ps

You should see a column of row names;
"CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES".
This will tell you that docker is installed and working.

docker images
docker pull mongo
docker pull mongo-express
docker pull node
docker images

Q1 - Definition

(a) What is mongoDB?
(b) What is mongo-express?
(c) What is the default tag for images? How do you specify a tag?
(d) What does the “pull” command in Docker do?
(e) Where are mongo and mongo-express being pulled from? [3]
(f) What is the image of "hello-world"?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Q2 - Results

(a) What does "docker images" return now? Show evidence.

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

We then need to run both of these images (which turns them into containers) to make mongodb database available for our application and to connect mongo with mongo-express web-server.

To connect, we need a docker network where the containers are running on. With both containers running in the same network, they can talk to each other only using the container name. The node server can connect from outside using localhost and port # (in chrome browser).

Enter **docker network ls** to see auto generated docker networks.
To create our own isolated docker network for mongodb and mongo-express to run in, enter **docker network create mongo-network.**
Enter **docker network ls** again to see your newly created network.

Q3 - Definition

(a) What networks were there upon default?
(b) What are they for?
(c) Describe driver and scope.

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Running Docker containers

T2. Now, we will run the mongo image as a Docker container in this network.This brings an image up into a container.

```
docker run -d \
-p 27017:27107 \
-e MONGO_INITDB_ROOT_USERNAME=mongo \
-e MONGO_INITDB_ROOT_PASSWORD=mongo \
--name mongodb \
--net mongo-network \
mongo
```

Q4 - Analysis

(a) What is the difference between a docker image and a docker container?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Q5 - Analysis

(a) What does each option in the command mean? (-d, -p, -e, --name, --net)

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

To check success (of running the container), enter **docker logs <string returned from previous command>**
For success, you must see (at the end), "Waiting for connections".

Q6 - Results & Analysis

(a) What else is seen within the container's logs?
(b) What is the MongoDB container waiting to connect with?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Now that mongodb is up and running successfully, we must run the mongo-express container along with.

```
docker run -d \
-p 8081:8081 \
-e ME_CONFIG_MONGODB_ADMINUSERNAME=mongo \
-e ME_CONFIG_MONGODB_ADMINPASSWORD=mongo \
-e ME_CONFIG_MONGODB_SERVER=mongodb \
--net mongo-network \
--name mongo-express \
mongo-express
```

Q7 - Analysis

(a) Do the admin username and password have to be in correspondence with the root username and password set with the mongodb container?
(b) Why or why not?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

To check success, enter **docker logs <string returned from previous command>**
You should see, "Welcome to mongo-express" and "listening at http://0.0.0.0:8081"
Here, you will also see your username and password (not the same as root or admin user and pass set within the containers) that are used for access to the web-server itself. These will be needed for later.

Access Mongo-Express
T3. Enter "localhost:8081" in a chrome browser, should see the mongo-express GUI page

Q8 - Analysis

(a) What username and password need to be entered here?
(b) Where can they be found?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Type in the "Database Name" text box, **user-account**. Click on the database you have made and create a collection named **users.**

We will now connect through node.js

Q9 - Definition

(a) **What is nodejs used for?**

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Enter **docker ps** in terminal to ensure mongo and mongo-express are running. (need to see Status Up x minutes)

Change directory in terminal to the app folder, inside techworld folder, inside Downloads folder.
Open vscode. File > Open Folder > app (same path as above).
Select "Yes I trust the authors" (if applicable)

Users collection is empty (mongo-express webpage).
Enter **node server.js** in the terminal (in app directory).

Q10 - Definition

(a) **What does "node server.js" do?**
(b) **What should you see as a response?**
(c) **Where is server.js configured to return that?**

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Then enter **localhost:3000** in another web browser to see the HTML file on screen.

Click edit profile, change something minor in the text boxes (Smith → Samson), then refresh the page of your users collection to see a new document loaded with an id, userid, email, interests, and name; specifically seeing that the name and email match what you changed them to.

You can click on the userid from this page to see another look at the new instance.

Open a new terminal instance, (so keep node running) enter **docker ps** to find the container ID of mongo, enter **docker logs <container id>** to see what is happening inside.

To see only what is added when a modification is made on the webpage, enter **docker logs <container id> -f**, make a line (----------) at the end of the stream, make another change in the web file, and look at the terminal again to see new activity generated by the change.

Q11 - Results & Analysis

(a) **What is happening within the log files? Explain in detail and provide evidence. Connect logs back to the instance in the users collection.**
(b) **Submit logs as attachment E1S1T3-mongo-express-change-mongo-logs.txt.**

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Step 2 - Docker-compose

Docker-compose

T1. In order to make it simpler to run docker containers we will configure a docker-compose.yaml file to do this for us.

Network configuration is not needed because docker compose takes care of creating a common network for the containers within the same yaml file. Create this file in VSCode, name it docker-compose.yaml. Make sure it saves within the app folder we've been working in.

To use this file, we must stop the running of the previous containers and start them again using the file once created. Go to the terminal and enter **docker ps**. Use the container ID of each container that is running and stop them using **docker stop <container id>**.

Copy what is seen in Appendix B to your file in VSCode. Save.

Enter **docker-compose -f docker-compose.yaml up -d**

Q1 - Definition & Results

(a) What does docker-compose do?
(b) What are the results of the docker-compose up command?
(c) What do -f and -d do?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Run **docker ps** to ensure both are running.

Enter **docker-compose -f docker-compose.yaml down**
Enter **docker-compose -f docker-compose.yaml up**

Q2 - Analysis

(a) What is the difference? (between using up or up -d)
(b) Are these logs the same/similar to what we saw in the logs of each running container prior?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Docker compose has created a network for the containers to run in. Scroll to the top of the output to see the name. You can also check in a new terminal instance **docker network ls**. In my instance, the network name is "app_default". This command's main output is the logs of both containers mixed based on time. Each container is represented with a color.

Refresh "localhost:8081" already open in the browser. You should see the mongo-express GUI. You'll notice that the user-account DB we made as well as the users collection inside are gone. Everytime we restart the containers, everything we have done will refresh and disappear. There is no data persistence (as of now).

Q3 - Definition & Analysis

(a) What is "data persistence"?

(b) How would we introduce data persistence in our docker environment?

(c) Aside from the database and collection being gone, what else has changed back to default?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Recreate the user-account DB and the users collection.

Start the JS file with "node server.js" in the terminal (in the app directory). You should see "app listening on port 3000!" Enter "localhost:3000" in the browser to pull up the webpage. Update (change) any sections on the web page and refresh your mongo-express page while inside the users collection to see the record of the change.

To stop these containers, enter **docker-compose -f docker-compose.yaml down**. This not only stops the containers but also removes the network that the up command created. Enter **docker network ls** to be sure the network is gone.

Q4 - Results

(a) What happened in the terminal that was continuously running "docker-compose -f docker-compose.yaml up" when "docker-compose -f docker-compose.yaml down" was entered?

(b) What happens if you refresh the page loaded from "localhost:8081"? Why?

(c) What happens if you refresh the page loaded from "localhost:3000"? Why?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Modified Dockerfiles

T2. We will now package our application (including mongodb, mongo-express, node to run the server.js webpage) into its own docker container. We are going to build a docker image from our node js spec and application, and prepare it to be deployed in the docker environment.

Dockerfile=blueprint for developing docker images.

Check the Dockerfile we are referencing in Appendix C.

We will be basing our own image on a node image meaning we need node installed inside our image (first line of Dockerfile). We then define environmental variables here instead of doing so by means of a docker-compose file (doing in a compose file is actually better, but for the time being).

Next we will run a directory that will live inside a container. The copy command next copies the current folder's files into /home/app. This command executes on the host machine to copy files

inside the container image. The final line starts the app with node server.js. This only works because we are building from a node image that has node installed. CMD is an entrypoint command; you can have multiple RUN commands but only one CMD line.

This file must be named "Dockerfile" with the Docker extension downloaded on VSCode.

The Dockerfile can be seen in Appendix C. Create this now.

Q5 - Definition

(a) What is a Dockerfile?

(b) Why must it be named Dockerfile?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Enter **docker-compose -f docker-compose.yaml up -d**.

From the terminal (inside the app directory), enter **docker build -t my-app:1.0 .** (including the ".").

Q6 - Results

(a) Provide evidence that you know this command worked.

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Enter **docker images** to see the image my-app with tag 1.0.

Enter **docker run my-app:1.0**. This should work and return "app listening on port 3000!".

Enter **docker ps** in a new terminal and you should see a row for my-app:1.0, a row for mongo-express, and a row for mongo.

Access console of running container

T3. To enter the container we made and access the terminal for it, enter **docker exec -it <container id> /bin/sh**. We see the # cursor showing we are inside a container.

Q7 - Definition & Results

(a) What does -it mean/do?

(b) Where does the starting point /bin/sh take you?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Now inside the container terminal:

See the environmental variables we set, user, pass: **env**
See the directory: **ls /home/app**

Enter: **exit**

Q8 - Results

(a) How did those files end up there inside the container?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Enter **docker-compose -f docker-compose.yaml** before the next step (if not, the compose file will not be able to be put down once in a new location)

Edit the app folder until only the images folder, the node_modules folder, index.html, package.json, package-lock.json, and server.js are inside. Everything else should be in the techworld folder and out of the app subfolder.

From this change we can change our copy line in the vscode Dockerfile to copy just the app folder (everything needed) and not the entire directory.

The line should now read **COPY ./app /home/app**. (Dockerfile(2) in atts)

When you adjust the Dockerfile, you must rebuild the image after, the old image cannot be overwritten. We need to delete the image, **docker rmi <image id>**.

If it says it is unable, being used by a stopped container [id], enter **docker stop <id> followed by **docker rm <id>**, then reenter **docker rmi <image id>**.**

Enter **docker images** to ensure the image has been deleted.
Enter **cd ..** to go out of the app folder and into the techworld folder (where the updated Dockerfile is).
Enter the same build command again to rebuild the image; **docker build -t my-app:1.0 .**

Enter the same run command again to run the image; **docker run my-app:1.0**. This should work and return "app listening on port 3000!".

Q9 - Analysis

(a) What is port 3000 used for?

(b) Where is it set to use 3000?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Enter **docker ps** in a new terminal and you should see a row for my-app:1.0, a row for mongo-express, and a row for mongo.

To enter the container we made and access the terminal for it, enter **docker exec -it [container id] /bin/sh**. We see the # cursor showing we are inside a container.

Back inside the *new* container's terminal:

Enter **ls /home/app**
exit

Q10 - Analysis

(a) How would you edit the docker-compose file for the my-app image to be started at the same time as mongo and mongo-express?

(b) Devise a method and test it. Show evidence.

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Mongo and Mongo-express using docker-compose

T4. Stop and start mongo and mongo-express with docker-compose down and up commands. Make sure "my-app:1.0" is not running. Enter **docker ps** to see mongo and mongo-express. Refresh localhost:8081 to see the default database.

Create a database and collection.

Make sure no terminal instances are running "node server.js" (this includes if my-app is running). Refresh "localhost:3000" (unsuccessful). Enter **npm run start** in a terminal within the app folder directory to see "app listening on port 3000!" Refresh localhost:3000 to see if this was successful.

Q11 - Analysis

(a) How is npm used in this case?

(b) How does it know to use port 3000?

(c) How does it know to run the server.js file?

(d) How is it different from nodejs?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Enter **docker-compose -f docker-compose.yaml down**

Make changes to docker-compose.yaml to mimic Appendix D. Save.

Enter **docker-compose -f docker-compose.yaml up -d**

Make sure 3000 and 8081 are able to be loaded. Create a database and collection. Make a change in the HTML page to see the change in your mongo db as usual.

Now, to restart the containers and see if the data is persistent. Containers down then up. Reload the 8081 page to see what happens.

Q12 - Analysis

(a) What is the result when you bring the container back up?

(b) Why?

(c) What are docker volumes?

(d) Where can you find them within the OS?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Appendix A:

sudo apt update
sudo apt upgrade

Applications to be downloaded in baseline including their commands.

Shutter: [6]
    sudo add-apt-repository ppa:linuxuprising/shutter
    sudo apt update
    sudo apt install shutter
Docker: sudo apt install docker
Nodejs: sudo apt install nodejs
Npm: sudo apt install npm

Downloads techworld zip file, unzip, cd into app folder for the rest [4]

select "Code"
select zip under download source code
move the folder to home folder

npm init (while inside directory)
    This will allow the web server to be ran

Express: (after npm) npm install express --save
Npm audit fix to fix vulnerabilities

Vscode:

sudo apt-get install wget gpg

wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > packages.microsoft.gpg

sudo install -D -o root -g root -m 644 packages.microsoft.gpg /etc/apt/keyrings/packages.microsoft.gpg

```
sudo sh -c 'echo "deb [arch=amd64,arm64,armhf
signed-by=/etc/apt/keyrings/packages.microsoft.gpg]
https://packages.microsoft.com/repos/code stable main" >
/etc/apt/sources.list.d/vscode.list'

rm -f packages.microsoft.gpg

sudo apt install apt-transport-https

sudo apt update

sudo apt install code
        # or code-insiders
```

Appendix B: (docker-compose.yaml(1) in atts)

```
version: '3'
services:
  mongodb:
    image: mongo
    ports:
     - "27017:27017"
    environment:
     - MONGO_INITDB_ROOT_USERNAME=mongo
     - MONGO_INITDB_ROOT_PASSWORD=mongo
  mongo-express:
    image: mongo-express
    ports:
     - "8081:8081"
    environment:
     - ME_CONFIG_MONGODB_ADMINUSERNAME=mongo
     - ME_CONFIG_MONGODB_ADMINPASSWORD=mongo
     - ME_CONFIG_MONGODB_SERVER=mongodb
    depends_on:
     - mongodb
```

Appendix C: Dockerfile (Dockerfile(1) in atts)

```
FROM node
ENV MONGO_DB_USERNAME=mongo \MONGO_DB_PWD=mongo
```

```
RUN mkdir -p /home/app
COPY . /home/app
CMD ["node", "/home/app/server.js"]
```

Appendix D: Updated docker-compose.yaml (docker-compose.yaml(2) in atts)

```
version: '3'
services:
  mongodb:
    image: mongo
    ports:
     - "27017:27017"
    environment:
     - MONGO_INITDB_ROOT_USERNAME=mongo
     - MONGO_INITDB_ROOT_PASSWORD=mongo
    volumes:
     - mongo-data:/data/db
  mongo-express:
    image: mongo-express
    ports:
     - "8081:8081"
    environment:
     - ME_CONFIG_MONGODB_ADMINUSERNAME=mongo
     - ME_CONFIG_MONGODB_ADMINPASSWORD=mongo
     - ME_CONFIG_MONGODB_SERVER=mongodb
    depends_on:
     - mongodb
volumes:
  mongo-data:
    driver: local
```

** Note: "volumes:" is the same level as "services:"

Acknowledgements:

"TechWorld with Nana" Youtube page, specifically the video "Docker Tutorial for Beginners [FULL COURSE in 3 Hours]" [2]

Professor Salib, for his help when learning about Docker, writing this lab originally, and revising this lab.

[1] https://github.com/sindresorhus/guides/blob/main/docker-without-sudo.md
[2] (11) Docker Tutorial for Beginners [FULL COURSE in 3 Hours] - YouTube
[3] Docker Hub Container Image Library | App Containerization
[4] https://gitlab.com/nanuchi/techworld-js-docker-demo-app
[5] Nana Janashia / techworld-js-docker-demo-app · GitLab
[6] https://itsfoss.com/install-shutter-ubuntu/

Lab 2 Instructions

OpenStack Deployment

1. **Learning Objectives**
   - What is openstack?
   - What is globals.yml?
   - What is passwords.yml?
   - What is an instance?
   - What is an image?
   - What is a subnet?
   - What is a private network?
   - What is a public network?

2. **Equipment & Tools**
   - 2 (same model) APs (flashed with DD-WRT preferably)
   - Ubuntu Server 22.04 VM (preferably)
   - USB-Ethernet Adapter
   - Ethernet cables

3. **Exercises**

3.1 Exercise 1 - OpenStack Deployment

Network Diagram (after setup view)

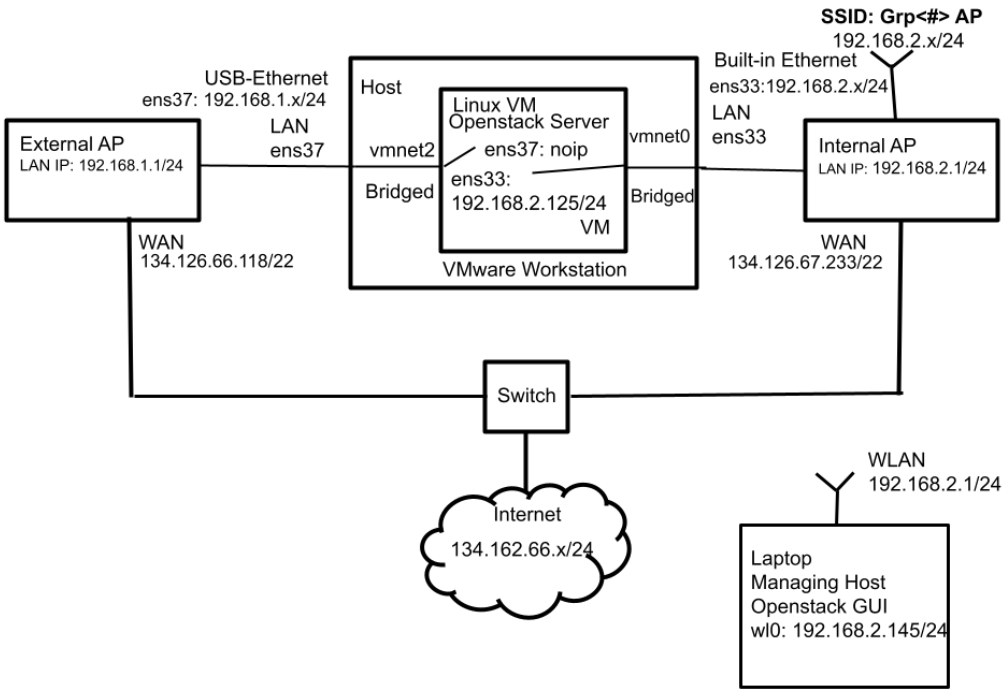


Figure 1: Network Diagram For Openstack Deployment

Step 1: Configuration

Setup

T1.

Setup your host running the VM with an ethernet cable (direct ethernet port on computer) directly to one AP LAN port (192.168.2.x). This will be your internal IP. Connect another ethernet cable (USB-Eth adapter to a USB port on computer) to a LAN port on the second AP (192.168.1.x).

Click Edit, Virtual Network Editor. We selected vmnet0 (bridged). Next to "Bridged to:", select where it says Automatic and select the interface name (on host) of the 2.x network. Then click "Add Network", keep default vmnet#, select bridged, click Save. In the same drop down box next to your new vmnet record and select the name of the interface (on host) associated with your 1.x network, then click save.

Click VM, click settings. Change Network Adapter "Network Connection" to Custom to vmnet0 with the check box not selected. Click "Add", select Network Adapter, click finish. Select Network Adapter 2, change connection to Custom vmnet2 The interfaces in your VM should now be populated with a 2.x address and (optionally) a 1.x address.

Perform an **ip a** to verify you have two interfaces on the host (find the interface names associated with the IPs just found in wired settings). On both APs, connect the WAN port to a switch connected to the internet.

Note: If not on a private network at JMU, you'll need to register the given IP address

Open /etc/netplan/00-installer* and make sure it mimics Appendix A.

Enter **sudo netplan apply** (should be no response).

We will be statically assigning an IP address to your 2.x interface. To do so, enter the webpage of this AP. Go to Services page, scroll to static leases, click add, and enter the EUI-48 address of this interface (2.x interface in VM). Enter the hostname of the machine (likely checkout), and the IP address you'll be statically assigning to this interface (192.168.2.100). Click Save then Apply settings.

Reboot virtual host to see the changes.

Q1 -

(a) What ip address is assigned to ens33 and ens37?

(b) Why are these addresses assigned to the interfaces?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Now, you should have an ens33 interface with a static IP, and an ens37 bridged interface with no assigned IP.

To check internet connectivity at this point,

ping 8.8.8.8

Installations

T2. In this Exercise we will start an openstack deployment to use for the rest of the lab. We start with preparing our VM for deployment. Make sure your VM is a ubuntu 22.04 VM that has never had a running deployment on it or docker running. We want a clear slate for this lab. Start with the commands in the terminal in the VM:

sudo apt update
sudo apt upgrade -y

** if any purple screen appears, hit enter, this should select OK and apply it **

sudo reboot

When rebooting you may be asked to select an option to download something if this appears press enter x2 (if needed).

Now that the VM has an ip address from 192.168.2.1 assigned to it we are ready to continue installing

sudo apt install git python3-dev libffi-dev gcc libssl-dev -y
** if any purple screen appears, hit enter, this should select OK and apply it **

sudo apt install python3-venv -y
** if any purple screen appears, hit enter, this should select OK and apply it **

python3 -m venv wallaby
source wallaby/bin/activate

Q2 -

(a) What does the command above do?
(b) What is wallaby?
(c) Provide evidence that the command above worked.

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Everything after this point in the lab needs to be done with the command "source wallaby/bin/activate" running.

pip install -U pip
pip install 'ansible>=6,<8'

Q3 -

(a) What is ansible?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Begin openstack install + create kolla directory
T3.

pip install git+https://opendev.org/openstack/kolla-ansible@stable/2023.1
sudo mkdir -p /etc/kolla
cd /etc
ls -al | grep kolla

If root root appears as the permissions for kolla, we need to change the permissions to checkout checkout by executing the command:

sudo chown $USER:$USER kolla

Q4 -

(a) Why do we need to change the permissions of this file?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

ls -al | grep kolla

Check permissions changed to checkout checkout.

cd kolla

Now we are going to add files that are critical for deployment to the vm and move them to a directory where we can better access them. **In this step the . after the commands containing it are VERY important**.

cp -r ~/wallaby/share/kolla-ansible/etc_examples/kolla/* .

Now execute a **ls** command
You should see two files; passwords.yml and globals.yml. If these are not present go back to the beginning of task 3 and try the commands again.

Now we must transfer the rest of the files that we will need for deployment.
cp ~/wallaby/share/kolla-ansible/ansible/inventory/all-in-one .

Now execute a **ls** command you should see to files; passwords.yml and globals.yml and the newly added all-in-one. If these are not present go back to the beginning of task 3 and try the commands again.

Now we need to check the contents of passwords.yml by using the command **sudo vim passwords.yml.** The file should have lots of blue text with lines that end with ":" that are empty.

> Note: If the file starts with lots of paragraphs and no colons this may be the same contents as the globals.yml. Check globals.yml and if it is the same, download passwords.yml from canvas and replace the one you have with the downloaded version.

Now let's modify globals.yml. Start with the command:

sudo vim globals.yml

Modify globals.yml to mimic **Appendix B**

Q5 -

(a) What did we enable in globals.yml?
(b) Why did we enable these services?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

More installation + populate passwords.yml file
T4.

cd
kolla-ansible install-deps
kolla-genpwd

sudo vim /etc/kolla/passwords.yml

There should be new information added to password.yml, Should be mostly pink/purple.

Q6 -

(a) Why did the passwords.yml file change after the 2 commands we executed?
(b) What are the contents of passwords for?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Now that you have all the files we need for deployment we can start openstack deployment. These commands for kolla-ansible are very important; there may be errors for the command. You must fix the error and redo the command that failed before continuing. Refer to Appendix C for help. We may not have every error covered in this **Appendix C** it is just common error fixes.

First we need to change the directory we are in.

cd /etc/kolla

kolla-ansible -i ./all-in-one bootstrap-servers
Output should be **localhost: ok = x failed = 0** (x meaning any number) (most important is that failed = 0)

kolla-ansible -i ./all-in-one prechecks
Output should be **localhost: ok = x failed = 0**

kolla-ansible -i ./all-in-one deploy
This command will take a while to complete.
Output should be **localhost: ok = x failed = 0**

Q7 - Open a web browser in firefox and enter the url "192.168.2.x".
- **(a) What does this url bring up? Provide evidence.**
- **(b) Why does this web page appear the way it does?**
- **(c) Provide another way to know your deployment is working.**

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Step 2: Beginning customization

Post deploy actions
T1.

Now that we have an openstack deployment we are now able to start seeing what openstack is capable of doing. We will start with the administrative perspective of openstack. We will start by installing python scripts:

pip install python-openstackclient -c https://releases.openstack.org/constraints/upper/2023.1
kolla-ansible post-deploy (to generate clouds, admin)***

Q1 -
- **(a) What does a Kolla-ansible post-deploy command do?**

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

To make sure that this command works do a **ls** command and make sure that the directory contains the newly generated files clouds and admin-openrc.sh are in the directory if these are not present restart from the beginning of the task.

We will need the information in the admin-openrc.sh file for access into openstack deployment on GUI. Start by going into the file:

sudo vim /etc/kolla/admin-openrc.sh

Inside this file there should be an input for an admin password. Copy that and save it to a txt file so you can use it to login to your openstack instance.

Q2 - Open a private browser (firefox), enter 192.168.2.x, your deployment static IP.

(a) What does this url bring up?

(b) How do you login to the openstack page?

(c) Provide evidence that your login attempt worked.

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

INIT-RUNONCE

T2.

Be sure to have done:

source wallaby/bin/activate
cd /etc/kolla

source admin-openrc.sh

Everything after this point in the lab needs to be done with the command “source admin-openrc.sh” running.

Now we want to start creating images, instances, and networks using the administration terminal in openstack. We want to start with a completely clean openstack deployment so first check that there are no images on the instance.

openstack image list

This should be empty. Now we are going to check that instances are not running on the openstack webpage.

On a web browser enter the url for your openstack instance. Login using your admin credentials, then go to the instance tab and make sure there is nothing on the screen. Check networks and images as well.

Note: Security Groups will not be empty, these are fine.

Q3 -

(a) Why does your openstack need to be empty right after deployment?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

The rest of this task includes commands to set variables that will simplify further commands.

```
KOLLA_CONFIG_PATH=${KOLLA_CONFIG_PATH:-/etc/kolla}

ARCH=$(uname -m)
CIRROS_RELEASE=${CIRROS_RELEASE:-0.6.1}
IMAGE_PATH=/opt/cache/files/
```

This is where the cirros image is taken from.

Q4 -

(a) What is Cirros?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

```
IMAGE_URL=https://github.com/cirros-dev/cirros/releases/download/${CIRROS_RELEASE}/
IMAGE=cirros-${CIRROS_RELEASE}-${ARCH}-disk.img
IMAGE_NAME=cirros
IMAGE_TYPE=linux
```

These commands contain the network details for the private network's subnet

```
IP_VERSION=${IP_VERSION:-4}
DEMO_NET_CIDR=${DEMO_NET_CIDR:-'10.0.0.0/24'}
DEMO_NET_GATEWAY=${DEMO_NET_GATEWAY:-'10.0.0.1'}
DEMO_NET_DNS=${DEMO_NET_DNS:-'8.8.8.8'}
```

These commands contain the details for the public network's subnet

```
ENABLE_EXT_NET=${ENABLE_EXT_NET:-1}
EXT_NET_CIDR=${EXT_NET_CIDR:-'192.168.1.0/24'}
EXT_NET_RANGE=${EXT_NET_RANGE:-'start=192.168.1.150, end=192.168.1.199'}
EXT_NET_GATEWAY=${EXT_NET_GATEWAY:-'192.168.1.1'}

export OS_CLIENT_CONFIG_FILE=${KOLLA_CONFIG_PATH}/clouds.yaml
export OS_CLOUD=kolla-admin
```

This turns the image file downloaded earlier into an OpenStack glance image in your deployment.

```
IMAGE_PATH='./'
curl --fail -L -o ${IMAGE_PATH}/${IMAGE} ${IMAGE_URL}/${IMAGE}
```

Network, router, image, instance creation

T3.

Create an image. [4]

openstack image create --disk-format qcow2 --container-format bare --public --property os_type=${IMAGE_TYPE} --file ${IMAGE_PATH}/${IMAGE} ${IMAGE_NAME}

openstack image list

Q5 -

(a) What does each part of the "openstack Image create" command do?

(b) What is QCOW2 format?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

We will create a router to connect the public and private networks created.

openstack router create demo-router
SUBNET_CREATE_EXTRA=""

Now, to configure the network in your deployment.

First, create a private network. [3]

openstack network create demo-net

Then, a subnet for this network using previously defined variables.

openstack subnet create --ip-version ${IP_VERSION} --subnet-range ${DEMO_NET_CIDR} --network demo-net --gateway ${DEMO_NET_GATEWAY} --dns-nameserver ${DEMO_NET_DNS} ${SUBNET_CREATE_EXTRA} demo-subnet

Q6-

(a) What does each part of the "openstack subnet create" command do?

(b) What is the subnet of the network?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Then add the private network as an interface to the router.

openstack router add subnet demo-router demo-subnet

Create your public network.

openstack network create --external --provider-physical-network physnet1 --provider-network-type flat public1

openstack subnet create --no-dhcp --ip-version ${IP_VERSION} --network public --subnet-range ${EXT_NET_CIDR} --gateway ${EXT_NET_GATEWAY} public1-subnet

Set the external gateway of the router as the public network just created.

openstack router set --external-gateway public1 demo-router

ADMIN_PROJECT_ID=$(openstack project list | awk '/ admin / {print $2}')

ADMIN_SEC_GROUP=$(openstack security group list --project ${ADMIN_PROJECT_ID} | awk '/ default / {print $2}')

Create security rules to be applied to the default security project in OpenStack. [1]

openstack security group rule create --ingress --ethertype IPv${IP_VERSION} --protocol icmp ${ADMIN_SEC_GROUP}

openstack security group rule create --ingress --ethertype IPv${IP_VERSION} --protocol tcp --dst-port 22 ${ADMIN_SEC_GROUP}

openstack security group rule create --ingress --ethertype IPv${IP_VERSION} --protocol tcp --dst-port 8000 ${ADMIN_SEC_GROUP}

openstack security group rule create --ingress --ethertype IPv${IP_VERSION} --protocol tcp --dst-port 8080 ${ADMIN_SEC_GROUP}

Q7 -

(a) What does each rule allow?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Create a keypair

ssh-keygen -t ecdsa -N '' -f ~/.ssh/id_ecdsa

openstack keypair create --public-key ~/.ssh/id_ecdsa.pub mykey

Change the quota settings on the amount of instances, cores, and ram.

openstack quota set --instances 40 --force ${ADMIN_PROJECT_ID}

openstack quota set --cores 40 --force ${ADMIN_PROJECT_ID}

openstack quota set --ram 96000 --force ${ADMIN_PROJECT_ID}

Q8 -

(a) Why is it necessary/helpful to do this?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Now, create five flavors to be used for instances in OpenStack.

openstack flavor create --id 1 --ram 512 --disk 1 --vcpus 1 m1.tiny

openstack flavor create --id 2 --ram 2048 --disk 20 --vcpus 1 m1.small

openstack flavor create --id 3 --ram 4096 --disk 40 --vcpus 2 m1.medium

openstack flavor create --id 4 --ram 8192 --disk 80 --vcpus 4 m1.large

openstack flavor create --id 5 --ram 16384 --disk 160 --vcpus 8 m1.xlarge

Q9 -

(a) What is a flavor?

(b) How are they used in OpenStack?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Finally, create a VM instance using the image created earlier. [2]

openstack server create --image cirros --flavor m1.tiny --key-name mykey --network demo-net demo1

Q10 -

(a) What do you see in your OpenStack network?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Go to your openstack web page and select the instances tab.

Click on the instance and go to the console tab. Login to the instance (the login information is in the console).

ping 8.8.8.8

Q11 -

(a) Are you successful?

(b) Why or why not?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

In order to enable your OpenStack server to reach the internet, you must do the following.

Enter the webpage of your 2.x AP. Click on NAT/QoS. Click on Port Range Forwarding. Click Add. Enter "22" under start, "22" under end, Both under protocol, 192.168.1.1 under IP, and check in enable.

In order to ensure connectivity from the inside of the newly made instance, we must send a successful ping to 8.8.8.8. Enter the console of the instance (on the GUI). Since the security group rules are in place, this should be successful. Ping 8.8.8.8 until successful.

We will add a floating IP to the instance.

Now, in order to have connectivity from the outside in, we must add a floating IP. First, click floating IP in the network section. Click allocate floating IP, enter public1 (public network) as the source and click allocate. Now you should have an IP available to allocate to an instance. Click compute and then instances. There should be a button "Associate floating IP", if not, press the down arrow on the right side of the instance and click Associate floating IP. Select the floating IP that we have just allocated from the public network. Assign this IP to this instance. Now, in a separate terminal (on host), ping the floating IP and see if you are successful.

Using this floating IP, we will access the instance from the outside.

Now, what is most helpful from the floating IP, is to ssh from the same separate terminal you have just used into the instance. Enter "ssh {hostname}@{ip address}. Enter yes, then the password. And you have successful connectivity into the instance.

Q12 -

(a) What rule allows this process to work?

Provide evidence. Reference the evidence with specific description of the relevant information. Provide citations to external sources.

Appendix A

Instance names may be different depending on your instance

```
# This is the network config written by ‘subiquity’
network:
  version: 2
  renderer: networkd
  ethernets:
    ens33:
      dhcp4: true
    ens38:
      dhcp4: false
      addresses: []
```

Appendix B

```
**Ansible options**
`workaround_ansible_issue_8743: yes

**Kolla options**
kolla_base_distro: "ubuntu"
openstack_release: "2023.1"
kolla_internal_vip_address: "192.168.2.x(125)"   ← set to your static IP
```

Neutron - Networking options
network_interface: "ens33" ← might be different if you did not change your interface names
neutron_external_interface: "ens38 The interfaces in your VM should now be populated with a 2.x address and (optionally) a 1.x address. " ← might be different

OpenStack options
enable_haproxy: "no"
enable_etcd: "yes"
enable_kuryr: "yes"

Nova - Compute Options
nova_compute_virt_type: "qemu"

```

## Appendix C

Fail resolutions

If fail at ensure localhost in /etc/hosts, enter "sudo kolla-ansible -i ./all-in-one bootstrap-servers", enter pass, followed by "kolla-ansible -i ./all-in-one bootstrap-servers", should continue

If fail at checking docker version (sudo: need a password), enter "sudo kolla-ansible -i ./all-in-one prechecks", enter pass, followed by "kolla-ansible -i ./all-in-one prechecks", should continue

openstack subnet create --no-dhcp --network public1 --subnet-range 192.168.1.0/24 --gateway 192.168.1.1 public1-subnet

If this command returns errors, go to deployment page in firefox, click networks, click the drop arrow on public1 network, click create subnet, enter the following details

Subnet Name: public1-subnet
Network Address: 192.168.1.0/24
IP Version: IPv4
Gateway IP: 192.168.1.1
Disable Gateway: unchecked

Click Next

Enable DHCP: unchecked
Allocation Pools: 192.168.1.150,192.168.1.199
```

[1] OpenStack Docs: security group rule
[2] OpenStack Docs: Launch instances
[3] OpenStack Docs: Create and manage networks
[4] OpenStack Docs: Disk and Container Formats

Lab 3 Instructions

Admin, Users, Passwords, Projects

1. **Learning Objectives**
   - What is openstack?
   - What is an Administrator?
   - What is User?
   - What is a user capability?
   - What is an administrator capability?
   - What is a Project?
   - What is a member?
2. **Equipment & Tools**
   - 2 (same model) APs (flashed with DD-WRT preferably)
   - Ubuntu Server 22.04 VM (preferably)
   - USB-Ethernet Adapter
   - Ethernet cables

Network Diagram (after setup view)

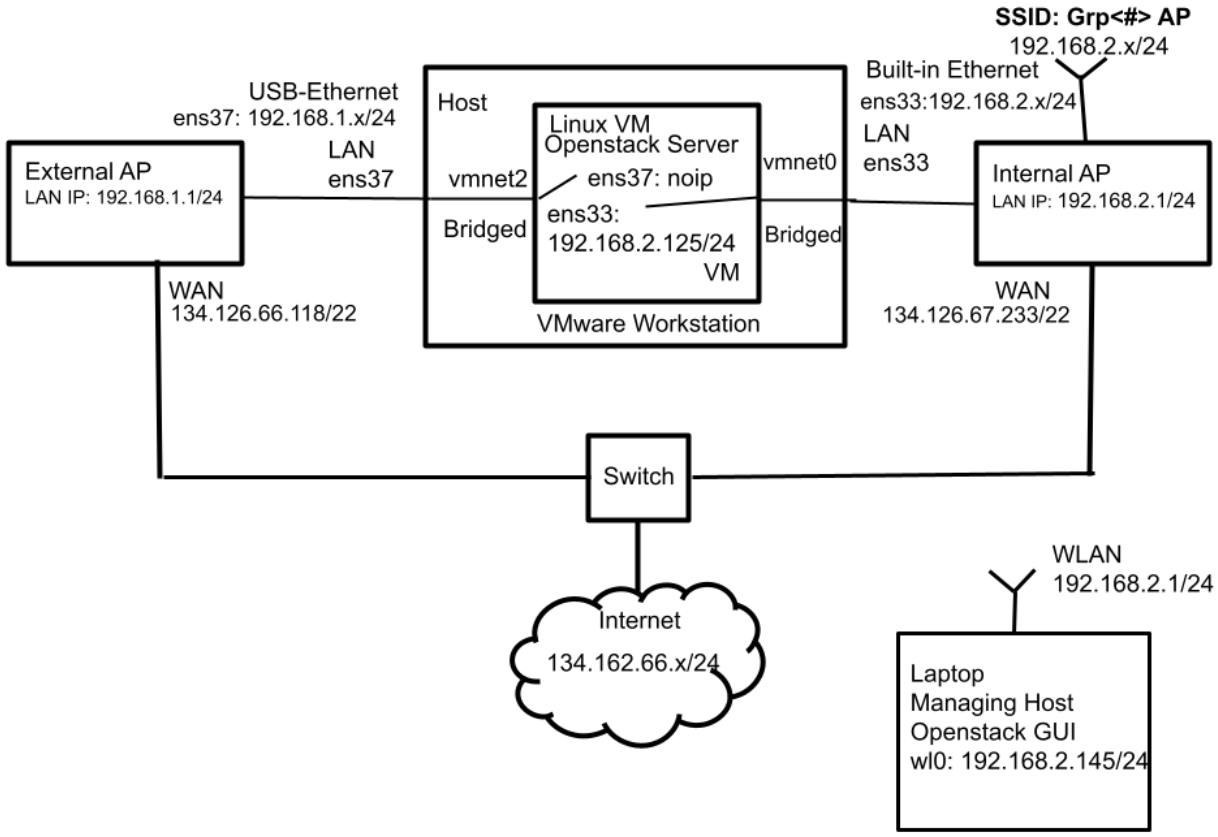


Figure 1: Network Diagram For Openstack Deployment

3. Exercises

3.1 Exercise 1 - Create a User

Step 1:

T1.

source wallaby/bin/activate
cd /etc/kolla
source admin-openrc.sh

To begin exploration, we created a new project. We were unaware at the time, but this was the cornerstone of this section and essentially creates a whole new openstack dashboard.

openstack project list
openstack project create {name} --domain default

{name} = the name you want for your project

We then added a new user. This contained a username, email, password, a primary project assignment, and a role. We added the project we created previously as the primary project for this user.

openstack user create --project {name} --password {password} {username}
{name} = the name you assigned to your project
{password} = the password you are assigning to your new user
{username} = the username assigning to your new user

The different roles also became an important part of the section of the project as they entirely control what each user can do. We then added the role "_member_". We were unsure what the difference was between the "member" role and the "_member_" role at the start but shortly discovered there was not much difference besides the "member" role had the image from our admin project already loaded but the other had nothing.

openstack role list
openstack role add --user {username} --project {name} {member}
{username} = the username assigning to your new user
{name} = the name you assigned to your project
{role} = choose from list above, likely admin or member

All new projects came loaded with a public network (with no subnet attached and no internet), as well as the security groups that also came loaded in our original admin project.

We eventually changed the role of this user in this project to the "member" role and used that in all future users. At this point, we logged in as a new user for the first time and saw the new dashboard on the openstack web browser.

Username: {new username}
Password: {new password}

This showed us a different view than we were used to as now we were members and not admins/heat stack owners. We lost access to the admin section in the left side navigation bar, and a few other tabs in the Identity section on the navigation bar. All of the instances we had created in the admin project were gone, as well as the floating IPs, router, networks, subnets, and assigned floating IPs. We eventually made another project and another user with this second project as its primary project. This user had a member role on this project.

T2.
Now that we have explored logging in as different users, seen new projects, experimented with roles, we wanted to make the new projects functional.

We also experimented with adding new admin users and reader users. As a reader, you are able to see all that a member can, yet you cannot add or delete anything (router, subnet, network, instance). An admin user has admin access to the project and regains the admin section on the navigation bar. Admin also has the ability to delete instances, routers, subnets, and networks from the system as well as see what other users have access to the project. A _heat_stack_user_ is the highest amount of permissions a user can have. Heat_stack_users have all the same permissions of an admin but also have the ability to assign members to a project, see any project happening on the openstack deployment and change members roles including an admins role.

We focused now on one project in order to bring functionality. We added another network to act as an internal network and added a subnet to it.

openstack network create {network name} --project {project name}

openstack subnet create {subnet0} --project {project1} --network {net1} --subnet-range {10.0.1.0/24}

We then added an instance to this network.

openstack flavor list
openstack image list
openstack keypair list
openstack security group rule list default

openstack network list
openstack server create --image {cirros} --flavor {m1.tiny} --key-name {mykey} --network {net1} {demo1}

**from what we can tell, you cannot add a project tag to an instance creation command, and therefore cannot create an instance for project1 project from terminal, only GUI

T3.

We attached a router in between the public and internal networks.

#creating router inside project1
openstack router create routerA --project project1

#attaching router to public network
openstack router set routerA --external-gateway public1

#attaching router to new network
openstack router add subnet routerA subnet1

From here, we began to test connectivity from inside the instance. We now needed to create a subnet attached to our public network. On a project that's only user has the role "member", we are unable to do this. We found that a project needs a user with the admin role in order to do this. To combat this, we added the admin user to the project as an admin role, and this added the subnet (associated with our 1.x AP) that was configured in the admin project.

openstack role add --user {username} --project {name} {role}

We tried to ping 8.8.8.8, and were unsuccessful. We were not sure why, and after a lot of troubleshooting, we found that once the four additional security rules were added, we were able to access the internet. Another important thing to check was the IPs of the internal network's subnet to be sure the gateway ends with .1.

openstack security group rule create --ingress --ethertype IPv4 --protocol icmp --project project1 {security-group-ID}

openstack security group rule create --ingress --ethertype IPv4 --protocol tcp --project project1 --dst-port 22 {security-group-ID}

openstack security group rule create --ingress --ethertype IPv4 --protocol tcp --project project1 --dst-port 8000 {security-group-ID}

openstack security group rule create --ingress --ethertype IPv4 --protocol tcp --project project1 --dst-port 8080 {security-group-ID}

https://docs.openstack.org/ocata/user-guide/cli-create-and-manage-networks.html

Step 2: Openstack Containers

T1.

First go into your ssh openstack client

Source wallaby/bin/activate

cd /etc/kolla

Source admin-openrc.sh

Now that you are in your openstack deployment you need to access the globals.yml.

One inside uncomment

OpenStack options

enable_openstack_core: "yes"

enable_glance: "{{ enable_openstack_core | bool }}"

enable_haproxy: "no"

enable_keystone: "{{ enable_openstack_core | bool }}"

enable_neutron: "{{ enable_openstack_core | bool }}"

enable_nova: "{{ enable_openstack_core | bool }}"

Then save the document and reconfigure the openstack deployment.

kolla-ansible -i ./all-in-one reconfigure

Once this command has worked successfully log on to your openstack webpage. You should see a new tab in the side bar that is labeled “container infra” and “Object store”. We will now create a container and start messing around with opentacks container capabilities.

T2. ssh to containers

We now have containers that are able to run. Since the consoles even when they work do very very little. But we are able to allocate floating Ips to them so our thought process naturally brought us to the idea of ssh, since most of the containers do not have a terminal prompt already installed. We started this experiment with a container made from a Ubuntu image. We did this because this container without us having to change any default setting already came with a terminal. We then used the allocated ip address to this container, went into our wireless client and executed the command “ssh (image name of container)@floatingIP”. We were able to connect to the container and were given a prompt to enter the password to complete the connection but then we were met with a realization that we didn't know the password to the container. We then went back into the ubuntu container and found that there was no way to find the current password of the user. We reset the password so that we knew what it was and made sure the ssh server was running. Then we went back to the wireless client and the prompt popped up. We entered the new password and it told us the wrong password on all three prompts and after the last attempt said private key authentication needed.

We figured out that the container will not allow public key authentication and we have no way of using private key authentication so ssh was not a liable option. `

T3. VOLUMES

Begins with adding a second hard disk into the box of the computer
You can check for the new disk by entering "sudo fdisk -l" before and after putting it in

Check which /dev/sdx is new $<$-- x = a, b, c
    In this case, we are using /dev/sdb
Enter "sudo pvcreate /dev/sdb"
    Be sure the disk is unmounted. If mounted enter "sudo su"
Enter "mount /dev/sdb1 /vols" $<$-- vols is just a created directory
    If response is "mount: /vols: /dev/sdb1 already mounted on /vols."
    Enter "umount /dev/sdb1 /vols"
Enter "sudo vgcreate cinder-volumes /dev/sdb1"
Enter "y" to the wipe it? question

In globals.yml
Uncomment...

    cinder_volume_group: "cinder-volumes"
    enable_cinder: "yes"
    enable_cinder_backend_lvm: "yes"
    cinder_backend_ceph: "no"

Enter "kolla-ansible -i ./all-in-one reconfigure"
Enter "sudo reboot" once reconfiguration is finished

On Openstack webpage
Click Volumes
Click Create Volume
Enter name, leave the rest blank

Click Compute, click Instances
Drop down on a running instance
Click Attach Volume
Select the volume just made

Check for "in-use" on Volumes page

In SSH Terminal
Enter "lsblk"
Should see lines regarding "cinder-volumes" underneath \ref{fig:lsblk-vols}

Step 3:
T1. PORT FORWARDING

First, connect to your 2.x AP wired. Access the webpage and find the WAN IP address (134.126..) through Status, Sys-Info. Click the Security tab, and scroll to the Block WAN Requests section. Uncheck "Block Anonymous WAN Requests (ping)". Go to the terminal and ping your WAN IP address. Click the NAT/QoS tab, and hit "Add" below the Forwards section. You will be entering two entries for port forwarding to work by webpage and by terminal ssh. In the first entry; Application: "OpenStack1", Protocol: Both, Source Net: 0.0.0.0/0, Port from: 2200, IP Address: <the address used to access your deployment, 2.x static IP>, Port to: 22, Enable: checked in. In the second entry; Application: "OpenStack2", Protocol: Both, Source Net: 134.126.0.0/0, Port from: 80, IP Address: <the address used to access your deployment, 2.x static IP>, Port to: 80, Enable: checked in. Hit save. Hit apply settings. Remove the wired connection to your AP2. Try to ssh from the terminal with command "ssh checkout@134.126.x.x -p 2200". This should be successful. Now, try to access OpenStack from a browser by entering your WAN IP address. This may take a moment, but should be successful. Finally, to allow remote access to the AP webpage without wired/wireless connection, you must go to the Administration tab, Remote Access section, select Enable

T2. MONGO + MONGO EXPRESS

Add these security rules to your default security group

☐	Ingress	IPv4	TCP	22 (SSH) - 27017	0.0.0.0/0
☐	Ingress	IPv4	TCP	22 (SSH) - 27018	0.0.0.0/0
☐	Ingress	IPv4	TCP	22 (SSH) - 27019	0.0.0.0/0

Ssh to openstack terminal from laptop
On openstack webpage create mongo container
CONFIGURATION MONGO
Info tab
- Name: mongodb
- Image: mongo
- Image Driver: Docker hub
- Image Pull Policy: If not present
- Command: mongod
- Check in start container after creation

Spec tab

Hostname: mongodb

Networks tab

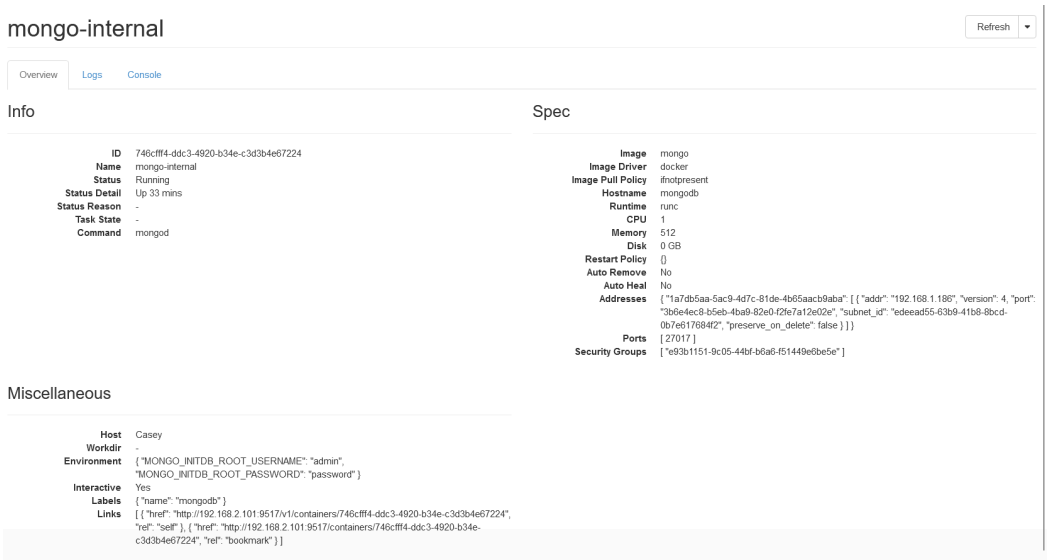
Public1

Miscellaneous tab

Environment Variables: MONGO_INITDB_ROOT_USERNAME=admin, MONGO_INITDB_ROOT_PASSWORD=password (no space between 2nd one and comma)

Labels tab

Labels: name=mongodb



On openstack webpage create mongo-express container

T3. CONFIGURATION MONGO-EXPRESS

Info tab

Name: mongo-express

Image: mongo-express

Image Driver: Docker hub

Image Pull Policy: If not present

Command: mongo-express

Check in start container after creation

Spec tab

Hostname: mongo-express

Networks tab

Public1

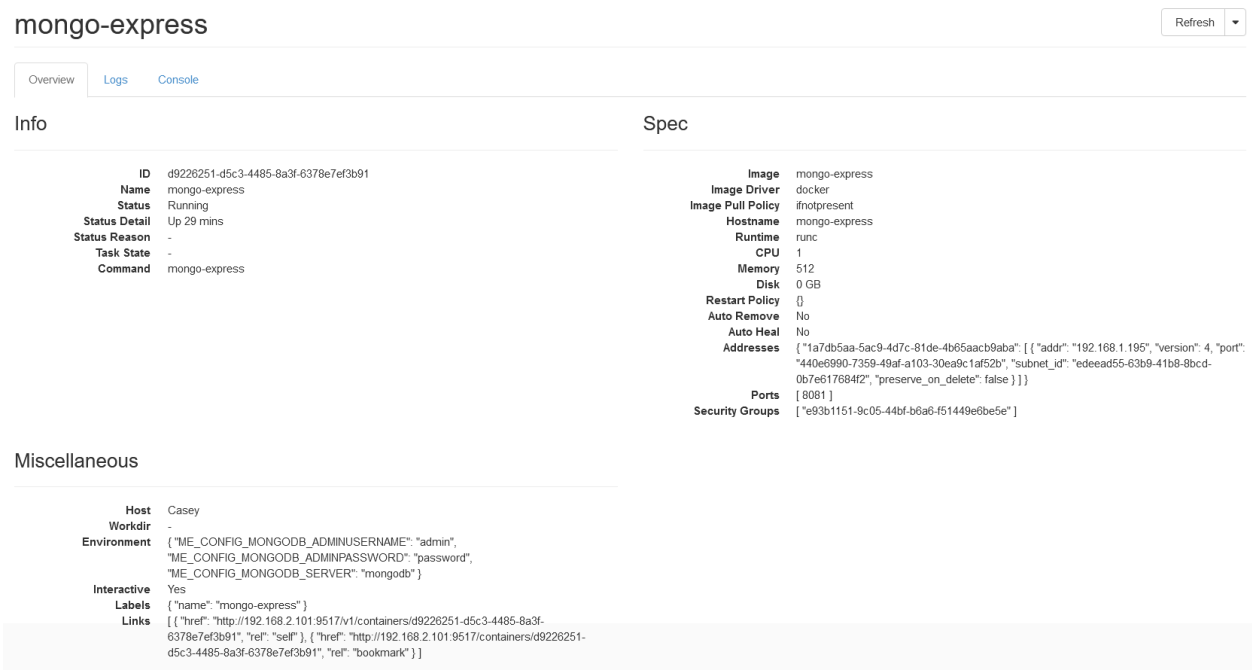
Miscellaneous tab

Environment Variables: ME_CONFIG_MONGODB_ADMINUSERNAME=admin, ME_CONFIG_MONGODB_ADMINPASSWORD=password,

ME_CONFIG_MONGODB_SERVER=mongodb (no space between 2nd one and comma and 3rd one and comma)

Labels tab

Labels: name=mongo-express



Important to note, passwords and usernames match, hostnames are set in spec, connected to public network

In Overview of your new mongo-express container, find the IP address assigned from the public network
In a webpage, enter <express IP address>:8081 to see webpage, you will be asked for user and pass
This is not the same as the user and pass set as environmental variables

Go to ssh terminal
Enter docker ps (might need sudo)
Copy the container ID of the express container created in openstack (not created by docker in terminal)
Enter docker logs <container ID>
Credentials should be shown in red (admin:pass)
Enter on express webpage, should be logged in and see databases

Appendix F - Lab Reports

Lab 1 Report

James Madison University

Information Technology (IT) Program

Capstone Project Labs

Fall 2023

Lab 1 - Docker creation + exploration lab

Group 1

Wednesday 13th December, 2023

22:54:17

Author(s):
Casey Alexander
Katherine Botticelli

Submitted to:
Dr. Emil Salib

Honor Pledge: I have neither given nor received help on his lab that violates the spirit of the JMU Honor Code.

Casey Alexander	Katherine Botticelli
Signature	*Signature*
Date	*Date*

Execution: Here is my percentage of executing the the hands-on portion of the lab. It is expected that each partner execute 100%.

C. Alexander (100%)	K. Botticelli (100%)
Signature	*Signature*
Date	*Date*

Contributions: Here is my Lab Report Contribution Percentages for this lab. It is expected that the sum of the contributions of the group members is 100%.

C. Alexander (50%)	K. Botticelli (50%)
Signature	*Signature*
Date	*Date*

Rubric Check: I have read and complied with every item listed in this lab's rubric as posted on Canvas.

C. Alexander (Yes)	K. Botticelli (Yes)
Signature	*Signature*
Date	*Date*

Contents

List of Figures

List of Tables

List of Attachments & Screenshots

Attachments

1 /Dockerfile(2)
2 /docker-compose.yaml(1)
3 /Dockerfile(1)
4 /docker-compose.yaml(2)

L1E1S1T3

5 /E1S1T3-mongo-express-change-mongo-logs.txt

Screenshots

1 /L1E1S1N1.png
2 /title-seal.jpg

L1E1S1T1

3 /L1E1S1T1-13.png
4 /L1E1S1T1-1.png
5 /L1E1S1T1-18.png
6 /L1E1S1T1-7.png
7 /L1E1S1T1-12.png
8 /L1E1S1T1-2.png
9 /L1E1S1T1-26.png
10 /L1E1S1T1-3.png
11 /L1E1S1T1-27.png
12 /L1E1S1T1-14.png
13 /L1E1S1T1-15.png
14 /L1E1S1T1-8.png
15 /L1E1S1T1-21.png
16 /L1E1S1T1-32.png
17 /L1E1S1T1-28.png
18 /L1E1S1T1-16.png
19 /L1E1S1T1-23.png
20 /L1E1S1T1-31.png
21 /L1E1S1T1-5.png
22 /L1E1S1T1-24.png

```
23          /L1E1S1T1-34.png
24          /L1E1S1T1-6.png
25          /L1E1S1T1-22.png
26          /L1E1S1T1-33.png
27          /L1E1S1T1-4.png
28          /L1E1S1T1-29.png
29          /L1E1S1T1-20.png
30          /L1E1S1T1-19.png
31          /L1E1S1T1-9.png
32          /L1E1S1T1-10.png
33          /L1E1S1T1-25.png
34          /L1E1S1T1-17.png
35          /L1E1S1T1-30.png
36          /L1E1S1T1-11.png
    L1E1S1T2
37          /L1E1S1T2-2.png
38          /L1E1S1T2-6.png
39          /L1E1S1T2-7.png
40          /L1E1S1T2-1.png
41          /L1E1S1T2-8.png
42          /L1E1S1T2-5.png
43          /L1E1S1T2-3.png
44          /L1E1S1T2-4.png
    L1E1S1T3
45          /L1E1S1T3-18.png
46          /L1E1S1T3-1.png
47          /L1E1S1T3-17.png
48          /L1E1S1T3-16.png
49          /L1E1S1T3-14.png
50          /L1E1S1T3-2.png
51          /L1E1S1T3-19.png
52          /L1E1S1T3-11.png
53          /L1E1S1T3-5.png
54          /L1E1S1T3-9.png
55          /L1E1S1T3-7.png
56          /L1E1S1T3-13.png
57          /L1E1S1T3-4.png
58          /L1E1S1T3-8.png
59          /L1E1S1T3-12.png
60          /L1E1S1T3-10.png
61          /L1E1S1T3-3.png
62          /L1E1S1T3-15.png
63          /L1E1S1T3-6.png
    L1E1S2T1
64          /L1E1S2T1-8.png
65          /L1E1S2T1-15.png
66          /L1E1S2T1-9.png
67          /L1E1S2T1-11.png
68          /L1E1S2T1-13.png
69          /L1E1S2T1-19.png
```

```
70          /L1E1S2T1-14.png
71          /L1E1S2T1-4.png
72          /L1E1S2T1-7.png
73          /L1E1S2T1-17.png
74          /L1E1S2T1-5.png
75          /L1E1S2T1-3.png
76          /L1E1S2T1-6.png
77          /L1E1S2T1-18.png
78          /L1E1S2T1-2.png
79          /L1E1S2T1-1.png
80          /L1E1S2T1-10.png
81          /L1E1S2T1-16.png
82          /L1E1S2T1-20.png
83          /L1E1S2T1-12.png
    L1E1S2T2
84          /L1E1S2T2-3.png
85          /L1E1S2T2-7.png
86          /L1E1S2T2-1.png
87          /L1E1S2T2-5.png
88          /L1E1S2T2-4.png
89          /L1E1S2T2-8.png
90          /L1E1S2T2-6.png
91          /L1E1S2T2-2.png
    L1E1S2T3
92          /L1E1S2T3-11.png
93          /L1E1S2T3-2.png
94          /L1E1S2T3-13.png
95          /L1E1S2T3-5.png
96          /L1E1S2T3-22.png
97          /L1E1S2T3-12.png
98          /L1E1S2T3-19.png
99          /L1E1S2T3-14.png
100          /L1E1S2T3-3.png
101          /L1E1S2T3-21.png
102          /L1E1S2T3-6.png
103          /L1E1S2T3-17.png
104          /L1E1S2T3-18.png
105          /L1E1S2T3-4.png
106          /L1E1S2T3-1.png
107          /L1E1S2T3-10.png
108          /L1E1S2T3-23.png
109          /L1E1S2T3-20.png
110          /L1E1S2T3-9.png
111          /L1E1S2T3-7.png
112          /L1E1S2T3-16.png
113          /L1E1S2T3-15.png
    L1E1S2T4
114          /L1E1S2T4-9.png
115          /L1E1S2T4-4.png
116          /L1E1S2T4-14.png
```

```
117         /L1E1S2T4-24.png
118         /L1E1S2T4-23.png
119         /L1E1S2T4-11.png
120         /L1E1S2T4-10.png
121         /L1E1S2T4-8.png
122         /L1E1S2T4-12.png
123         /L1E1S2T4-2.png
124         /L1E1S2T4-3.png
125         /L1E1S2T4-5.png
126         /L1E1S2T4-17.png
127         /L1E1S2T4-6.png
128         /L1E1S2T4-1.png
129         /L1E1S2T4-18.png
130         /L1E1S2T4-15.png
131         /L1E1S2T4-25.png
132         /L1E1S2T4-22.png
133         /L1E1S2T4-16.png
134         /L1E1S2T4-7.png
135         /L1E1S2T4-20.png
136         /L1E1S2T4-19.png
137         /L1E1S2T4-13.png
138         /L1E1S2T4-21.png
```

1 Introduction

Docker is a very powerful and important tool to learn as a technology professional. The value gained from this lab is not strictly in the execution and understanding of the topics themselves, but in the baseline knowledge you will gain to propel you through OpenStack. Docker containers are valuable because they are essentially compressed virtual machines, running only the service you are looking to use, and relying on the OS of the machine or platform in use. Learning Docker commands helps when troubleshooting OpenStack because when enabling containers, it is important to understand the the view of containers in the terminal and in the deployment browser. The "kolla" of "kolla-ansible" deployment, what we use when deploying OpenStack, relies on pre-configured containers that help run through deployment [1], so this is another reason why it is important to know Docker before attempting OpenStack.

In this lab, we learned all about Docker. We learned all the commands, how to create containers, an example of how to practically use a container, and multiple ways to bring containers up. Learning about docker-compose was very useful because after seeing containers brought up individually, you can really see how helpful and important it can be to keep all projects in line. We also learned about nodejs and npm, two services to runa javascript file or script. We also learned about creating an image from our own Dockerfile which would be very helpful in real-life use.

2 Network Diagrams

2.1 Exercise 1: Docker container setup

2.1.1 Step 1: MongoDB and Mongo-Express setup

Figure 1 represents the network arrangement constructed for Exercise 1, Step 1 tasks.

2.1.2 Step 2: Docker-compose

Figure 1 represents the network arrangement constructed for Exercise 1, Step 2 tasks.

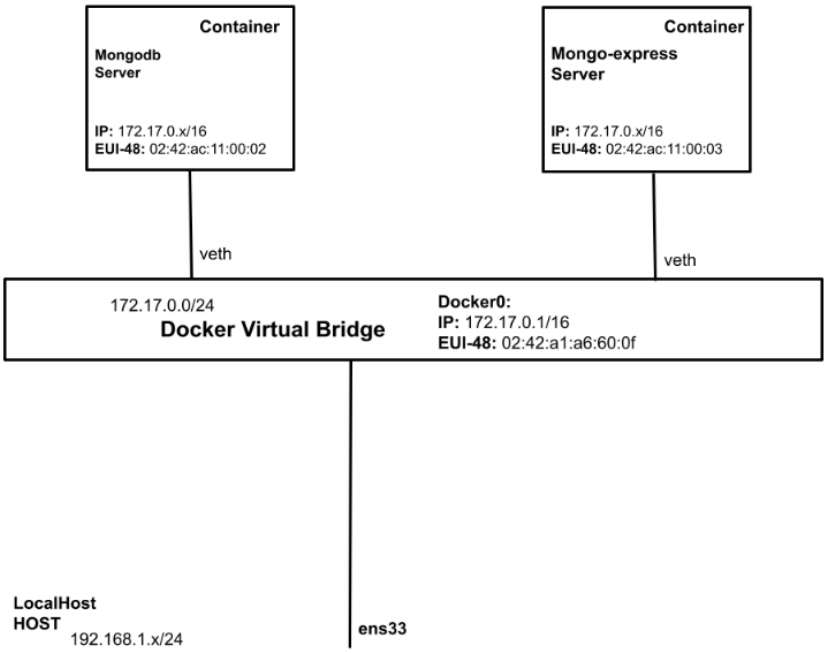


Figure 1: Net Diagram

3 Lab Exercises: Results & Analysis

3.1 Exercise 1 - Docker container setup

3.1.1 Step 1: MongoDB and Mongo-Express setup

Figure 1 represents the network we constructed to perform all or some of the tasks in Exercise 1, Step 1.

Prepare for MongoDB and Mongo-Express Containers

T1: Done. See the evidence (screenshots in images/L1E1S1T1 subfolder).

Q1: (a) What is MongoDB?
(b) What is Mongo-express?
(c) What is the default tag for images? How do you specify a tag?
(d) What does the "pull" command in Docker do?
(e) Where are Mongo and Mongo-express being pulled from?
(f) What is the image of "hello-world"?

A1: (a) MongoDB is a NoSQL database program that can store and create databases [2]. "MongoDB always stored of high volume data and can support many programming languages [3]".
(b) Mongo-express is an interactive web page that allows you to manage a MongoDB database in a web browser. This allows you to create, delete, and modify databases on your MongoDB server [3].
(c) The default tag for docker images is ":latest" as seen in Figure 2. This means that we are getting the latest version of the image that we are pulling from docker. We can modify this tag by instead of just pulling the docker image by its name such as "MongoDB" we can pull it using a specific version like "mongodb: 3.1" this will pull the specific version we are looking for not just the latest version [4].
(d) "Docker pull command downloads Docker images from the internet[5]". This means that when we pull we are essential doing the same thing a wget command in Docker to get the image form the Internet.
(e) The MongoDB and the Mongo-express images are pulled from Github Packages that are stored on Docker hub [5].
(f) The image pulled by "hello-world" is a linux based image created to allow testing in Docker [6].

```
checkout@ubuntu:~$ docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
cbe3537751ce: Pull complete
a80d99d2ce19: Pull complete
cdb44dc221f3: Pull complete
52cece2eeeb6: Pull complete
9484737e86c4: Pull complete
43ad935b75c0: Pull complete
a3ac6a8edff6: Pull complete
90580617c703: Pull complete
3c932f959341: Pull complete
Digest: sha256:b679b96ec8a2692ebb6f7622b9097974c1f751b413b3db5a0629a244ae2c
6950
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
```

Figure 2: Pulling Mongo from docker hub with tag "latest"

Q2: (a) What does "docker images" return now? Show evidence.

A2: The command "docker images" returns all of the images we have pulled so far which are node, mongo, mongo-express and hello-world as seen in Figure 3.

```
checkout@ubuntu:~$ docker images
REPOSITORY      TAG       IMAGE ID       CREATED       SIZE
node            latest    b866e35a0dc4   5 days ago    1.1GB
mongo           latest    5acb2131d51f   9 days ago    757MB
mongo-express   latest    bbc568e1f48f   9 days ago    202MB
hello-world     latest    feb5d9fea6a5   2 years ago   13.3kB
```

Figure 3: Checking current docker images

Q3: (a) What networks were there upon default?
(b) What are they for?
(c) Describe driver and scope.

A3: (a) The default networks were "default", "host" and "none" as seen in Figure 4.
(b) The default network is for Docker containers to be assigned IP addresses so they are able to communicate with each other inside of docker. The host network is what the host machine, or VM in this case, is running on, and the none network is a default placement of a network with nothing inside of it.
(c) In Docker, there are network drivers running behind the scenes of every network which can be seen in Figure 5. A bridge driver (with bridge network) is the default driver applied and it installs rules in the host machine so containers on different bridge networks cannot communicate with each other [7]. A host network driver (with the host network) means the container's network is not isolated from the host. These containers can be seen from the host [8]. The null driver goes along with the "none" network and does not do anything [9]. All

3, bridge, host, and null drivers are local scope drivers, meaning they only provide their service on a single host (non-overlay) [10].

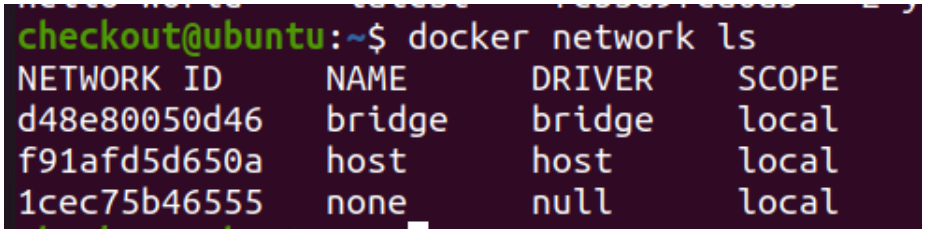


Figure 4: Checking default docker networks

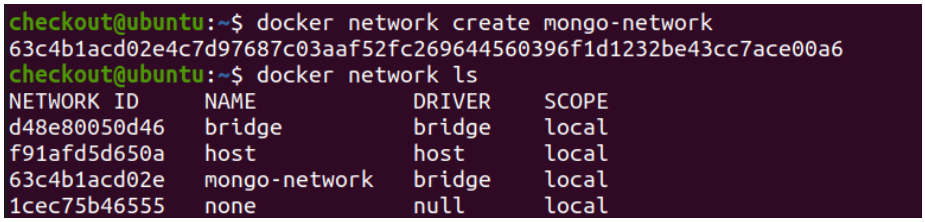


Figure 5: Creating new docker network

Running Docker containers

T2: Done. See the evidence (screenshots in images/L1E1S1T2 subfolder).

Q4: (a) What is the difference between a docker image and a docker container?

A4: (a) A docker image is pulled from Docker hub or generated by a Dockerfile. These are stored as images. A container is the running form of an image. An image gets brought up into a container, so it can be said they are the same, just in different states. A Docker image is the template or instructions for the container to run from [11].

Q5: (a) What does each option in the command mean? (-d, -p, -e, –name, –net)

A5: In the Docker run command seen in Figure 6, there are many options used. These similar options can be seen in Figure 7 as well. The first, -d, means to "detach", or run the container in the background, and print the container ID. This allows the container to run without holding up the terminal. -p is to select the port, or "publish" the containers port to the host. The port number on the left is the host port and the right is the container port [12]. -e is for setting environmental variables that are specific to the image, in this case, username and password. –name assigns the name of the container, and –net assigns the network for the container to be a part of (the one we created) [13].

```
checkout@ubuntu:~$ docker run -d \
> -p 27017:27017 \
> -e MONGO_INITDB_ROOT_USERNAME=mongo \
> -e MONGO_INITDB_ROOT_PASSWORD=mongo \
> --name mongodb \
> --net mongo-network \
> mongo
1ee750d3193494a617f27cd075aa22c26e5ceecb9af6e087fbe13ee2144fbfcd
```

Figure 6: Docker running Mongo

```
checkout@ubuntu:~$ docker run -d \
> -p 8081:8081 \
> -e ME_CONFIG_MONGODB_ADMINUSERNAME=mongo \
> -e ME_CONFIG_MONGODB_ADMINPASSWORD=mongo \
> -e ME_CONFIG_MONGODB_SERVER=mongodb \
> --name mongo-express \
> --net mongo-network \
> mongo-express
a0d365958f86aa397fd53e7bb988d69730b807b9ec8e7ea83b77ef9c20658df2
```

Figure 7: Docker running Mongo-express

Q6: (a) What else is seen within the container's logs?
(b) What is the MongoDB container waiting to connect with?

A6: Besides "Waiting for connections", which can be seen in Figure 8, there are also messages such as "Server Restarted" as seen in Figure 9, "child process started successfully, parent exiting" in Figure 10, and "Connection accepted" in Figure 11.
(b) The MongoDB container is waiting to connect with the Mongo-express container, as the express GUI is what gives the user access to the database.

```
{"t":{"$date":"2023-12-11T22:36:06.246+00:00"},"s":"I",  "c":"NETWORK",  "i
d":23016,   "ctx":"listener","msg":"Waiting for connections","attr":{"port"
:27017,"ssl":"off"}}
```

Figure 8: Mongo container, Waiting for connections

```
{"t":{"$date":"2023-12-11T22:36:01.140+00:00"},"s":"I",  "c":"CONTROL",  "
id":20698,   "ctx":"main","msg":"***** SERVER RESTARTED *****"}
```

Figure 9: Mongo container logs

```
child process started successfully, parent exiting
```

Figure 10: Mongo container logs

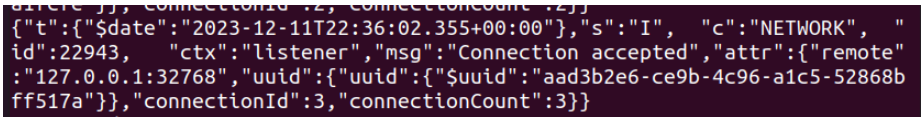


Figure 11: Mongo container logs

Q7: (a) Do the admin username and password have to be in correspondence with the root username and password set with the mongodb container? (b) Why or why not?

A7: (a) Yes, the environmental username and password set for mongoDB must be the same as the username and password set for mongo-express. (b) Having the same password works as authentication in order to connect the two containers.

Access Mongo-Express

T3: Done. See the evidence (screenshots in images/L1E1S1T3 subfolder and attachments in atts/L1E1S1T3 subfolder).

Q8: (a) What username and password need to be entered here? (b) Where can they be found?

A8: (a) Once the url in Figure 12 is entered, we are prompted to enter a username and password. Although both containers are configured with the same username and password as each other, those are not the values for here. The user is admin and the password is pass. (b) They are found within the logs of the express container, seen in Figure 13.

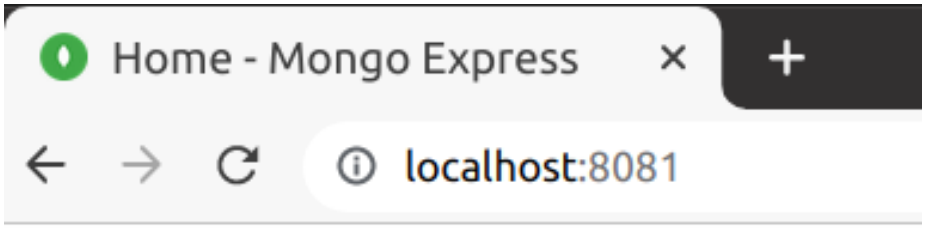


Figure 12: Access express webpage

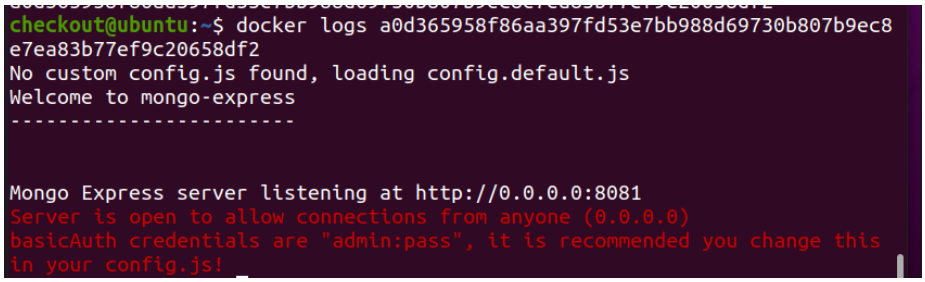


Figure 13: Mongo-express container logs

Q9: (a) What is nodejs used for?

A9: (a) Nodejs is used to run the javascript file selected to be accessed from the browser [13].

Q10: (a) What does "node server.js" do?
(b) What should you see as a response?
(c) Where is server.js configured to return that?

A10: (a) This command executes the function of nodejs and runs the javascript file.
(b) As a response, we see "app listening on port 3000!".
(c) As seen in Figure 14, in server.js there is a section that sets port 3000 to listen, and prints to the user that it is successful.

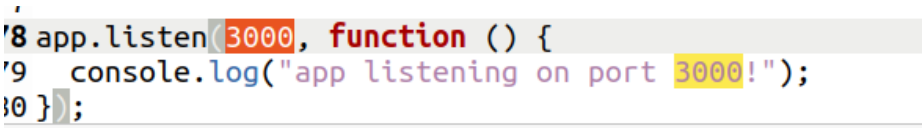


Figure 14: server.js port

Q11: (a) What is happening within the log files? Explain in detail and provide evidence. Connect logs back to the instance in the users collection.
(b) Submit logs as attachment E1S1T3-mongo-express-change-mongo-logs.txt.

A11: (a) The log can be seen in Figure 15. First the connection to the mongodb is accepted and successfully authenticated, there is a network message saying "received first command", and then the connection ends. It is not clear what change was made during this time, just that there was something done between connection and disconnection.
(b) The log file can be seen in atts/L1E1S1T3 subfolder.

Figure 15: Webpage alteration logs

3.1.2 Step 2: Docker-compose

Figure 1 represents the network we constructed to perform all or some of the tasks in Exercise 1, Step 2.

Docker-compose

T1: Done. See the evidence (screenshots in images/L1E1S2T1 subfolder).

Q1: (a) What does docker-compose do?
(b) What are the results of the docker-compose up command?
(c) What do -f and -d do?

A1: (a) Docker-compose streamlines the use of multi-container applications. It gives the ability to treat each container as one and keep them all aligned with eachother, whether on start, stop, viewing logs, etc [14]. The compose file can be seen in Figure 16.
(b) The command and its result can be seen in Figure 17, where it tells me that a new network with default driver has been created, and that both containers have been ran successfully.
(c) The -f tag is followed by the yaml file being used [15]. The -d tag is the same as during the run command, and allows the containers to be ran in the background instead of streaming the logs in the terminal.

Figure 16: Compose file

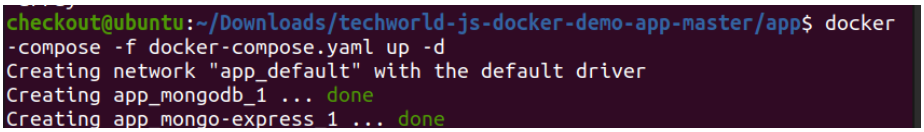


Figure 17: Bring compose file up

Q2: (a) What is the difference? (between using up or up -d)
(b) Are these logs the same/similar to what we saw in the logs of each running container prior?

A2: (a) The command output when using -d in the command can be seen in Figure 17, while some of the result of the command without using the -d option can be seen in Figure 18, and continues to stream logs in the terminal, including Figure 19 showing what we have seen before within the mongo-express container containing the username and password needed to access the database on express. This version keeps streaming the logs of any container within the file until they are brought down.
(b) Yes, as mentioned, the express logs are the same, and the mongo logs are the same before the express container produces logs as seen in

Figure 20, showing waiting for connections.

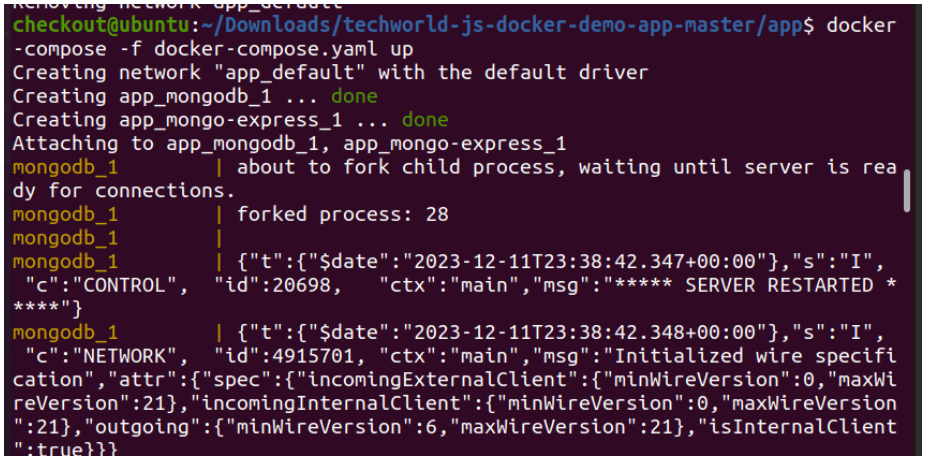


Figure 18: Compose file up, with logs

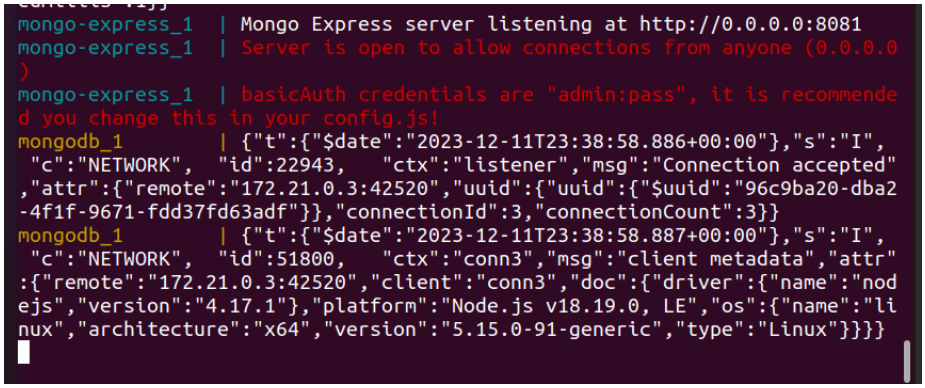


Figure 19: Evidence of express container logs

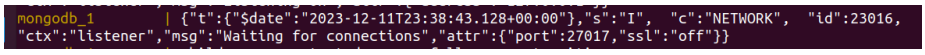


Figure 20: Evidence of properly functioning mongo container

Q3: (a) What is "data persistence"?
(b) How would we introduce data persistence in our docker environment?
(c) Aside from the database and collection being gone, what else has changed back to default?

A3: (a) Data persistence is the remainder of data created by an application, even after said application has stopped or closed [16]. This is vital because otherwise, the database would never save any work, and it would be useless.
(b) To implement data persistence in Docker, we create a volume and attach it to a directory [17]. It persists the data to the host, since docker uses the host OS. This way, even after the containers are sent down and back up, the data remains.
(c) Other than the records we can see in the users collection being gone, the changes made in the js webpage are also reverted back to original. This is because the change on the webpage is tied to the record in the database, so when the containers go down, the record is removed from the database, and therefore the change is removed from the page.

Q4: (a) What happened in the terminal that was continuously running "docker-compose -f docker-compose.yaml up" when "docker-compose -f docker-compose.yaml down" was entered?
(b) What happens if you refresh the page loaded from "localhost:8081"? Why?
(c) What happens if you refresh the page loaded from "localhost:3000"? Why?

A4: (a) Once the compose file was brought down seen in Figure 21, the terminal that was continuously running the logs of the containers quit and was exited as seen in Figure 22.
(b) If the express page was refreshed, it would not render as the connection from express to mongodb has been closed so the gui cannot be seen.
(c) For a similar reason, if this page was refreshed, it would return a blank page because even with node running, the javascript file needs to see connection to the database in order to render, so with the database down, this cannot open.

```
checkout@ubuntu:~/Downloads/techworld-js-docker-demo-app-master/app$ docker
-compose -f docker-compose.yaml down
Stopping app_mongo-express_1 ... done
Stopping app_mongodb_1       ... done
Removing app_mongo-express_1 ... done
Removing app_mongodb_1       ... done
Removing network app_default
```

Figure 21: Compose down

```
mongodb_1        | {"t":{"$date":"2023-12-11T23:58:58.331+00:00"},"s":"I",
 "c":"CONTROL",  "id":23138,   "ctx":"SignalHandler","msg":"Shutting down",
"attr":{"exitCode":0}}
app_mongodb_1 exited with code 0
checkout@ubuntu:~/Downloads/techworld-js-docker-demo-app-master/app$
```

Figure 22: Compose file logs ended

Modified Dockerfiles

T2: Done. See the evidence (screenshots in images/L1E1S2T2 subfolder).

Q5: (a) What is a Dockerfile?
(b) Why must it be named Dockerfile?

A5: (a) A Dockerfile is a text document that contains commands to create an image. These commands in the file are done for the user upon running. The file can be turned into an image and then ran in a container. A Dockerfile provides a user full terrain to create an image with anything they could want or need [18]. They can be simple or complex, and you can even use a preexisting image as your starting point in the Dockerfile. An example can be seen in Figure 23.
(b) This is the default name, although there can be some alteration to the naming convention. In some cases, the Dockerfile can be called <something>.Dockerfile, still with no extension [19].

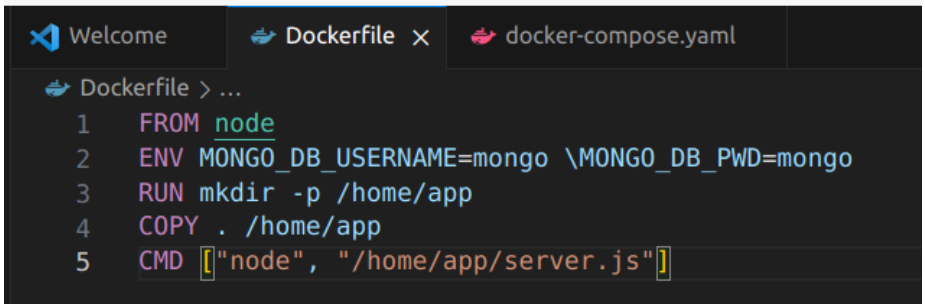


Figure 23: Dockerfile to run app

Q6: (a) Provide evidence that you know this command worked.

A6: (a) The build command can be seen in Figure 24 and the proof it worked in Figure 25. The build command takes a Dockerfile from a file to a Docker image, and the usable images can be seen from the command. In Figure 24, the lines with "1/3" or "2/3" or "3/3" contain direct proof that lines from the Dockerfile were ran exactly as is and were successful in making changes on the node image this used as a baseline.

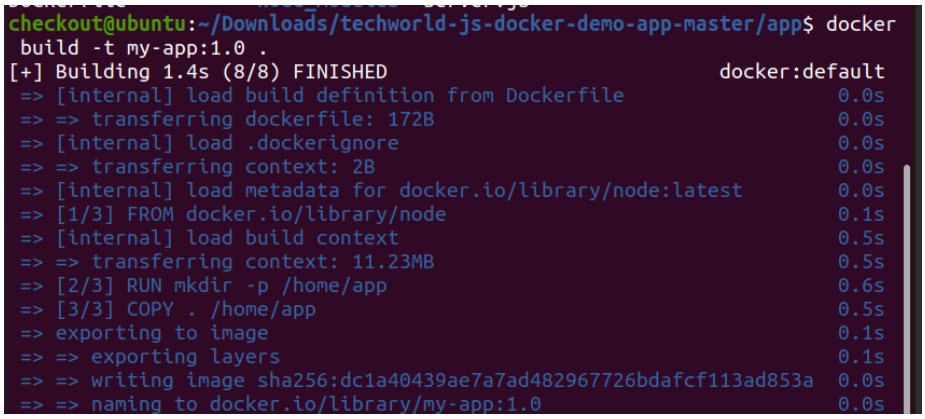


Figure 24: Build image from Dockerfile

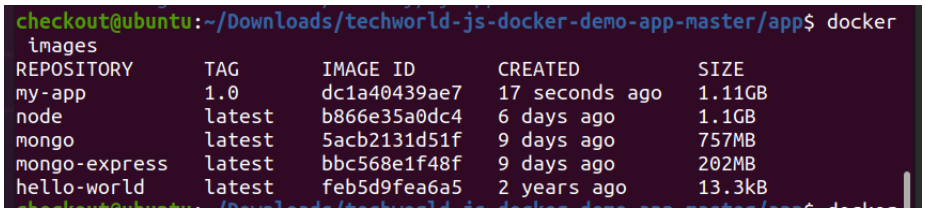


Figure 25: Check docker images

Access console of running container

T3: Done. See the evidence (screenshots in images/L1E1S2T3 subfolder).

Q7: (a) What does -it mean/do?
(b) Where does the starting point /bin/sh take you?

A7: (a) "-it" is the tag for interactive terminal, and it provides the ability to access the inside of the container for any needed reason. An example can be seen in Figure 26. The string of numbers in the command seen is the container ID of the container we are looking to access the terminal of.
(b) The directory upon accessing the terminal is "/" which can be seen in Figure 27. From there you can see variables such as in Figure 28, you can see the contents of a directrory as in Figure 29, and change directory just as you normally would as in Figure 30. This helps provide a deeper understanding to the containers by seeing the files placed were they were sent from the Dockerfile.

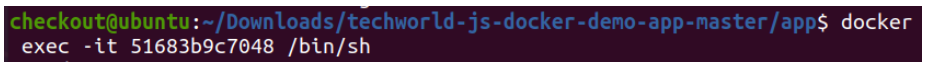


Figure 26: Enter interactive terminal of container

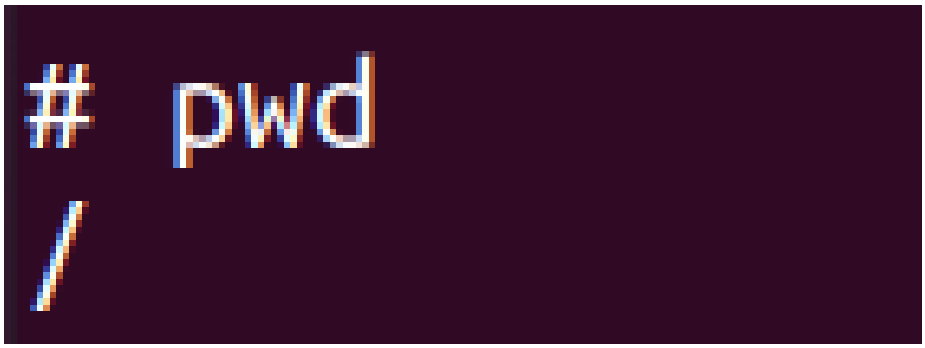


Figure 27: Directory entered

```
# env
NODE_VERSION=21.4.0
HOSTNAME=51683b9c7048
YARN_VERSION=1.22.19
HOME=/root
MONGO_DB_USERNAME=mongo
TERM=xterm
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
MONGO_DB_PWD=mongo
PWD=/
```

Figure 28: Environmental variables

```
# ls /home/app
Dockerfile          images        package-lock.json
change.txt          index.html    package.json
docker-compose.yaml node_modules  server.js
```

Figure 29: /home/app Directory

```
# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
# cd home
# ls
app  node
# cd app
# ls
Dockerfile          images        package-lock.json
change.txt          index.html    package.json
docker-compose.yaml node_modules  server.js
```

Figure 30: Access directory

Q8: (a) How did those files end up there inside the container?

A8: (a) As mentioned, in the Dockerfile, there is a line "COPY ./app /home/app" seen in Figure 31 which copied the contents of the app folder on the host into the newly created app folder in the container. The contents of the folder on the host can be seen in Figure 32, and the folder in the container can be seen in Figure 33.

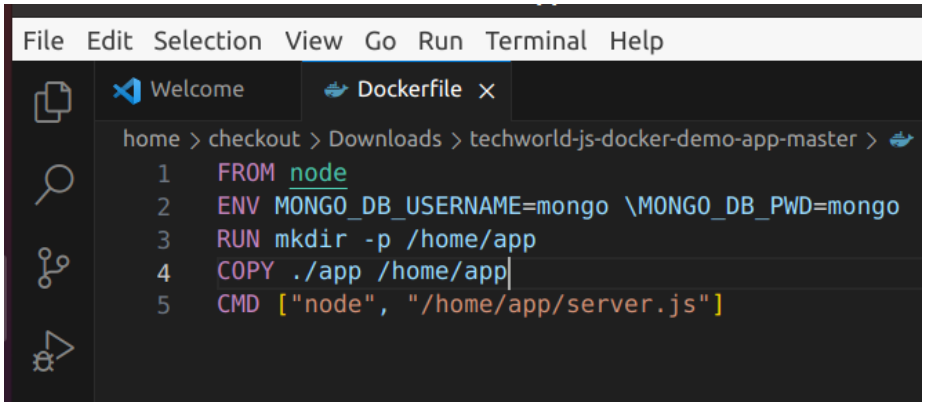


Figure 31: Edited Dockerfile

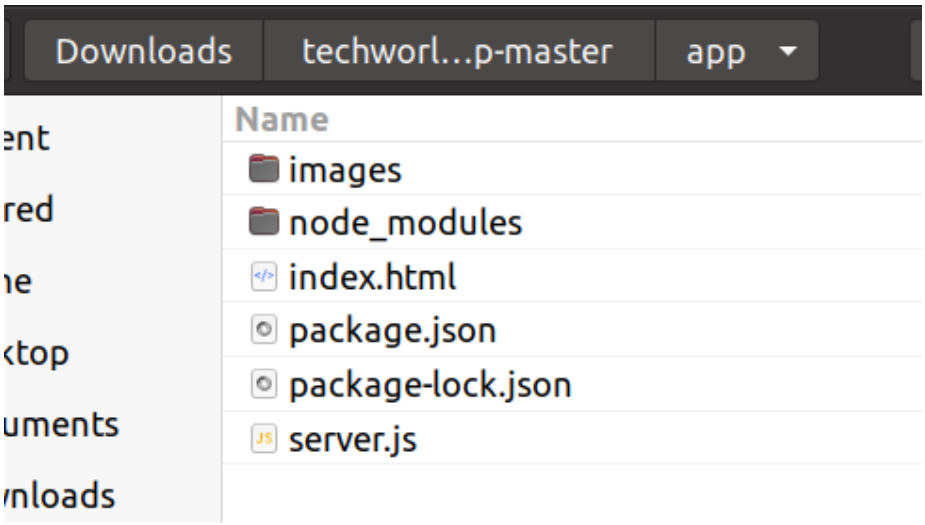


Figure 32: Modified app folder

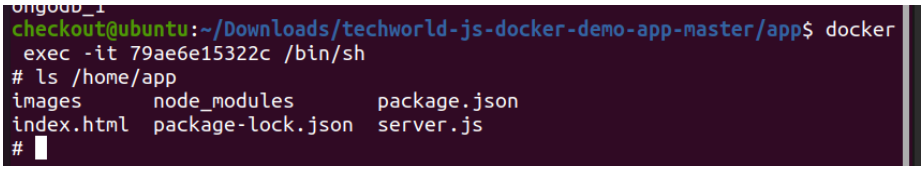


Figure 33: Check edited folder in interactive terminal

Q9: (a) What is port 3000 used for?
(b) Where is it set to use 3000?

A9: (a) Port 3000 is not a required port number for this to have worked, but 3000 is a typical one to use for accessing a containerized application in Docker [20].
(b) It is set to be used in server.js as seen in Figure 14. Although "node server.js" is not running at this point in a terminal, since the container running that we created from our Dockerfile, the command is executed at runtime because of the CMD line in the Dockerfile that this container is running from as seen in Figure 31. Since the programmed response from running "node server.js" is that the app is listening on port 3000, this is what is returned when running the image to a container.

Q10: (a) How would you edit the docker-compose file for the my-app image to be started at the same time as mongo and mongo-express?
(b) Devise a method and test it. Show evidence.

A10: (a) We edited the compose file to add my-app as a service as can be seen in Figure 34. Originally, we set both ports to be 3000, but since the host is already listening on that port to run the javascript file, we reveiced the error that can be seen in Figure 35. What is interesting about docker-compose files it that although there was an error, the other containers were unscathed and went up with no problem. Once the host port was changed to what can be seen in the file, we ran the same command again and received "done" in green.
(b) All the containers were able to go up at once as seen in Figure 36 they are all running. Figure 37 shows both browsers able to be loaded.

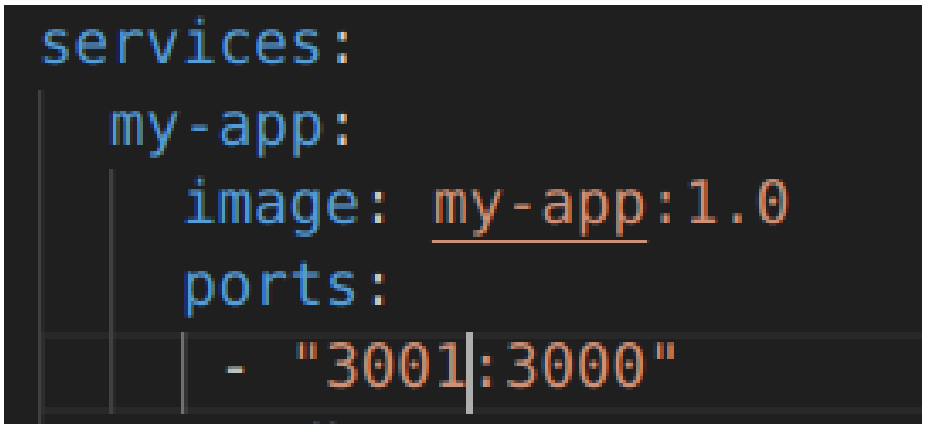


Figure 34: Adding my-app image to

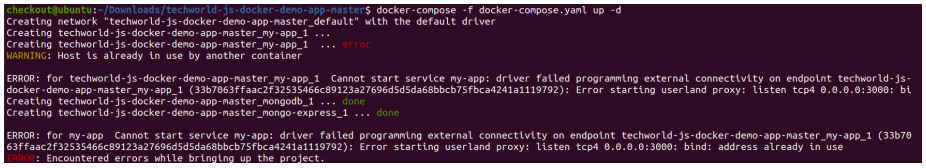


Figure 35: Running new compose file with an error

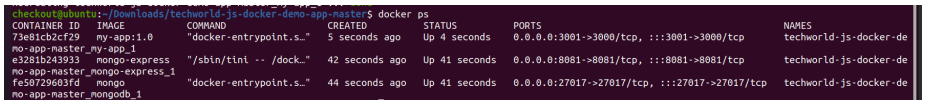


Figure 36: Checking running containers

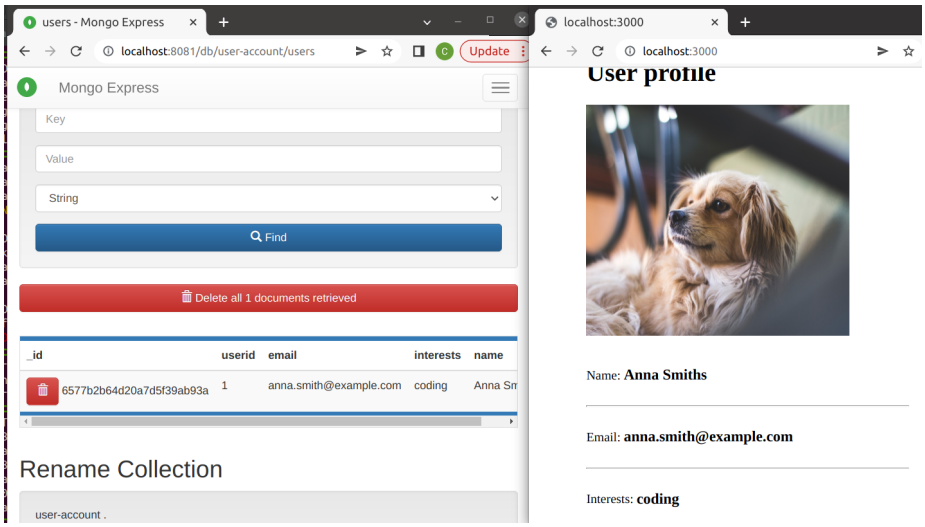


Figure 37: Working webpage connections

Mongo and Mongo-express using docker-compose

T4: Done. See the evidence (screenshots in images/L1E1S2T4 subfolder).

Q11: (a) How is npm used in this case?
(b) How does it know to run the server.js file?
(c) How is it different from nodejs?

A11: (a) npm is used in place of node to run the javascript file.
(b) npm is able to run server.js because of the package.json file seen in the app directory. Npm will run the "scripts" attribute seen with the json file in Figure 38 [21]. It goes immediately to the start line and runs the package.
(c) It is different because node runs javascript, while npm manages packages including applications and their dependencies [22].

package.json change.txt

```
{
  "name": "docker",
  "version": "1.0.0",
  "description": "docker containers",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "Nana Janashia",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.1",
    "express": "^4.18.2",
    "mongodb": "^3.3.3"
  },
  "keywords": [
    "docker"
  ],
  "devDependencies": {}
}
```

Figure 38: Package ran with npm

Q12: (a) What is the result when you bring the container back up?
(b) Why?
(c) What are docker volumes?
(d) Where can you find them within the OS?

A12: (a) The containers going down then back up can be seen in Figure 39. Once brought back up, before, the records would be gone and the changes on the webpage would be gone, but this time they were still there. The change remaining after the containers restart in the database can be seen in Figure 40, and the change remaining on the webpage can be seen in Figure 41.
(b) This is because of the addition of volumes into the docker-compose file which can be seen in Figure 42.
(c) Volumes are a filesystem introducing data persistence to the docker environment, and the files are stored on the host. When a container with volumes is started, "Docker loads the read-only image layer, adds a read-write layer on top of the image stack, and mounts volumes onto the container filesystem. [23]"
(d) The data records can be found on the host at /var/liv/docker/volumes as seen in Figure 43 and Figure 44.

```
checkout@ubuntu:~/Downloads/techworld-js-docker-demo-app-master$ docker-com
pose -f docker-compose.yaml down
Stopping techworld-js-docker-demo-app-master_mongo-express_1 ... done
Stopping techworld-js-docker-demo-app-master_mongodb_1       ... done
Removing techworld-js-docker-demo-app-master_mongo-express_1 ... done
Removing techworld-js-docker-demo-app-master_mongodb_1       ... done
Removing network techworld-js-docker-demo-app-master_default
checkout@ubuntu:~/Downloads/techworld-js-docker-demo-app-master$ docker-com
pose -f docker-compose.yaml up -d
Creating network "techworld-js-docker-demo-app-master_default" with the def
ault driver
Creating techworld-js-docker-demo-app-master_mongodb_1 ... done
Creating techworld-js-docker-demo-app-master_mongo-express_1 ... done
```

Figure 39: Containers down

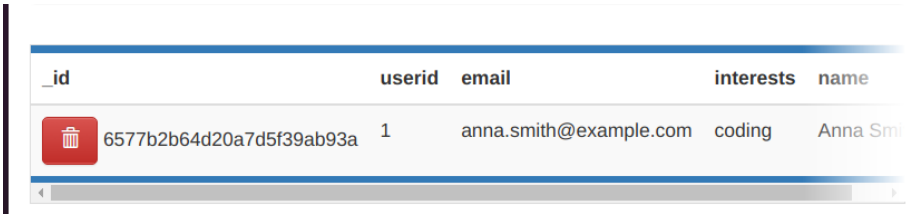


Figure 40: Edit webpage

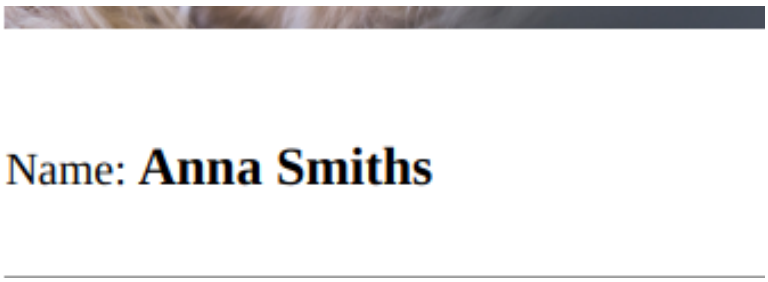


Figure 41: Webpage edit

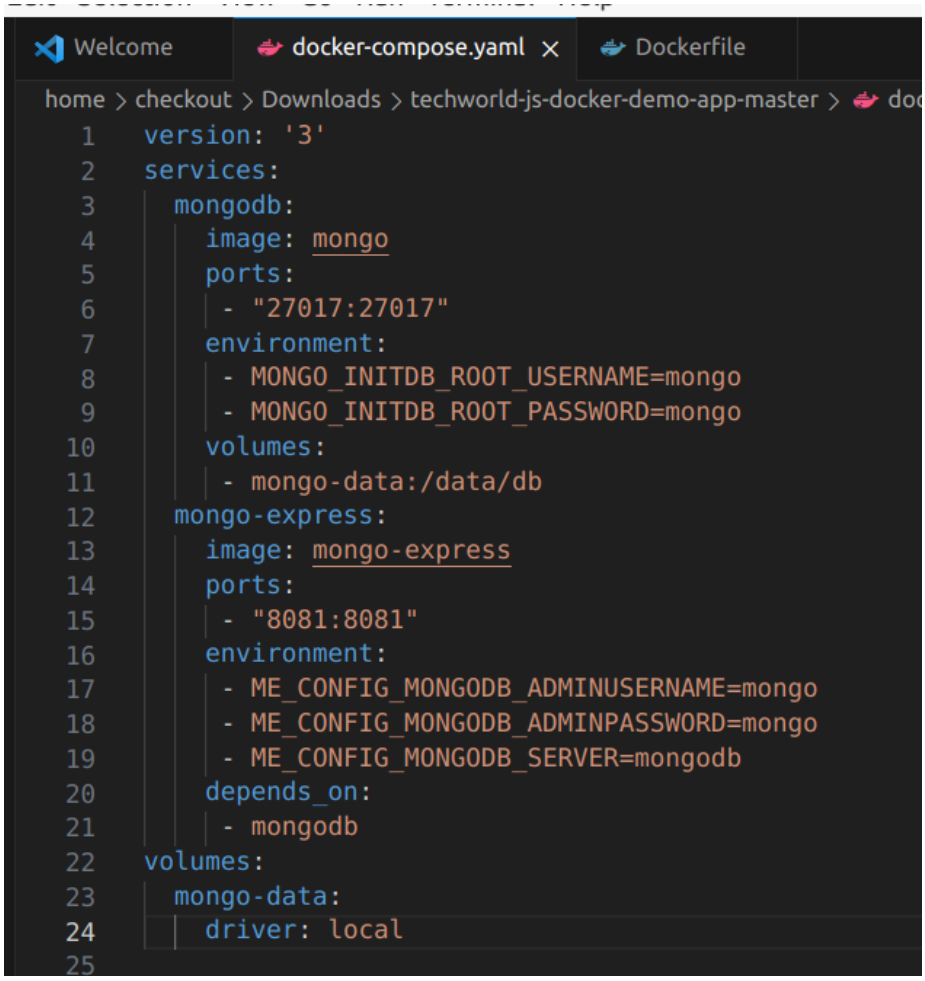


Figure 42: Adding volumes to compose

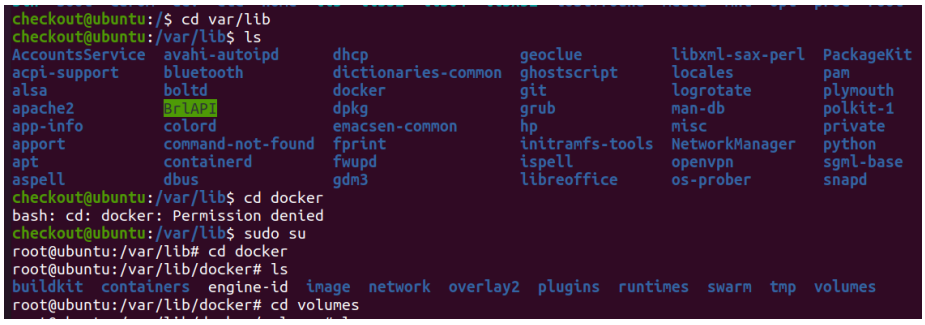


Figure 43: Checking location of volumes

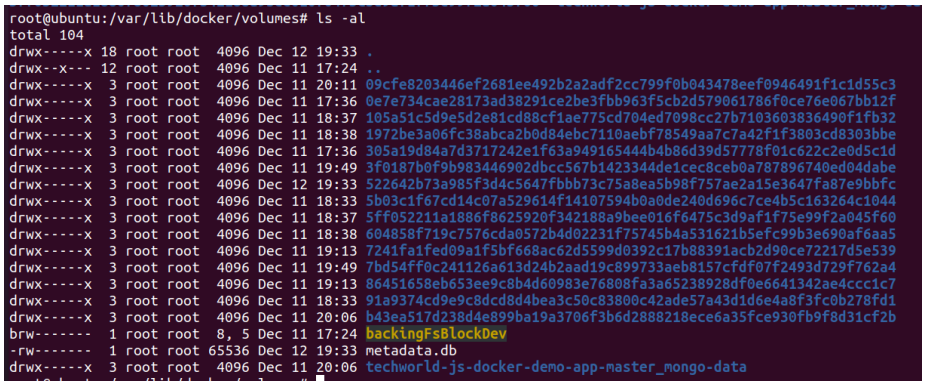


Figure 44: Listed volumes saved

4 KLT & EVC

4.1 Key Learning & Takeaways (KLT)

Overall, this lab teaches the main components of using Docker with a lot of repetition to get us used to the commands and altering the containers. We appreciate the value of Docker, especially knowing how it can be used within OpenStack. Besides it's use in OpenStack, it is always valuable to learn new tools and services that can assist projects in the future.

4.2 Expectancy-Value-Cost (EVC)

Expectancy
Yes, I felt I could do the steps and tasks in the lab. This lab did not take long for us as we are used to two exercise labs. The tasks were clear and explained enough to be able to do.
Value
We feel value in this lab because of the value of Docker itself and the value of containers within OpenStack. It is very helpful to understand Docker before adding containers in OpenStack so you better know what you are dealing with.
Cost
There was a time constraint while doing this lab, but not enough that we were prevented from producing a proper report for this lab.

5 Lab Observations, Suggestions & Best Practices

5.1 Observations

All edits observed were added or changed as we went through the lab.

5.2 Suggestions

Suggestions for this lab mainly include adding the routing aspect of the lab as a second exercise or another step, but time has not allowed that at this point.

5.3 Best Practices

The best practice during this lab was being able to edit and change the instructions where we saw fit throughout execution.

6 Lab References

[1] J. Jameson. "Containerize openstack with docker." Retrieved on 2023-12-13. (2015), [Online]. Available: https://www.redhat.com/en/blog/containerize-openstack-docker.

[2] docker hub. "Mongo." Retrieved on 2023-12-12. (2023), [Online]. Available: https://hub.docker.com/_/mongo.

[3] S. Agarwal. "Setting up mongo express 101: A comprehensive guide." Retrieved on 2023-12-12. (2023), [Online]. Available: https://hevodata.com/learn/mongo-express/#:~:text=Mongo%20Express%20is%20an%20interactive,databases%2C%20collections%2C%20and%20documents..

[4] docker hub. "Docker pull." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.docker.com/engine/reference/commandline/pull/.

[5] Github. "Working with the docker registry." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.github.com/en/enterprise-server@3.7/packages/working-with-a-github-packages-registry/working-with-the-docker-registry.

[6] docker hub. "Hello-world." Retrieved on 2023-12-12. (2023), [Online]. Available: https://hub.docker.com/_/hello-world.

[7] docker docs. "Bridge network driver." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.docker.com/network/drivers/bridge/.

[8] docker docs. "Host network driver." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.docker.com/network/drivers/host/.

[9] arun-gupta. "None network with null driver." Retrieved on 2023-12-12. (2015), [Online]. Available: https://github.com/docker/machine/issues/2221.

[10] M. Church. "Understanding docker networking drivers and their use cases." Retrieved on 2023-12-12. (2016), [Online]. Available: https://www.docker.com/blog/understanding-docker-networking-drivers-use-cases/.

[11] aws. "Key differences: Docker images vs. docker containers." Retrieved on 2023-12-12. (2023), [Online]. Available: https://aws.amazon.com/compare/the-difference-between-docker-images-and-containers/#:~:text=A%20Docker%20container%20is%20a%20self%2Dcontained%2C%20runnable%20software%20application,containers%20over%20an%20application's%20lifecycle..

[12] M. Y. "How to change port of an docker image?" Retrieved on 2023-12-12. (2022), [Online]. Available: https://forums.docker.com/t/how-to-change-port-of-an-docker-image/129706.

[13] R. Sheldon. "Node.js (node)." Retrieved on 2023-12-12. (2022), [Online]. Available: https://www.techtarget.com/whatis/definition/Nodejs.

[14] docker docs. "Docker compose overview." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.docker.com/compose/#:~:text=Compose%20is%20a%20tool%20for,to%20configure%20your%20application's%20services..

[15] docker docs. "Overview of docker compose cli." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.docker.com/compose/reference/.

[16] MongoDB. "An introduction to data persistence." Retrieved on 2023-12-12. (2023), [Online]. Available: https://www.mongodb.com/databases/data-persistence.

[17] docker docs. "Persist the todo data." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.docker.com/get-started/05_persisting_data/#:~:text=By%20creating%20a%20volume%20and,the%20host%20in%20the%20volume..

[18] docker docs. "Dockerfile reference." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.docker.com/engine/reference/builder/#:~:text=A%20Dockerfile%20is%20a%20text,can%20use%20in%20a%20Dockerfile%20..

[19] docker docs. "Filename." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.docker.com/build/building/packaging/#:~:text=Filename,distinct%20Dockerfiles%20for%20specific%20purposes..

[20] M. Abbas. "Docker port mapping explained." Retrieved on 2023-12-12. (2023), [Online]. Available: https://medium.com/@AbbasPlusPlus/docker-port-mapping-explained-c453dfb0ae39#:~:text=Accessing%20the%20Containerized%20Application,%2F%2F%5BHOST_IP%5D%3A3000%20..

[21] npm Docs. "Npm-start." Retrieved on 2023-12-12. (2021), [Online]. Available: https://docs.npmjs.com/cli/v7/commands/npm-start.

[22] hostadvice. "Importance of understanding the differences between node.js and npm." Retrieved on 2023-12-12. (2023), [Online]. Available: https://hostadvice.com/blog/web-hosting/node-js/node-js-vs-npm/#:~:text=They%20Are%20Two%20Different%20Things&text=serve%20different%20purposes.-,Node.,js%20applications..

[23] S. Zivuku. "Understanding docker volumes." Retrieved on 2023-12-12. (2023), [Online]. Available: https://earthly.dev/blog/docker-volumes/#:~:text=A%20Docker%20volume%20is%20an,keeping%20them%20secure%20and%20effective..

7 Acknowledgments

We would like to acknowledge Professor Salib for helping during the creation of the instructions and the continued editing of them. We'd also like to acknowledge him for the help to learn and understand these topics last semester in order to put this lab together.

Lab 2 Report

James Madison University
Information Technology (IT) Program
Capstone Project Labs
Fall 2023

Lab 2 - OpenStack Deployment

Group 1

Thursday 14th December, 2023

17:37:08

Author(s):
Casey Alexander
Katherine Botticelli

Submitted to:
Dr. Emil Salib

Honor Pledge: I have neither given nor received help on his lab that violates the spirit of the JMU Honor Code.

Casey Alexander	Katherine Botticelli
Signature	*Signature*
Date	*Date*

Execution: Here is my percentage of executing the the hands-on portion of the lab. It is expected that each partner execute 100%.

C. Alexander (100%)	K. Botticelli (100%)
Signature	*Signature*
Date	*Date*

Contributions: Here is my Lab Report Contribution Percentages for this lab. It is expected that the sum of the contributions of the group members is 100%.

C. Alexander (50%)	K. Botticelli (50%)
Signature	*Signature*
Date	*Date*

Rubric Check: I have read and complied with every item listed in this lab's rubric as posted on Canvas.

C. Alexander (Yes)	K. Botticelli (Yes)
Signature	*Signature*
Date	*Date*

Contents

List of Figures

List of Tables

List of Attachments & Screenshots

Attachments

Screenshots

.

1 /title-seal.jpg
2 /L2E1S1N1.png

L2E1S1T1

3 /L2E1S1T1-3.png
4 /L2E1S1T1-11.png
5 /L2E1S1T1-5.png
6 /L2E1S1T1-7.png
7 /L2E1S1T1-15.png
8 /L2E1S1T1-8.png
9 /L2E1S1T1-12.png
10 /L2E1S1T1-13.png
11 /L2E1S1T1-4.png
12 /L2E1S1T1-2.png
13 /L2E1S1T1-10.png
14 /L2E1S1T1-14.png
15 /L2E1S1T1-1.png
16 /L2E1S1T1-16.png
17 /L2E1S1T1-9.png
18 /L2E1S1T1-6.png

L2E1S1T2

19 /L2E1S1T2-10.png
20 /L2E1S1T2-8.png
21 /L2E1S1T2-3.png
22 /L2E1S1T2-5.png
23 /L2E1S1T2-9.png
24 /L2E1S1T2-11.png
25 /L2E1S1T2-2.png
26 /L2E1S1T2-6.png
27 /L2E1S1T2-7.png

/L2E1S1T2-1.png
/L2E1S1T2-4.png

L2E1S1T3

/L2E1S1T3-12.png
/L2E1S1T3-1.png
/L2E1S1T3-18.png
/L2E1S1T3-14.png
/L2E1S1T3-19.png
/L2E1S1T3-7.png
/L2E1S1T3-2.png
/L2E1S1T3-17.png
/L2E1S1T3-11.png
/L2E1S1T3-4.png
/L2E1S1T3-3.png
/L2E1S1T3-13.png
/L2E1S1T3-6.png
/L2E1S1T3-5.png
/L2E1S1T3-10.png
/L2E1S1T3-16.png
/L2E1S1T3-9.png
/L2E1S1T3-8.png
/L2E1S1T3-15.png

L2E1S1T4

/L2E1S1T4-8.png
/L2E1S1T4-1.png
/L2E1S1T4-3.png
/L2E1S1T4-11.png
/L2E1S1T4-9.png
/L2E1S1T4-10.png
/L2E1S1T4-4.png
/L2E1S1T4-5.png
/L2E1S1T4-2.png
/L2E1S1T4-12.png
/L2E1S1T4-6.png
/L2E1S1T4-7.png

L2E1S2T1

/L2E1S2T1-6.png
/L2E1S2T1-3.png
/L2E1S2T1-7.png
/L2E1S2T1-8.png
/L2E1S2T1-2.png
/L2E1S2T1-5.png
/L2E1S2T1-4.png
/L2E1S2T1-10.png
/L2E1S2T1-.png
/L2E1S2T1-9.png

L2E1S2T2

/L2E1S2T2-6.png
/L2E1S2T2-5.png
/L2E1S2T2-13.png

```
74          /L2E1S2T2-4.png
75          /L2E1S2T2-3.png
76          /L2E1S2T2-8.png
77          /L2E1S2T2-7.png
78          /L2E1S2T2-9.png
79          /L2E1S2T2-11.png
80          /L2E1S2T2-2.png
81          /L2E1S2T2-1.png
82          /L2E1S2T2-12.png
83          /L2E1S2T2-10.png
    L2E1S2T3
84          /L2E1S2T3-14.png
85          /L2E1S2T3-40.png
86          /L2E1S2T3-2.png
87          /L2E1S2T3-35.png
88          /L2E1S2T3-33.png
89          /L2E1S2T3-37.png
90          /L2E1S2T3-15.png
91          /L2E1S2T3-11.png
92          /L2E1S2T3-24.png
93          /L2E1S2T3-16.png
94          /L2E1S2T3-8.png
95          /L2E1S2T3-17.png
96          /L2E1S2T3-10.png
97          /L2E1S2T3-13.png
98          /L2E1S2T3-7.png
99          /L2E1S2T3-31.png
100         /L2E1S2T3-36.png
101         /L2E1S2T3-32.png
102         /L2E1S2T3-29.png
103         /L2E1S2T3-18.png
104         /L2E1S2T3-6.png
105         /L2E1S2T3-38.png
106         /L2E1S2T3-22.png
107         /L2E1S2T3-28.png
108         /L2E1S2T3-23.png
109         /L2E1S2T3-12.png
110         /L2E1S2T3-20.png
111         /L2E1S2T3-30.png
112         /L2E1S2T3-27.png
113         /L2E1S2T3-21.png
114         /L2E1S2T3-26.png
115         /L2E1S2T3-1.png
116         /L2E1S2T3-5.png
117         /L2E1S2T3-34.png
118         /L2E1S2T3-3.png
119         /L2E1S2T3-19.png
120         /L2E1S2T3-25.png
121         /L2E1S2T3-4.png
122         /L2E1S2T3-9.png
```

123 /L2E1S2T3-39.png

1 Introduction

Openstack is a cloud computing service that can be either public cor Private. The value gained in this lab is learning how to deploy a cloud server using kolla-ansible in deployment which is predefined containers that help ease deployment and make customising in openstack easy [1]. Openstack has the ability for creating instants which are essentially VM stored in the cloud. This lab focuses on the importance of creating and deploying our openstack instance as well as creating networks, subnets, routers, instances and images in Openstack through an administrator perspective.

In this lab we learned about Openstack and how to deploy and configure of deployment using globals.yml. We learned how to create the password.yml file for openstack deployment. This lab also heavily focused on how to create instances, subnets, networks and images in openstack. We also learned about the connects in openstack between the private and public networks.

2 Network Diagrams

2.1 Exercise 1 - OpenStack Deployment

2.1.1 Step 1: Configuration

Figure 1 represents the network arrangement constructed for Exercise 1, Step 1 tasks.

2.1.2 Step 2: Beginning customization

Figure 1 represents the network arrangement constructed for Exercise 1, Step 2 tasks.

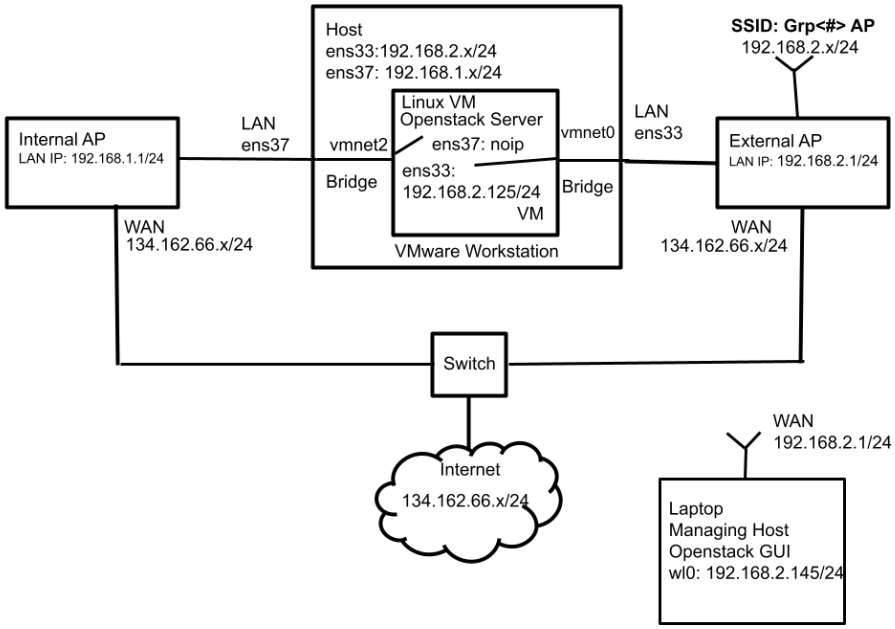


Figure 1: Network Diagram

3 Lab Exercises: Results & Analysis

3.1 Exercise 1 - OpenStack Deployment

3.1.1 Step 1: Configuration

Figure 1 represents the network we constructed to perform all or some of the tasks in Exercise 1, Step 1.

Setup

T1: Done. See the evidence (screenshots in images/L2E1S1T1 subfolder).

Q1: (a) What ip address is assigned to ens33 and ens37?
(b) Why are these addresses assigned to the interfaces?

A1: (a)The ip address assigned to ens33 is 192.168.2.100 and there is no ip address assigned to ens38 as seen in Figure 4.
(b) 192.168.2.100 is assigned to ens33 because we statically set the ip address for the eui-48 address of the ens33 interface as seen in Figure 3. There is no address assigned to ens37 because as seen in Figure 2 we set this interface to have no ip address assigned to it.

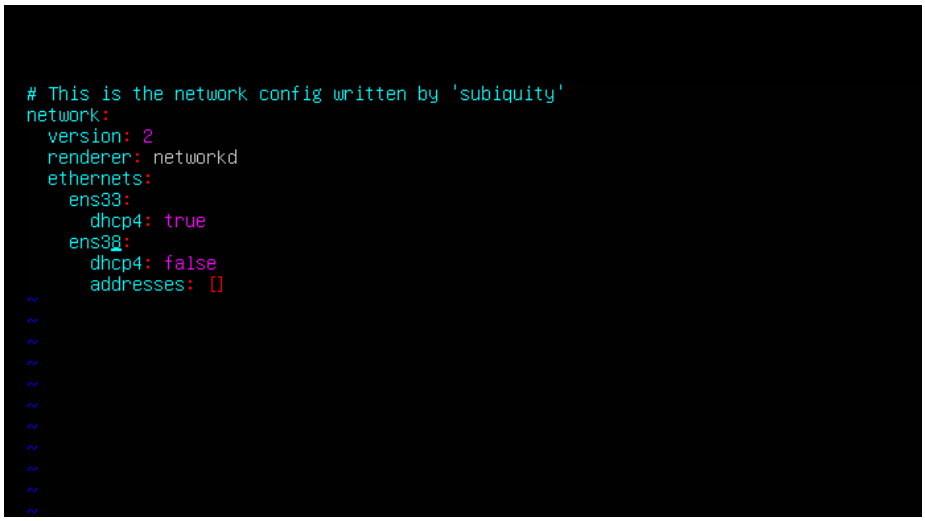


Figure 2: Netplan File

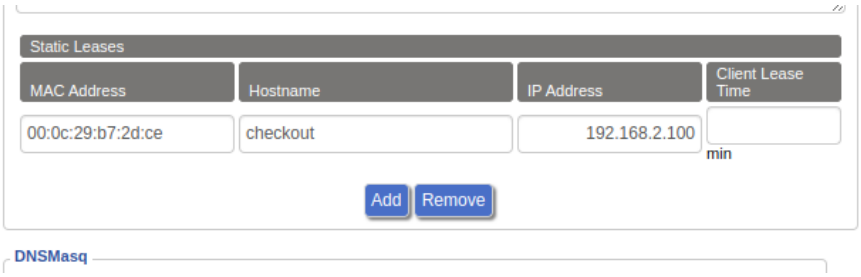


Figure 3: 192.168.2.1 Router Static Ip settings

Figure 4: ip in Openstack VM

Installations

T2: Done. See the evidence (screenshots in images/L2E1S1T2 subfolder).

Q2: (a) What does the command above do?
(b) What is wallaby?
(c) Provide evidence that the command above worked.

A2: (a) The command "source wallaby/bin/activate" runs the python code for wallaby and puts the user into a wallaby terminal as seen in Figure 5 where the terminal now has a (wallaby) before the normal terminal prompt.
(b) Wallaby is an open source cloud infrastructure software. It is the version of openstack we are running. Openstack wallaby comes with many services such as nova and keystone , it also always has the ability for running kolla deployment [2].
(c) The command works as seen in Figure 5 where the terminal now has (wallaby) in before the normal prompt meaning wallaby is running.

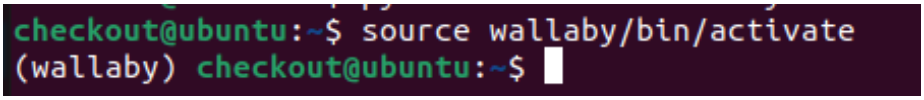


Figure 5: command and terminal proof of wallaby running

Q3: (a) What is ansible?

A3: Ansible is automation configuration that helps easy openstack deployment [3]. It is also always for configurations of a deployed openstack instance to be simple and not make a fully redeployment happen after every change [4].

Begin openstack install + create kolla directory

T3: Done. See the evidence (screenshots in images/L2E1S1T3 subfolder).

Q4: (a) Why do we need to change the permissions of this file?

A4: Originally the kolla directory is under root permissions as seen in Figure 6 this means that every time we would like to go into this directory we will need to be in root permission to access it which would require us to be sudo su for every time we go top this file. Changing permission as seen in Figure 7 to checkout as seen in Figure 8 is so we don't need to be sudo su to be in this directory every time we need access to it.

```
(wallaby) checkout@ubuntu:~$ sudo mkdir -p /etc/kolla
(wallaby) checkout@ubuntu:~$ cd /etc
(wallaby) checkout@ubuntu:/etc$ ls -al | grep kolla
drwxr-xr-x   2 root root       4096 Dec 12 22:52 kolla
```

Figure 6: Root permissions of kolla directory

```
(wallaby) checkout@ubuntu:/etc$ sudo chown $USER:$USER kolla
```

Figure 7: Command "sudo chown $USER:$USER kolla"

```
(wallaby) checkout@ubuntu:/etc$ ls -al | grep kolla
drwxr-xr-x   2 checkout checkout   4096 Dec 12 22:52 kolla
(wallaby) checkout@ubuntu:/etc$
```

Figure 8: Checkout permissions of kolla directory

Q5: (a)What did we enable in globals.yml?
(b)Why did we enable these services?

A5: (a) We enabled etcd as seen in Figure 9. We also enabled kutyr as seen in Figure 10
(b) the service etcd is used to keep track of what services are working and distributes keys in Openstack [5]. The kutyr services is a neutron services that works with networking in side of Docker for us it helps with container networking [6].

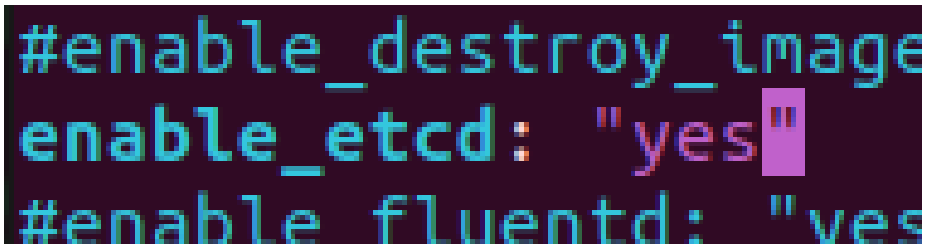


Figure 9: Enabled etcd service in Globals.yml

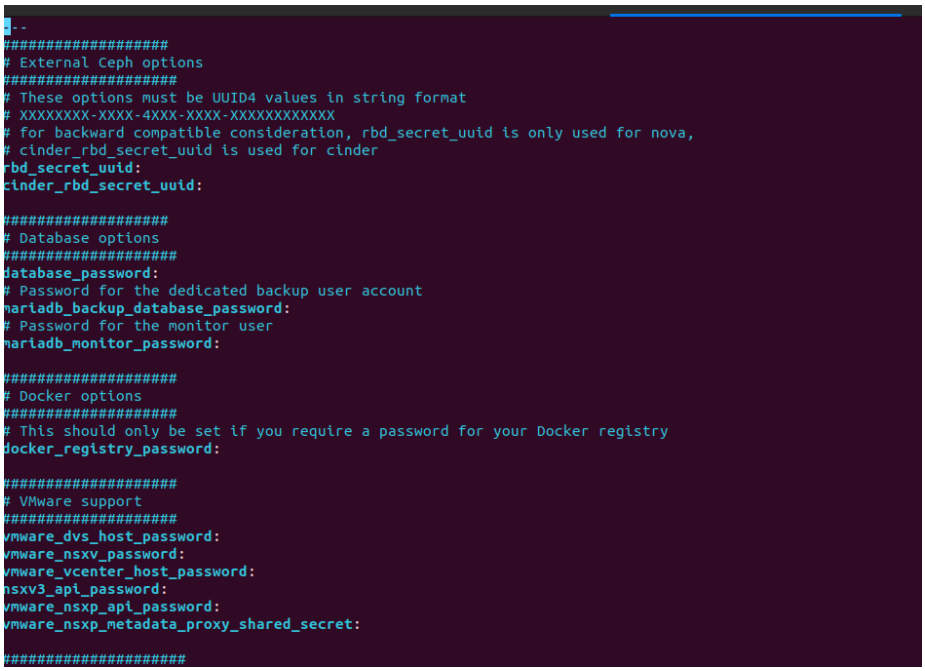
Figure 10: Enabled Kutyr service in Globals.yml

More installation + populate passwords.yml file

T4: Done. See the evidence (screenshots in images/L2E1S1T4 subfolder).

Q6: (a) Why did the passwords.yml file change after the 2 commands we executed?
(b) What are the contents of passwords for?

A6: (a) The command koll-genpwd is for generating new passwords [1] for the password.yml file in the kolla directory as seen in Figure 11 the password.yml is empty with no password in it after we run the command kolla-genpwd the password.yml file is populated with password as seen in Figure 12.
(b) The password.yml file is for ansible deployment to store passwords for the containers needed to deploy openstack in kolla ansible.



```
---
####################
# External Ceph options
####################
# These options must be UUID4 values in string format
# XXXXXXXX-XXXX-4XXX-XXXX-XXXXXXXXXXXX
# for backward compatible consideration, rbd_secret_uuid is only used for nova,
# cinder_rbd_secret_uuid is used for cinder
rbd_secret_uuid:
cinder_rbd_secret_uuid:

####################
# Database options
####################
database_password:
# Password for the dedicated backup user account
mariadb_backup_database_password:
# Password for the monitor user
mariadb_monitor_password:

####################
# Docker options
####################
# This should only be set if you require a password for your Docker registry
docker_registry_password:

####################
# VMware support
####################
vmware_dvs_host_password:
vmware_nsxv_password:
vmware_vcenter_host_password:
nsxv3_api_password:
vmware_nsxp_api_password:
vmware_nsxp_metadata_proxy_shared_secret:

####################
```

Figure 11: password.yml with no passwords

Figure 12: password.yml with passwords

Q7: Open a web browser in Firefox and enter the url "192.168.2.x".
(a) What does this url bring up? Provide evidence.
(b) Why does this web page appear the way it does?
(c) Provide another way to know your deployment is working.

A7: (a) The url www.192.168.2.100 brings up an openstack login page as seen in Figure 13.
(b) Openstacks webpage appears the way it does because we set up our stack ip address 192.168.2.100 is our globals.yml that we use to deploy openstack to allow this ip address to be what our openstack instance runs of off so when we are bringing up the openstack login page we use that ip address to access it..
(c) We can tell that our deployment is working because when we do an ip a instead of just the 3 interfaces we got before we get a lot of interfaces representing all of the openstack services running on the VM and showing us that openstack is in fact deployed.

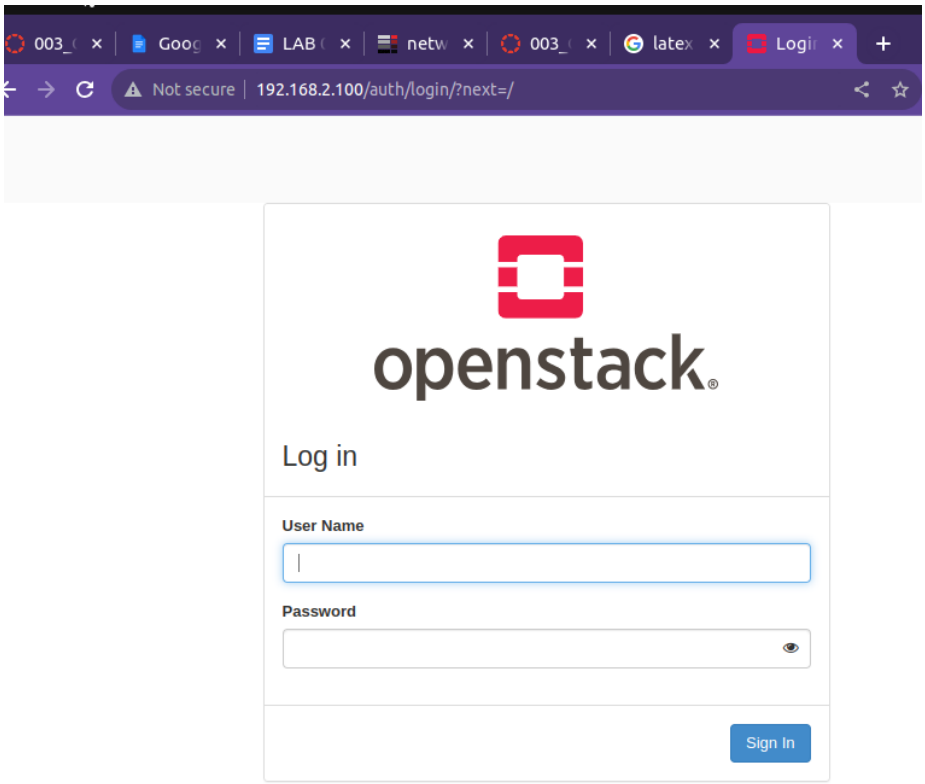


Figure 13: Openstack web Server Login page

3.1.2 Step 2: Beginning customization

Figure 1 represents the network we constructed to perform all or some of the tasks in Exercise 1, Step 2.

Post deploy actions

T1: Done. See the evidence (screenshots in images/L2E1S2T1 subfolder).

Q1: (a) What does a Kolla-ansible post-deploy command do?

A1: The kolla-ansible post deploy command is used to get the admin openrc file as seen in Figure 14. This file contains the password, username and other important information about the admin user of our newly deployed openstack instance as seen in Figure 15.

```
(wallaby) checkout@ubuntu:/etc/kolla$ ls
admin-openrc.sh  heat-engine          neutron-l3-agent           nova-novncproxy
all-in-one       horizon              neutron-metadata-agent     nova-scheduler
clouds.yaml      keystone             neutron-openvswitch-agent  nova-ssh
cron             keystone-fernet      neutron-server             openvswitch-db-server
etcd             keystone-ssh         nova-api                   openvswitch-vswitchd
fluentd          kolla-toolbox        nova-api-bootstrap         passwords.yml
glance-api       kuryr                nova-cell-bootstrap        placement-api
globals.yml      mariadb              nova-compute               rabbitmq
heat-api         memcached            nova-conductor
heat-api-cfn     neutron-dhcp-agent   nova-libvirt
```

Figure 14: kolla directory after post-deploy command

```
# Clear any old environment that may conflict.
for key in $( set | awk '{FS="="}  /^OS_/ {print $1}' ); do unset $key ; done
export OS_PROJECT_DOMAIN_NAME='Default'
export OS_USER_DOMAIN_NAME='Default'
export OS_PROJECT_NAME='admin'
export OS_TENANT_NAME='admin'
export OS_USERNAME='admin'
export OS_PASSWORD='TUj63K0TUMtR9UCKfHgu3CGTNoI80HOwc9dlb2vz'
export OS_AUTH_URL='http://192.168.2.100:5000'
export OS_INTERFACE='internal'
export OS_ENDPOINT_TYPE='internalURL'
export OS_IDENTITY_API_VERSION='3'
export OS_REGION_NAME='RegionOne'
export OS_AUTH_PLUGIN='password'
~
~
~
```

Figure 15: admin-openrc.sh file

Q2: Open a private browser (firefox), enter 192.168.2.x, your deployment static IP.
(a) What does this url bring up?
(b) How do you login to the openstack page?
(c) Provide evidence that your login attempt worked.

A2: (a) The url 192.168.2.100 brings up the login page with a username and password input for openstack as seen in Figure 16.
(b) To login to Openstack we use the admin-openrc file as seen in Figure 15 this file contains the username and password for our openstack instance. We use the username admin and the password "TUj63K0TUMtR9UC KfHgu3CGTNol80HOwc9lb2vz" and enter these values into their respective inputs on the browser page as seen in Figure 17 we then hit enter this will bring us into the openstack instance management web browser.
(c) In Figure 18 we can see that we have successfully logged into openstack.

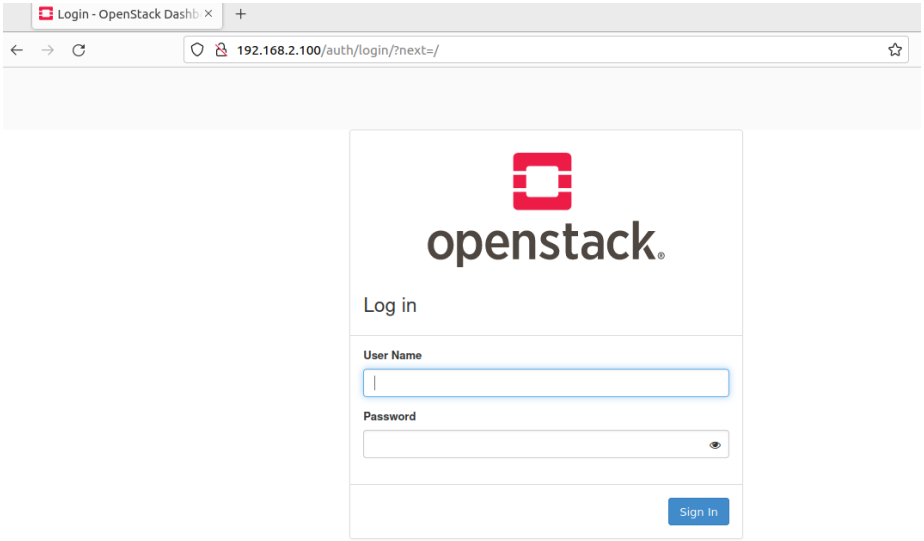


Figure 16: Openstack web Browser login Page

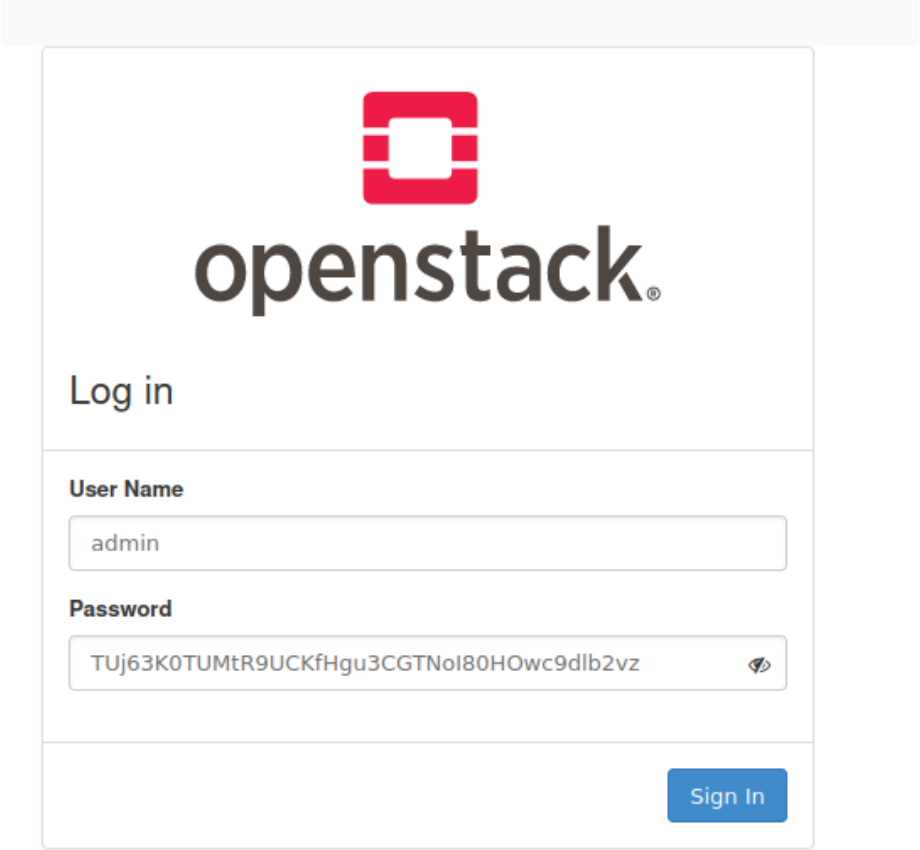


Figure 17: Openstack Login Page with username and passwor filled out

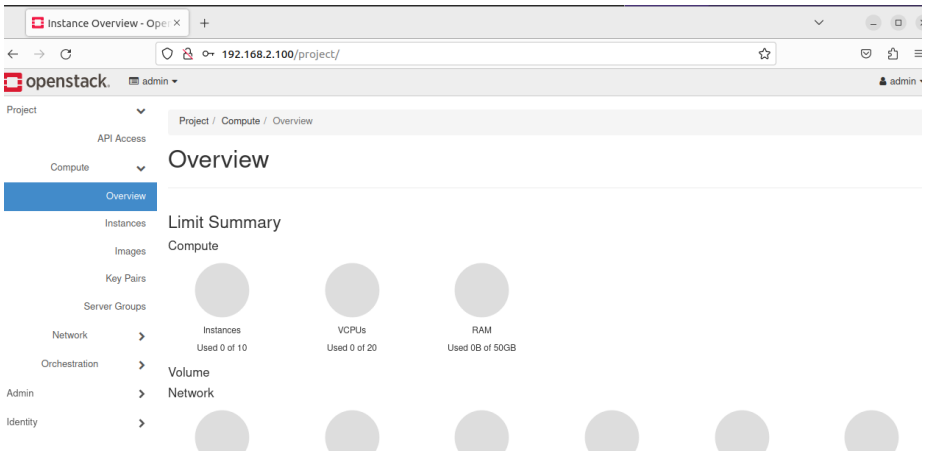


Figure 18: Overview page of Openstack web browser

INIT-RUNONCE

T2: Done. See the evidence (screenshots in images/L2E1S2T2 subfolder).

Q3: (a) Why does your openstack need to be empty right after deployment?

A3: Right after our deployment of openstack there is nothing in the openstack instance because we have yet to add anything image as seen in Figure 20, instance as seen in Figure 19 or networks as seen in Figure 21 sense everything in openstack is customizable there is very little default features of our openstack instance the only one we need on default is the security rule as seen in Figure 22.

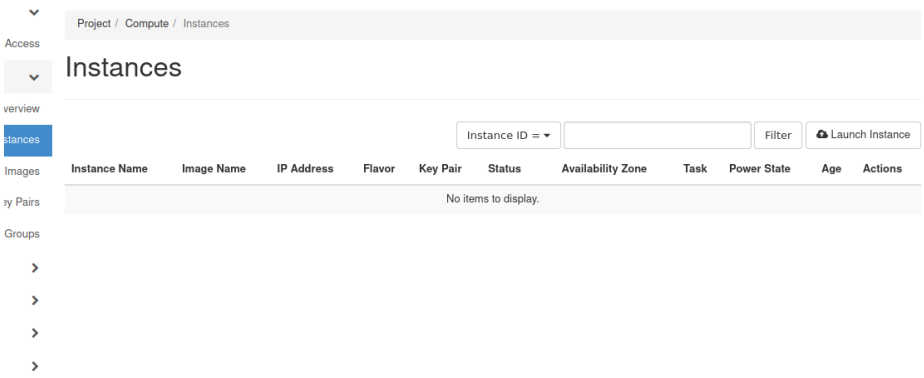


Figure 19: Instance page of Openstack web Browser

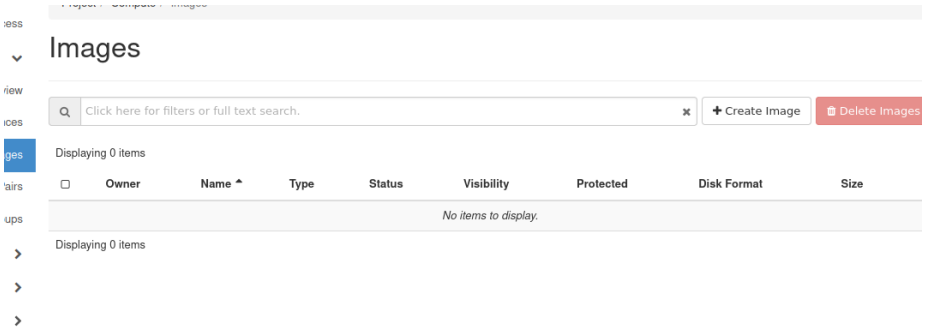


Figure 20: Images page of Openstack web Browser

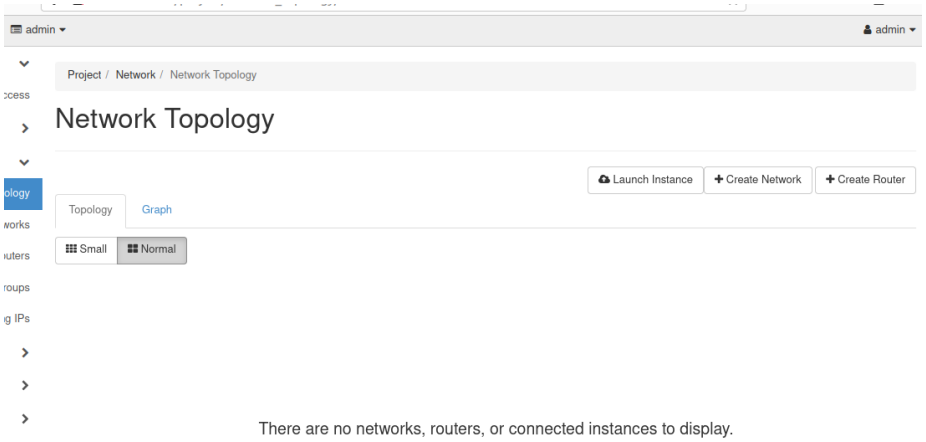


Figure 21: Network Topology on Openstack Web Browser

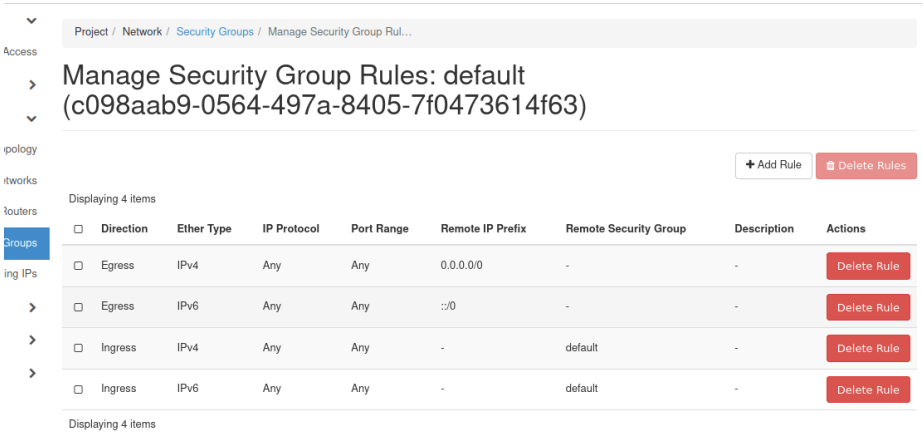


Figure 22: Default Security Group on Openstack Web Browser

Q4: (a) What is Cirros?

A4: Cirros is a Linux distribution with minimal function that is used to test images on cloud instances such as Openstack [7]. Which means it is a low risk image that is just meant to check the basic functions of our cloud deployment are working and not meant for functionality or storage.

Network, router, image, instance creation

T3: Done. See the evidence (screenshots in images/L2E1S2T3 subfolder).

Q5: (a) What does each part of the "openstack Image create" command do?
(b) What is QCOW2 format?

A5: a) The command openstack image creates an openstack image and can take many arguments into the command to customize the image that is created as seen in Figure 23 we used six arguments for our image. We chose the disk format wich is qcow2 which is a way the image is stored, container format which choosed the format of the containers made from the image such as Docker we leave our bare, we mad the image public, we set the os type linux, and we chose the image by putting in the image path to pull the image form and the last argument is the images name which is cirros in our case.
(b) QCOW2 is a storage format for virtual disks [8]. QCOW2 is used to create physical storage by connecting the create instance to a volume on the physical storage of the host to create memory persistence.

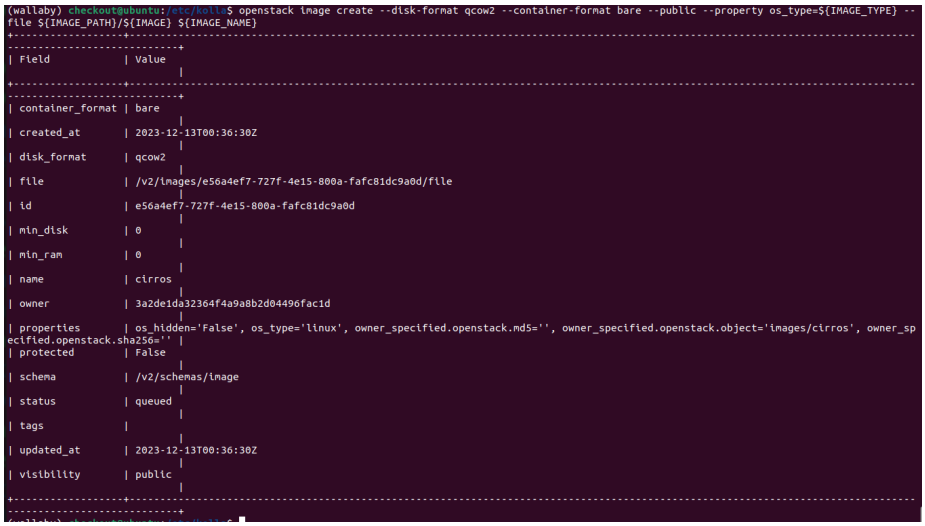


Figure 23: "openstack image Create" Command

Q6: (a) What does each part of the "openstack subnet create" command do?
(b) What is the subnet of the network?

A6: (a) The openstack subnet create command can be seen in Figure 24. The this command has lots of arguments –ip-verison sets the version of the subnet which is ipv4 in this case, –subnet-range sets the range of the subnet in our case that is 10.0.0.2-10.0.0.254, –network selects the network that the subnet is for, –gateway sets the gateway of the subnet in this case 10.0.0.1, –dns-name server sets the dns for the subnet in our case 8.8.8.8 and the last argument is the name of the subnet demo-subnet.
(b) The subnet of the network is 24 which is 255.255.255.0 and represented by the range of ip address is Figure 24 which is 10.0.0.2 - 10.0.0.0.254.

```
(wallaby) checkout@ubuntu:/etc/kolla$ openstack subnet create --ip-version ${IP_VERSION} --su
bnet-range ${DEMO_NET_CIDR} --network demo-net --gateway ${DEMO_NET_GATEWAY} --dns-nameserver
 ${DEMO_NET_DNS} ${SUBNET_CREATE_EXTRA} demo-subnet
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| allocation_pools    | 10.0.0.2-10.0.0.254                  |
| cidr                | 10.0.0.0/24                          |
| created_at          | 2023-12-13T00:39:58Z                 |
| description         |                                      |
| dns_nameservers     | 8.8.8.8                              |
| dns_publish_fixed_ip | None                                |
| enable_dhcp         | True                                 |
| gateway_ip          | 10.0.0.1                             |
| host_routes         |                                      |
| id                  | cb08cbb9-6089-4711-8db1-bdb200108602 |
| ip_version          | 4                                    |
| ipv6_address_mode   | None                                 |
| ipv6_ra_mode        | None                                 |
| name                | demo-subnet                          |
| network_id          | 81110d61-c503-459e-9516-d8addcec874c |
| project_id          | 3a2de1da32364f4a9a8b2d04496fac1d     |
| revision_number     | 0                                    |
| segment_id          | None                                 |
| service_types       |                                      |
| subnetpool_id       | None                                 |
| tags                |                                      |
| updated_at          | 2023-12-13T00:39:58Z                 |
+---------------------+--------------------------------------+
(wallaby) checkout@ubuntu:/etc/kolla$
```

Figure 24: "openstack subnet create" Command

Q7: (a) What does each rule allow?

A7: The first rule we implemented as seen in Figure 25 is an ip table rule that applies to incoming traffic and allows all icmp traffic into our openstack instance. The second rule we implemented as seen in Figure 26 this one is also allowing incoming tcp traffic with the destination port 22 to allow ssh in openstack instances. the third rule we implemented as seen in Figure 27 this is allowing tcp traffic from port 8000 and the fourth rule we implemented as seen in Figure 28 is to allow tcp traffic from port 8080 these are both allowing us to have http traffic so connect to web sites in the instances.

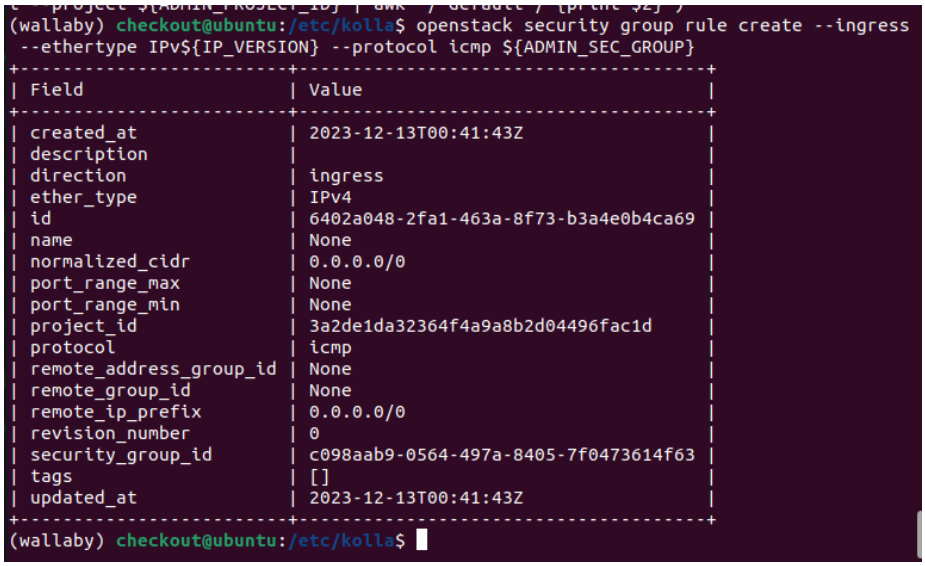


Figure 25: Openstack security group rule create ingress icmp

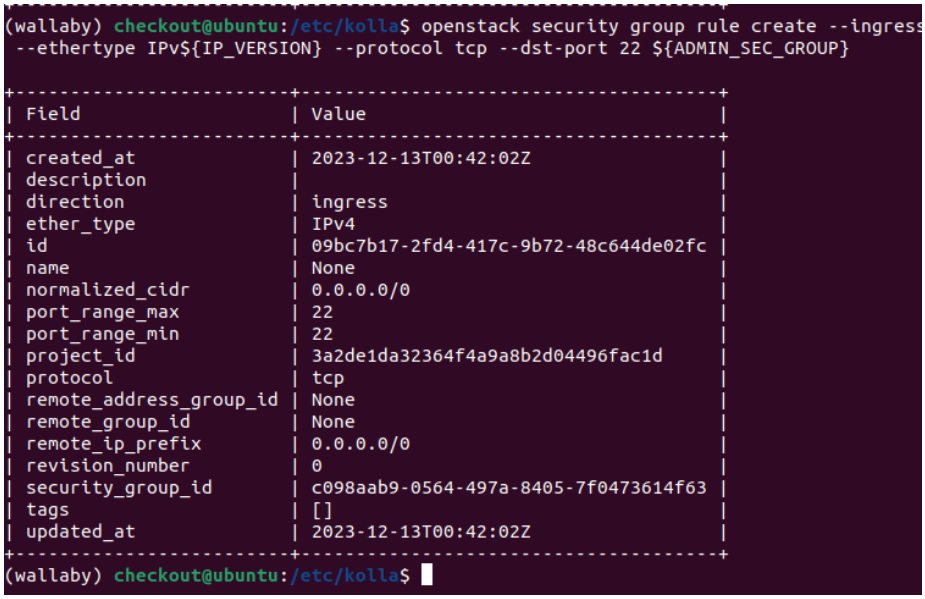


Figure 26: Openstack security group rule create ingress tcp 22

```
(wallaby) checkout@ubuntu:/etc/kolla$ openstack security group rule create --ingress
 --ethertype IPv${IP_VERSION} --protocol tcp --dst-port 8000 ${ADMIN_SEC_GROUP}
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| created_at              | 2023-12-13T00:42:20Z                 |
| description             |                                      |
| direction               | ingress                              |
| ether_type              | IPv4                                 |
| id                      | cc00006e-a6aa-43da-8614-756b407e91d2 |
| name                    | None                                 |
| normalized_cidr         | 0.0.0.0/0                            |
| port_range_max          | 8000                                 |
| port_range_min          | 8000                                 |
| project_id              | 3a2de1da32364f4a9a8b2d04496fac1d     |
| protocol                | tcp                                  |
| remote_address_group_id | None                                 |
| remote_group_id         | None                                 |
| remote_ip_prefix        | 0.0.0.0/0                            |
| revision_number         | 0                                    |
| security_group_id       | c098aab9-0564-497a-8405-7f0473614f63 |
| tags                    | []                                   |
| updated_at              | 2023-12-13T00:42:20Z                 |
+-------------------------+--------------------------------------+
(wallaby) checkout@ubuntu:/etc/kolla$
```

Figure 27: Openstack security group rule create ingress tcp 8000

```
(wallaby) checkout@ubuntu:/etc/kolla$ openstack security group rule create --ingress
 --ethertype IPv${IP_VERSION} --protocol tcp --dst-port 8080 ${ADMIN_SEC_GROUP}
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| created_at              | 2023-12-13T00:42:39Z                 |
| description             |                                      |
| direction               | ingress                              |
| ether_type              | IPv4                                 |
| id                      | cc4effa4-9e9b-42c1-98b5-47aeaa9dd00a |
| name                    | None                                 |
| normalized_cidr         | 0.0.0.0/0                            |
| port_range_max          | 8080                                 |
| port_range_min          | 8080                                 |
| project_id              | 3a2de1da32364f4a9a8b2d04496fac1d     |
| protocol                | tcp                                  |
| remote_address_group_id | None                                 |
| remote_group_id         | None                                 |
| remote_ip_prefix        | 0.0.0.0/0                            |
| revision_number         | 0                                    |
| security_group_id       | c098aab9-0564-497a-8405-7f0473614f63 |
| tags                    | []                                   |
| updated_at              | 2023-12-13T00:42:39Z                 |
+-------------------------+--------------------------------------+
(wallaby) checkout@ubuntu:/etc/kolla$
```

Figure 28: Openstack security group rule create ingress tcp 8080

Q8: (a) Why is it necessary/helpful to do this?

A8: The commands we executed in Figure 29 are openstack quota set commands that set the amount of space or instances we are able to create on our openstack deployment [9]. In Figure 30 we can see the instance

have “Used 1 of 40” listed underneath it and same goes for the RAM and VCPU this is showing us that we set the amount of RAM and VCPUs using these commands. This is helpful because we are able to change the amount of RAM or storage needed for a project at any time.

```
(wallaby) checkout@ubuntu:/etc/kolla$ openstack quota set --instances 40 --force ${ADMIN_PROJECT_ID}
(wallaby) checkout@ubuntu:/etc/kolla$ openstack quota set --cores 40 --force ${ADMIN_PROJECT_ID}
(wallaby) checkout@ubuntu:/etc/kolla$ openstack quota set --ram 96000 --force ${ADMIN_PROJECT_ID}
(wallaby) checkout@ubuntu:/etc/kolla$
```

Figure 29: Openstack quota set commands for instances, cores and RAM

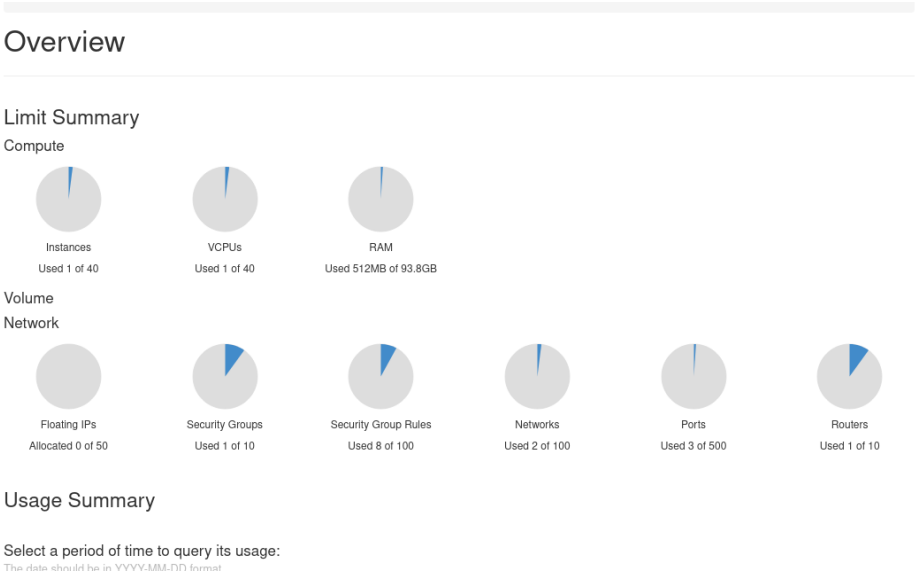


Figure 30: Openstack Overview page in Web Browser

Q9: (a) What is a flavor?
(b) How are they used in OpenStack?

A9: (a) A flavor is a configuration that defines the memory and storage capacity of an instance in Openstack [10].
(b) Flavors are used in Openstack to give instance storage and determine the size of the instance similar to when we allocate processors and memory for VM’s in workstations.

Q10: (a) What do you see in your OpenStack network?

A10: In the openstack Network Topology we can now see a router named demo-router, a public network called public1 and a private network called demo-net with an instance attached to it the public network as seen in Figure 31. We can also see that the public and private networks are connected through the router demo-router.

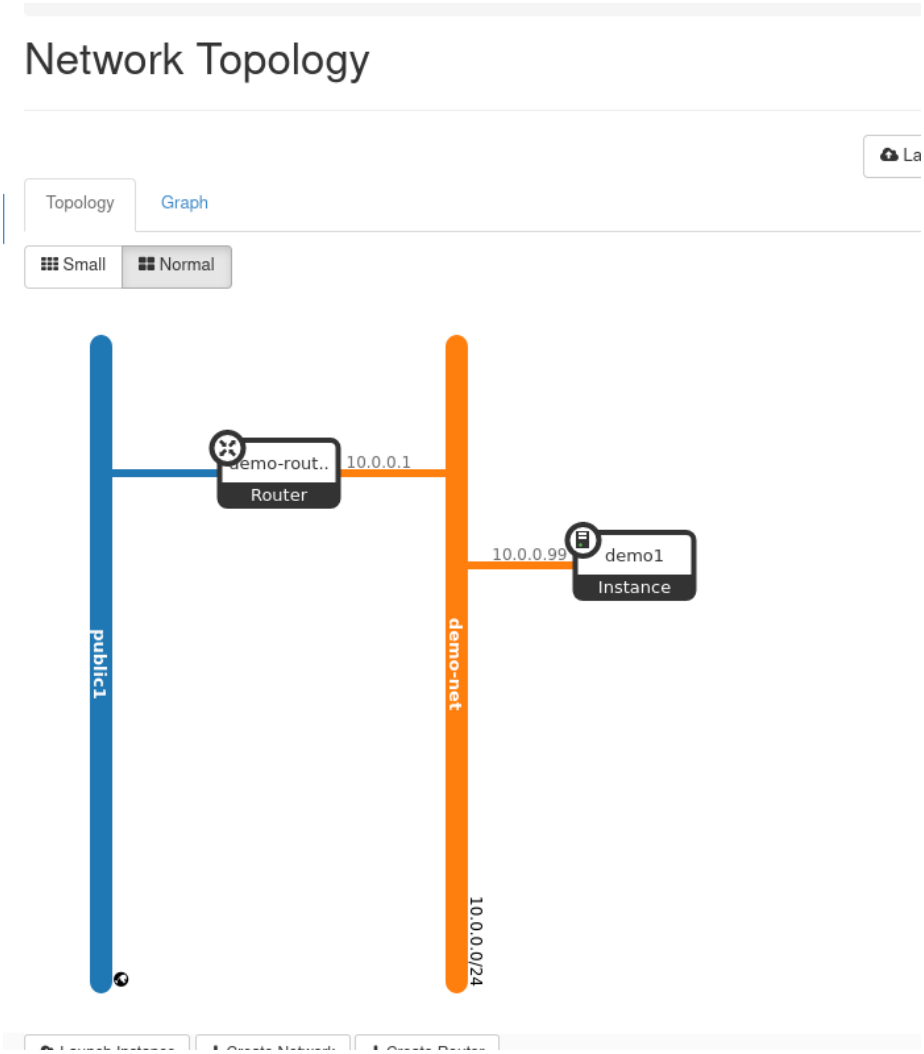


Figure 31: Network Topology in Openstack Web Browser

Q11: (a) Are you successful?
(b) Why or why not?

A11: (a) These pings are not successful as seen in Figure 32.
(b)These are not successful because 192.168.1.1 is not allowed through 192.168.2.1 to allow connectivity to the internet through the openstack instance so we have to add rules to the 192.168.2.1 router to allow there to be connectivity to the internet.

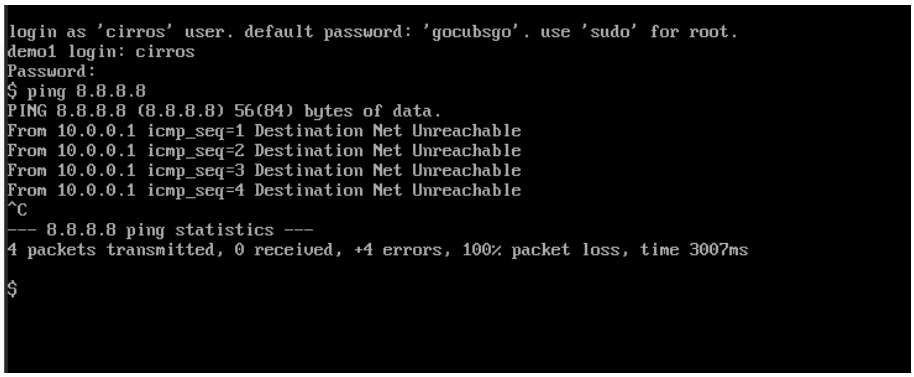


Figure 32: Unsuccessful pings to the internet on Cirros Instance

Q12: (a) What rule allows this process to work?

A12: The rules we added to the 192.168.2.1 route as seen in Figure 33 because it allows the 192.168.1.1 internet address to be allowed through the ap allowing it to have internet connect through the 192.168.1.1 public network of our openstack, we now have connectivity in the instances as seen in Figure 34.

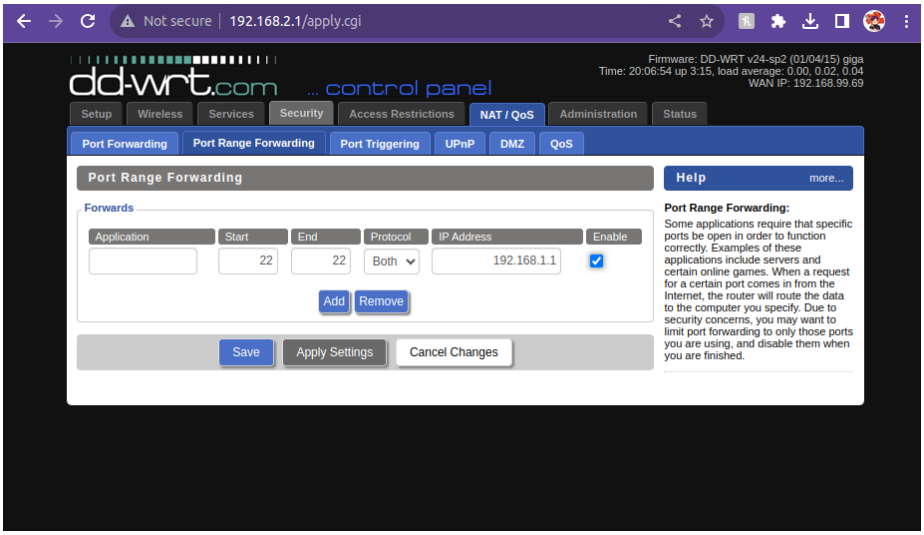


Figure 33: 192.168.2.1 AP Web Browser Port Forwarding Rules

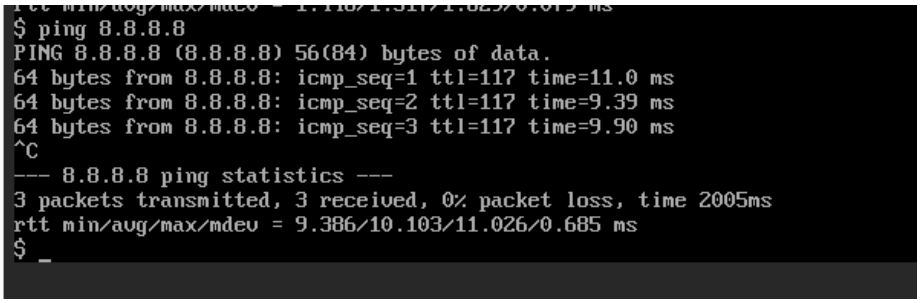


Figure 34: Unsuccessful pings to the internet on Cirros Instance

4 KLT & EVC

4.1 Key Learning & Takeaways (KLT)

The value and impact of this lab is the execution of deploying your own cloud instance and being able to create instance, images and networks to see how these things exist in a cloud space. Understanding the "behind the scenes" of cloud computing and how it is managed not just used in the web browser.

4.2 Expectancy-Value-Cost (EVC)

Expectancy

Yes I was able to complete this lab and found the steps and tasks well organized with a logical flow in topics.

Value

The Value of this lab is the understanding of how openstack is deployed and what its like to customize your own cloud services. As well as the basic over learning how to use openstack as an administrator and create images, instances and networks through the command line.

Cost

This lab was executed while also doing many other things so the cost was mostly just finding time to finish it.

5 Lab Observations, Suggestions & Best Practices

5.1 Observations

All observations were changed while completing this lab.

5.2 Suggestions

We suggest that this lab should use a different image then Cirros preferably a Ubuntu image. This would better help students see the benefit in openstack as they could see the environment they are most used to being used in a cloud setting. We would also suggest expanding the lab to have students create images, instances and networks in the GUI to see the process and learn how to openstack through the user perspective.

5.3 Best Practices

Our best practices during this lab was editing the lab as we went through to make sure it made sense and had a good flow as well as touched on all the points we wanted to get a cross while doing the execution of this lab.

6 Lab References

[1] Openstack. "Operating kolla." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.openstack.org/kolla-ansible/latest/user/operating-kolla.html.

[2] Openstack. "Openstack wallaby enhances security and cross-project integration with other open source technologies." Retrieved on 2023-12-12. (2023), [Online]. Available: https://www.openstack.org/software/wallaby/.

[3] Opendev. "Kolla ansible." Retrieved on 2023-12-12. (2023), [Online]. Available: https://opendev.org/openstack/kolla-ansible#:~:text=The%20Kolla%20Ansible%20is%20a,tools%20for%20operating%20OpenStack%20clouds..

[4] Github. "Openstack-ansible." Retrieved on 2023-12-12. (2023), [Online]. Available: https://github.com/openstack/openstack-ansible.

[5] openstack. "Etcd for ubuntu." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.openstack.org/install-guide/environment-etcd-ubuntu.html.

[6] openstack. "Kuryr." Retrieved on 2023-12-12. (2023), [Online]. Available: https://wiki.openstack.org/wiki/Kuryr.

[7] Openstack. "Centos." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.openstack.org/image-guide/obtain-images.html.

[8] Redhat. "Storage formats for virtual disks." Retrieved on 2023-12-12. (2023), [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.3/html/technical_reference/qcow2#:~:text=QCOW2%20is%20a%20storage%20format,between%20logical%20and%20physical%20blocks..

[9] openstack. "Quota list." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.openstack.org/python-openstackclient/pike/cli/command-objects/quota.html.

[10] vmware. "Openstack flavors." Retrieved on 2023-12-12. (2023), [Online]. Available: https://docs.vmware.com/en/VMware-Integrated-OpenStack/7.3/com.vmware.openstack.admin.doc/GUID-79C6C57F-81C9-43F8-8758-872131B0EDB3.html#:~:text=In%20OpenStack%2C%20a%20flavor%20is,vCPUs.

7 Acknowledgments

We would like to acknowledge Professor Salib for helping during the creation of the instructions and the continued editing of them. We'd also like to acknowledge him for the help to learn and understand these topics last semester in order to put this lab together.

Appendix G - Symposium slides

Symposium Report

OpenStack
Private Cloud Computing
Containers and Instances from Anywhere at Anytime

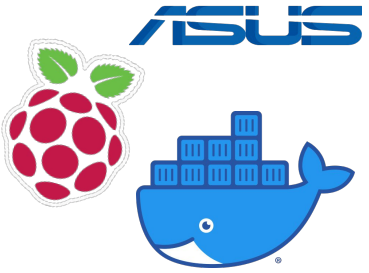
Casey Alexander and Katherine Botticelli

Advised by: Dr. Emil Salib

Contents

- Motivation
- Approach/Justification
- Setup (Network Diagram)
- Use Cases/Demos
- Challenges
- Next Steps
- Deliverables
- Pilot Trials

Motivation

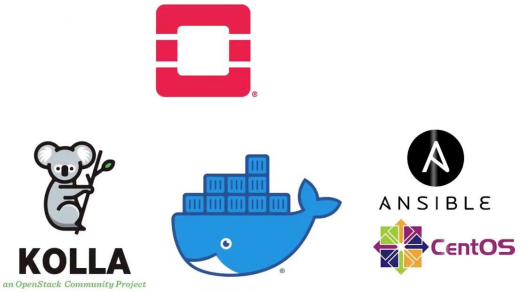


- Excess individual hardware
- Individual lab hardware is not accessible remotely
- Hardware resources are not dynamically allocated
- Execution of assignments only in lab
- Increasing class size

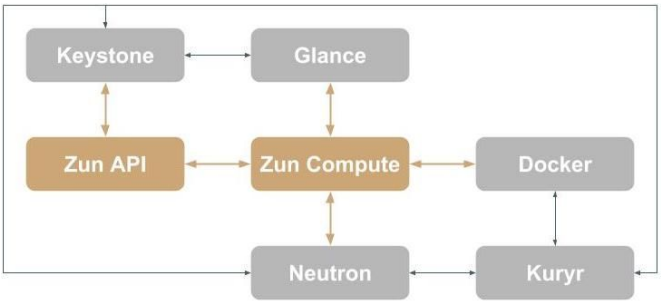
Approach: OpenStack

OpenStack is a free, open source cloud computing platform used as both public and private.

- Private cloud
- Kolla-ansible deployment
- Ease of user accessibility
- Software defined networking provides flexibility

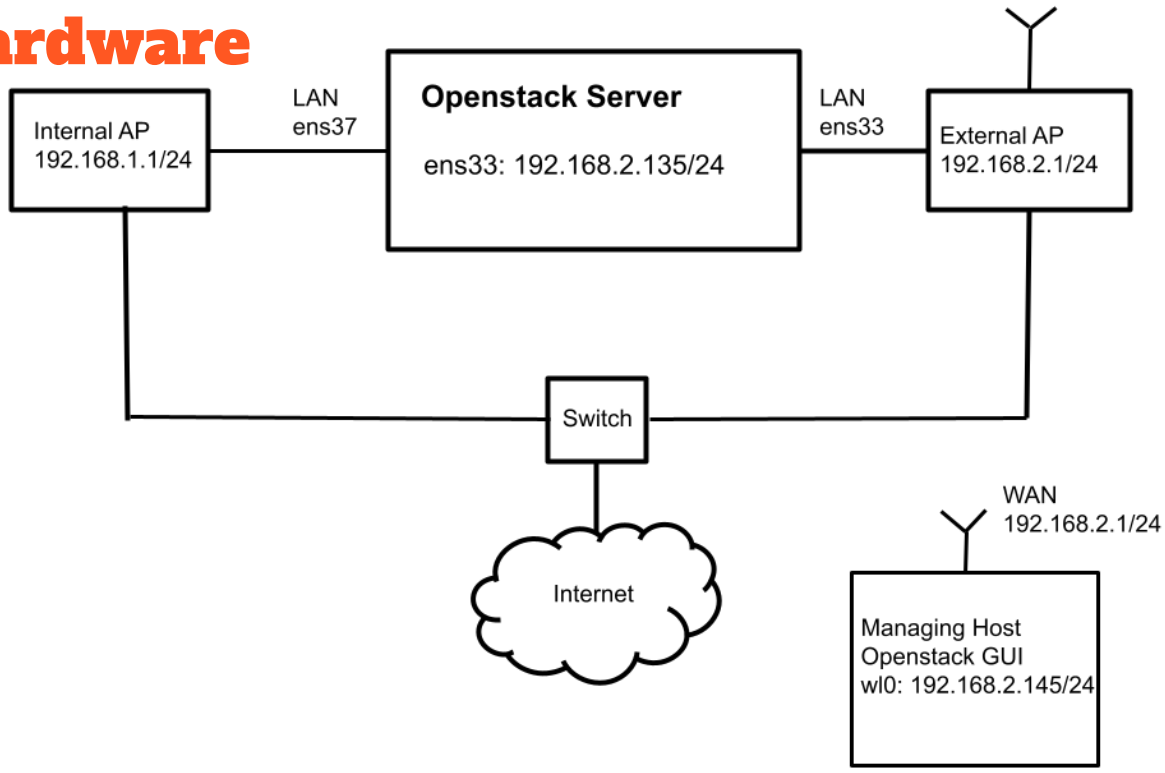


Justification

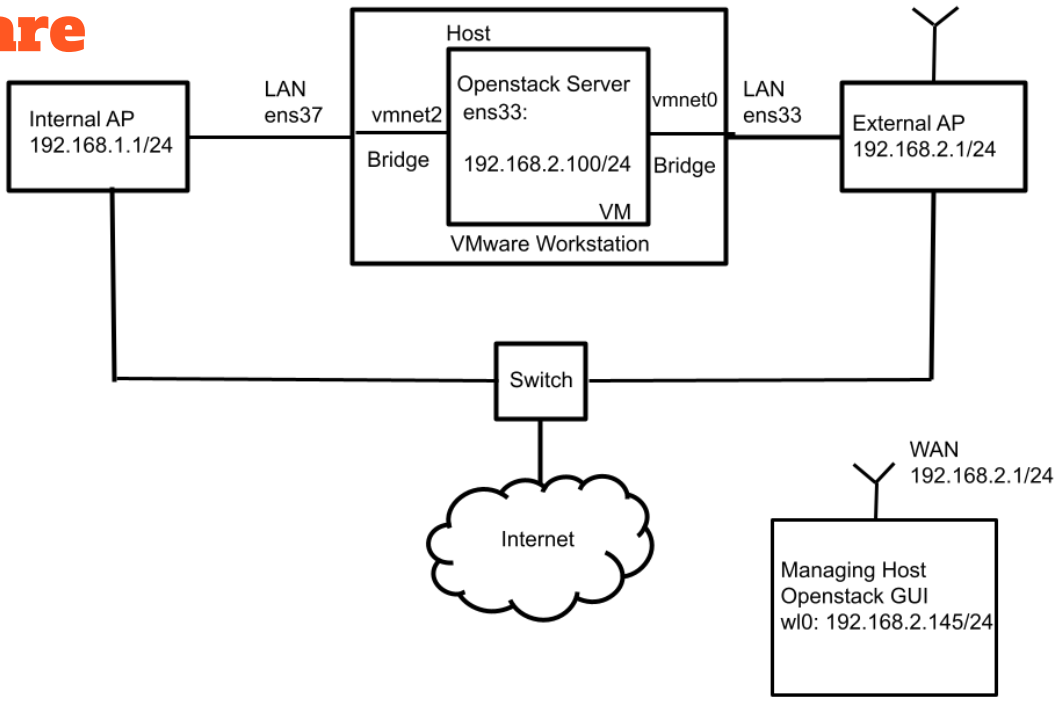


- Customizable
- Cost effective
- Security between projects, users
- Teach how to create and how it works, not just usage
- Provide ability for Kubernetes

Setup: Hardware



Setup: Software



Use Cases

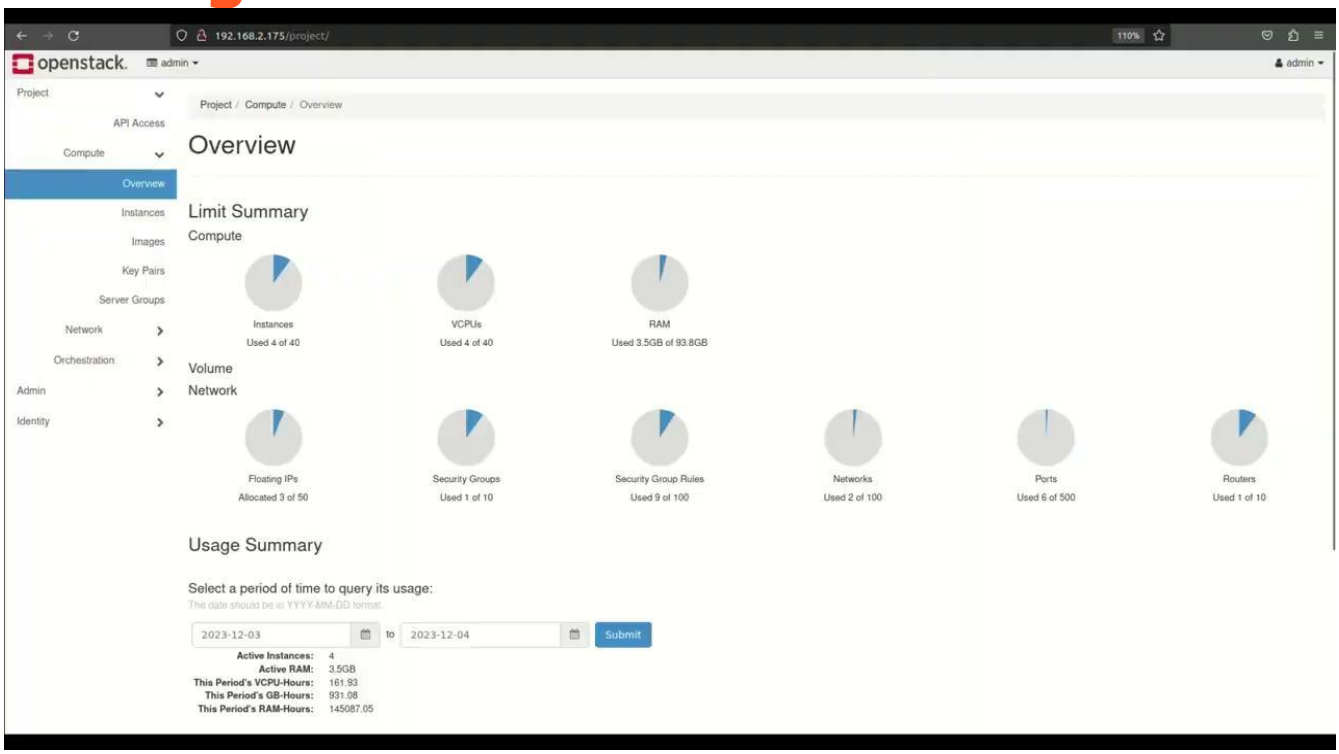
Admin

- Create deployment
- Create public network
- Create multiple users, on multiple projects

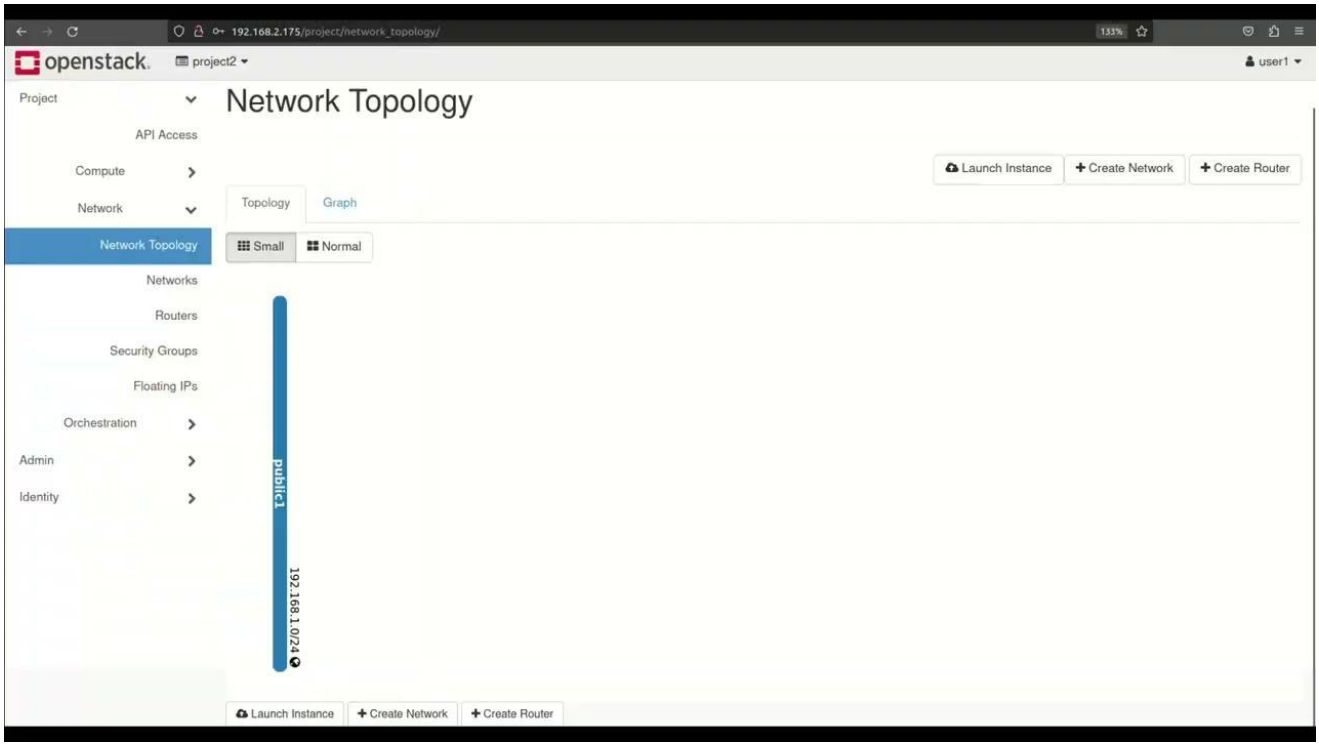
User

- Create private network
- Create instances
- Communication between two instances, instances to the internet
- Mongo and Mongo-Express Container Communications

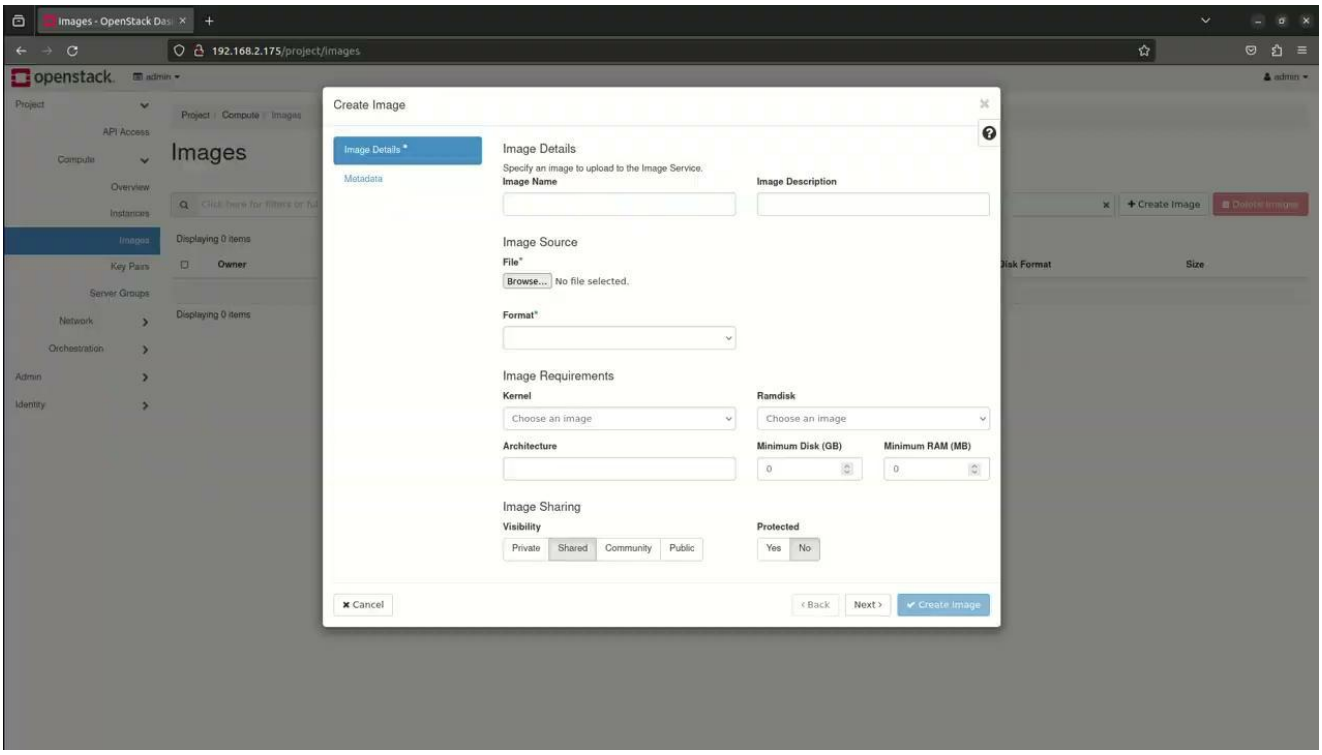
Accessing a new user



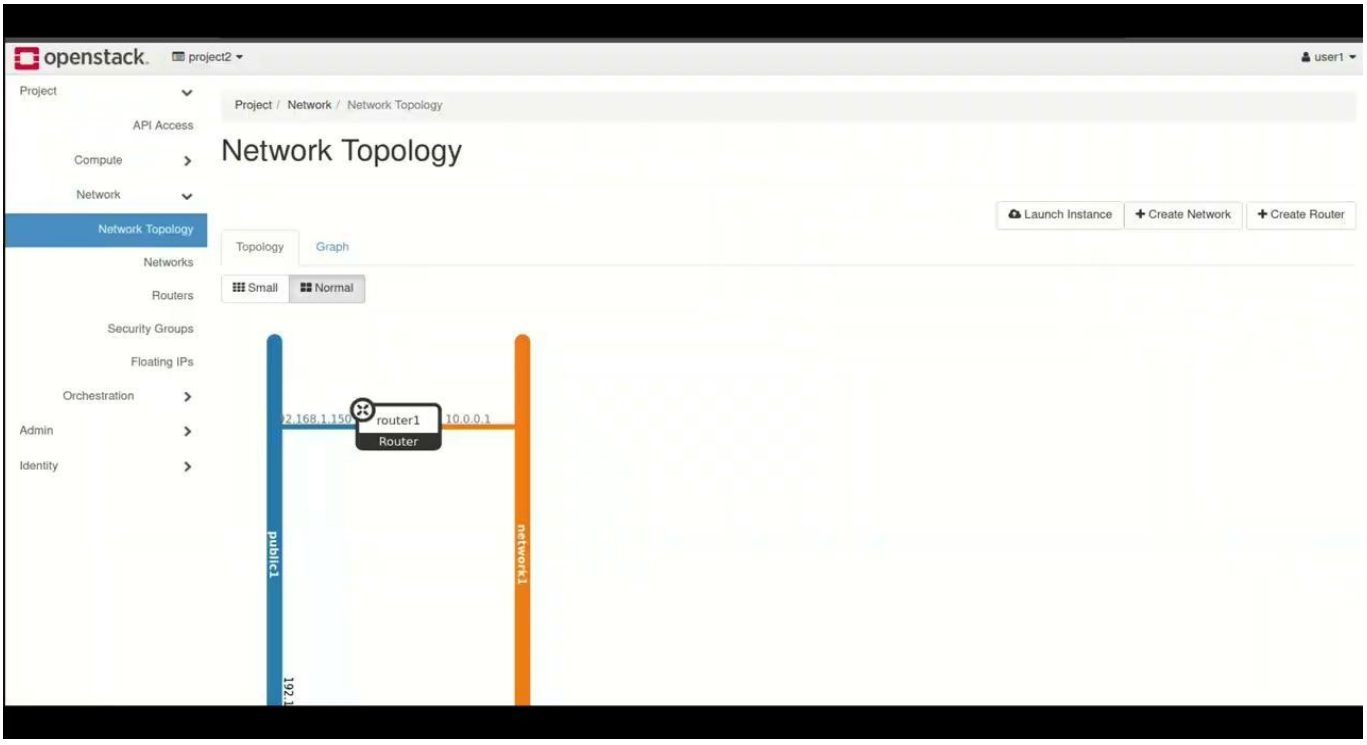
Setting up Network



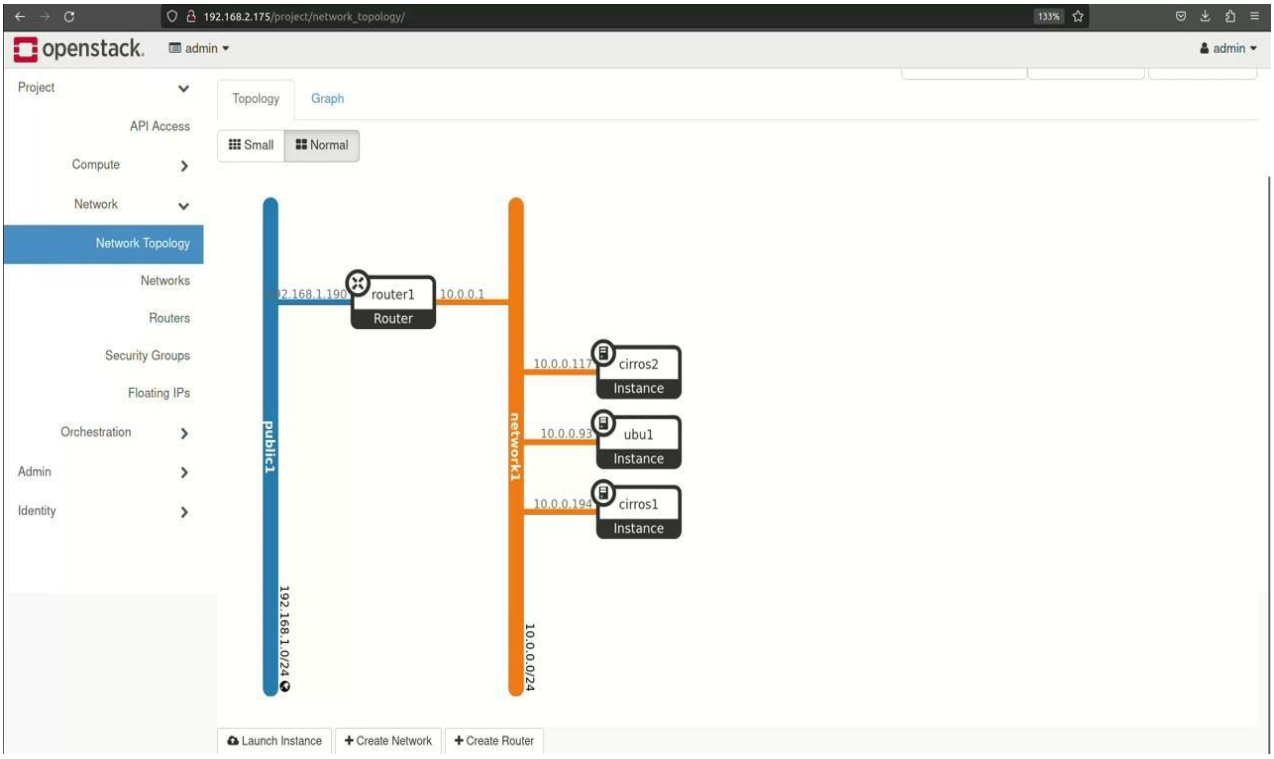
Creating an OpenStack Cirros Image



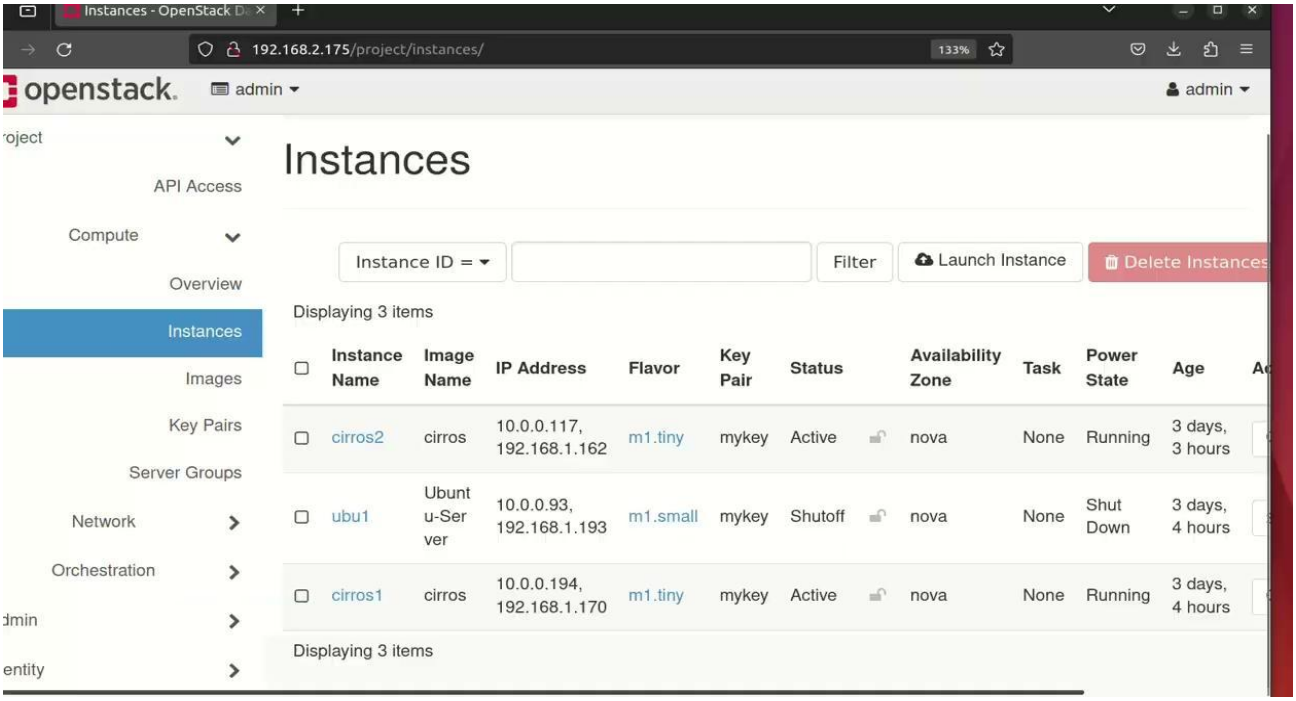
Creating VM Instance



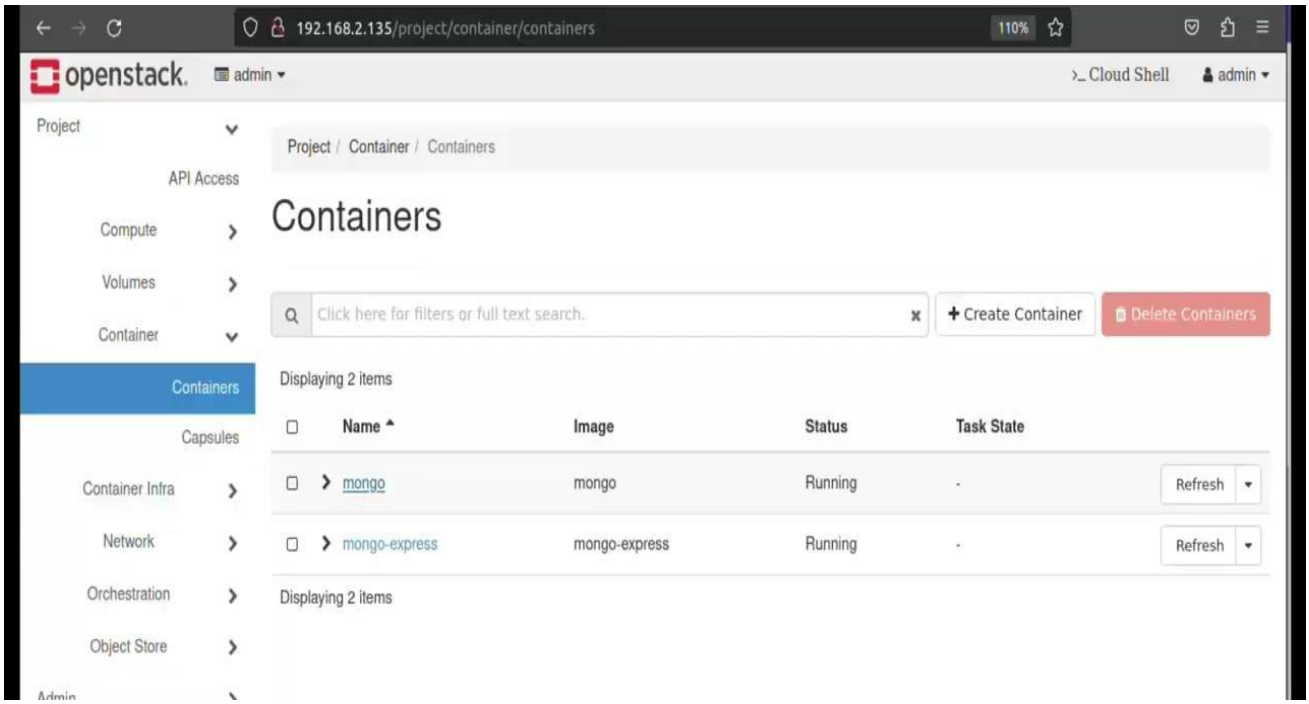
Communication Inside Instances



SSH Into Instance, Communication from Outside



Containers in OpenStack



Challenges

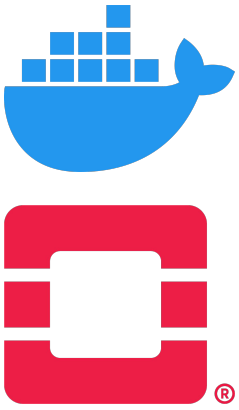
- Customization
- Solving errors
- Unfamiliarity
- Time

Next Steps

- User testing from home (anywhere, anytime)
- Create custom images
- Migrate labs with excess tools to OpenStack
- More OpenStack labs

Deliverables: Labs

- Lab 1 - Docker
- Lab 2 - Openstack deployment + configuration
- Lab 3 - Openstack usage for multiple users



Pilot Trials

- Tested lab instructions
- Checked clarity and accuracy
- Resulted in feedback to alter instructions
- More planned

Acknowledgements

- We would like to acknowledge and thank the unwavering help and support of Professor Salib
- We would like to thank Jenny for her help providing hardware and solving hardware issues quickly
- We would also like to thank Kagen for being a pilot tester so far and helping improve our final deliverables

Thank you!
Questions?