JAMES MADISON UNIVERSITY

COLLEGE OF SCIENCE & ENGINEERING (CISE)

INFORMATION TECHNOLOGY (IT)

IT 444 - CAPSTONE PROJECT DESIGN

Proposal: Labs on Docker Images and Containers using Cloud Computing Environments

Wednesday 10th May, 2023

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

Advisor:
Dr. Emil H. SALIB
Students:
Casey ALEXANDER &
Katherine BOTTICELLI

Contents

List of Attachments

1 Executive Summary

Our capstone is focused on containerized cloud computing, focusing specifically on Docker containers and the networking aspects of these containers. We will be creating lab exercises across various topics with the intention of putting them towards a new IT major course in the future. The lab exercises will be done with the help of public/private cloud platforms such as AWS or OpenStack and with the help of Kubernetes to implement different networking aspects for future IT students to learn about networking through the cloud. The labs' main purpose will be to teach students how to use Docker, how they can create docker images with a Dockerfile, docker compose files and their function, how they can bring up containers from their own images and much more in terms of simple aspects in Docker. The more advanced labs will go into teaching them how to apply networking features such as routing, ip forwarding, and even security aspects of networking to their containers. After completing these labs, we hope students have a better understanding/initial understanding of cloud computing and how containers play a role in cloud computing.

2 Related Work & Literature

Comparative experimental analysis of Docker container networking drivers

This related work dives into the importance of network drivers in container network such as Docker containers. This article focuses on different networking options for Docker their performance [1]. This paper has a focus on what bridge type would be best for different application workloads. Their findings were that the higher the bandwidth for an application use the Docker bridge macvlan bridge over the simpler bridge types such as overlay.

This related to our capstone as we are using and experiment with the bridge type in Docker when doing different network configurations and using the research they collected for this paper helps us better understand how the bridge type effects your Docker network and communications.

To Docker or Not to Docker: A Security Perspective

This paper gives an overview of the Docker setup such as how repositories, the docker daemon and how the memory and storage work in a Docker. They are comparing the container elements of Docker to a Linux container that also is a form of cloud container but has a different "container ecosystem" then Docker. This paper focus on the security concerns that Docker has , less of the security between Docker and Host but Docker containers security from other Docker containers. This paper goes into the different attacks and vulnerabilities that exist in Docker [2].

This paper is similar to our approach as it talks about how docker networking works such as ports and Docker containers being created on the Hosts network. This paper only acknowledges security issues with Docker and doesn't really give any suggesting for fixing these security issues other then security features Docker has by default. This is more of a paper just to help others understand the security risks of Docker and not what to do to fix them.

3 Research Design Process

This section will summarize the work that has been done thus far through our Design phase. The process we went through essentially led us to the ideas we have now. As we began, most of the work was research based. We had not yet experimented with the creation of Docker images, nor was it our first choice at the time. We had only heard about Kubernetes and that is what had peaked our interest. We felt strongly at this time about cloud computing as well. This is not to say that we are not now, just that the process that actually happened in the design phase was not what we had expected.

Something we were so fortunate to have was the knowledge of Dr. Salib on the path of starting Kubernetes without Docker first. We knew from his experience that this process will run us into way more speed bumps than necessary. For this reason, we pivoted our main focus to be Docker. Even though it was only our second real Capstone meeting, it was a turning point in the capstone for us. This is because we were so hooked on these ideas of cloud computing and Kubernetes, but we would have struggled to implement them without the extensive background knowledge of Docker first. The first hands-on work that we began doing was a YouTube video titled ”Docker Tutorial for Beginners” [3]. This video was a great speed to follow along with, in-depth, had thorough explanations, and was a great first project. We appreciate this video as well for the struggle that it presented to us, giving us a chance to problem solve these issues and learn that much more.

We thought that the video and the accompanying demo in our own environment would be done in just a week. This week was a lesson to us on how much work was needed to be done for us to accomplish all that we had in mind. By the following week, it was about halfway finished. Finishing this video and its demo ended up taking three weeks total. We then had a demo containing multiple Docker containers, some pulled from Docker hub, some created by us, a detailed docker-compose file, a working node.js server to start a Mongo Express instance and run a website. We learned here all about Dockerfile, docker-compose, docker volumes, docker repositories, using an AWS repository to push to and to pull from. During the creation of the demo we also created a baseline VM containing Docker, VSCode, and each of our AWS repository credentials so that any time we needed a fresh VM, we would start from a good place. We then wrote long and detailed instructions of the demo we had done which was our first attempt at writing lab-style instructions.

At this point, we had learned so much so quickly, so we took a week really dedicated to researching the topics we had just been experimenting with. We took a look into Dockerfiles, data persistence options, docker volumes save location, and docker volumes file access location. From this research and our hands-on work from before, we attempted to create our own demo environment as at this point we had a very good understanding of the Docker processes we were experimenting with. The remainder of the semester was spent experimenting with Dockerfiles, docker-compose configurations, and finished off with a demo - with the help of Dr. Salib - demonstrating how to include static routes and certain configuration in the docker-compose file to allow one container to act as a router and allow communication through it.

After starting from zero knowledge of Docker, there is much to learn still as we go into the integration of Kubernetes, AWS, OpenStack, and many more possibilities as we go into building labs to commemorate what we have learned and more.

4 Project Objectives and Deliverables

4.1 Opportunity Statement

We are attempting to combat the fact that VMs are used solely in the networking aspect of the IT major without integration of the other options that exist. There is much value to be gained from not only introducing Docker [4], but also integrating Docker and VMs into a working environment together. We are interested in experimenting with the possible integration of Kubernetes, OpenStack, and AWS to increase the overall functionality of the environment as well. The integration of AWS and OpenStack provides an aspect of cloud computing that would be beneficial to know. Although only a few choices are listed here, we are open to exploring any options that will benefit the labs and the learning they provide.

AWS and OpenStack are two cloud services that we are looking into to integrate into the creation of our labs [5]. AWS is a public and private cloud computing service that allows the user to setup an account and start using their different cloud services such as ECR and Kubernetes. The benefit of this includes being able to push Docker containers to the cloud service via the Elastic Container Registry (ECR) as well as use the cloud in different locations. If we only want to use the public cloud service of AWS, we will be using the ECR mainly because of its ability to store containers for use [6]. If we want to experiment creating and configuring our own private cloud, then we would do so with the guidance and permission of Dr. Salib in hopes of utilizing this cloud for the whole class, or even whole department. We initially decided on using AWS over OpenStack because AWS is more secure through Amazon's infrastructure [7]. Despite having this opinion we are still open to trying OpenStack and are willing to reconsider as we get to know OpenStack better. The main disadvantage we are finding it the cost of running our own private cloud.

We believe it would be efficient to use the cloud in conjunction with our Docker environment because the containers would be able to be pushed into the ECR as well as pulled back into the Docker environment. This can happen from any computer and any location which is what makes this efficient as mobility is important. The files that create Docker images would be stored with the student and therefore not needed to be pushed anywhere. Docker-compose files are able to bring up these containers after they are pulled from the ECR as well.

We have decided to use Docker as our container system over LXD because we have spent a significant amount of time using, learning, and experimenting with Docker at this point. After researching LXD [8], we found that Docker is more portable, can be integrated with tools like Kubernetes and almost all public clouds while LXD cannot, as well as that Docker is more widely known and used as the industry-leader. We have grown to feel very comfortable using Docker that it would not be advisable for us to change paths now [5].

Introducing a topic such as Docker in a lower level class is most effective as students are still open to brand new methods. If new IT students are introduced to Docker before they get to the real VM heavy courses such as IT460 and IT333, then the students would likely be more receptive to the new concepts of networking and be able to adapt more quickly. These adaptation skills will be important for their futures, as it is likely that brand new things will be introduced to them often and even in the following IT classes they will face. We are planning to introduce Docker in a possible new course that focuses on cloud computing using Docker (and possibly Kubernetes), in order to expand the knowledge and experience of the future IT students. With this new class, we'd also be enabling the instructor to introduce to undergraduate students curriculum that includes hands-on learning and exercises focused on topics that may not have been possible if the current solely VM approach had remained.

4.2 Proposed Solution

Our proposed approach towards a solution to this problem is to teach ourselves the ins and outs of Docker. From the experience we have gained thus far and will continue to grow upon, we plan to create educational labs to use Docker containers, networks, images, etc. instead of multiple VMs like is done in many labs currently. We were planning on taking 3-5 labs that are used in IT major classes currently and convert them to be Docker based instead of virtual machine based. We have strayed from this idea because we felt that having to work off of pre-established topics would box us in too much. It would reduce the full understanding of both VMs and Docker as it would limit the capabilities of both. We felt that during the transition a lab would lose too much of its value and of its material that would make it less beneficial to the student. We want the Docker based labs to be focused around topics that Docker works well with and was made for, not just topics we have done in VMs before. Since we will be creating new labs from scratch, we will not be losing function from the lab as we convert it from VMs to Docker. Instead, the student will now be able to go into just as deep of function in both the VM based learning and Docker as well.

We believe creating an Introduction to Docker lab would be necessary/beneficial for the students to go on to dive deep into Docker and its functions. Some other solution ideas can include a mixture of VMs with Docker, a mixture of Docker with Kubernetes, or docker with GNS3 and the inclusion of hardware as well.

The first lab that we would create start to finish would include general and introductory tools to Docker. This (roughly) could include adding yourself as a user, how to pull from docker hub, checking your images, running images, creating your own images, starting them, starting them from docker-compose, alterations in docker-compose, pushing to AWS repository, pulling from AWS repository. These topics can and likely will change/expand.

Another idea we could expand on is creating Docker images that enable certain networking tasks and deploy them to a public Github, allowing other curious students, teachers, and professors to experiment using these images or teach using them in actual implementation.

For the labs we will create after the intro, we have a wide range of topics we can choose from and experiment with. The topics are not chosen yet, but we plan for them to be selected from researching other projects that have been done, along with the help of Dr. Salib for topic creation. We can say for sure that we want to include basic networking tasks within these labs such as setting and checking IP addresses/MAC addresses, monitoring traffic within containers + on the created interfaces for containers on the host, checking logs of built containers as they start + as they are altered, altering networks within docker-compose, adding, changing iptables rules/static ip routes, etc. There truly is no defined list as we still have much to learn and research before these decisions are solid.

4.3 Project Goal & Objectives

The project goal is to create brand new labs with the intention of them being taught in a new IT course focusing on Docker and Kubernetes containers and container orchestration. To achieve this goal, we set out to accomplish the following objectives:

- Teach ourselves Kubernetes functions in order to plan to supplement labs with the integration of it
- Implement an introductory lab to Docker processes and Docker commands
- Provide teachings to create and use private AWS repository

- Create labs to expand on knowledge of Docker processes and go deeper into available functionality
- Integrate Kubernetes, GNS3, AWS/OpenStack into the learning material of the labs

We plan to set specific deadlines and goals for ourselves in order to stay on track and monitor our progress. We will closely follow the calendar that we have created (still to be edited) and have planned deliverables per week. Setting smaller goals on the path to the final goal will help us make sure that we are consistently making progress.

4.4 Deliverables

We plan to submit educational labs regarding Docker; each with Exercises, Steps, Tasks, and Questions to help students better grasp the topics. We have not yet decided the number of labs this will be as we are unsure the scope or length it will require per lab. We will also be submitting a final report that acts as a summary of both our design, implementation, and result sections. More informally, we will be submitting what we set for ourselves each week. To summarize, we plan to submit:

- Symposium presentation and hands-on demos
- Capstone project report
    1. Lab instructions
    2. Lab reports
    3. Environment setup

5 Timeline/Milestones

5.1 Proposed Timeline For Fall Semester 2023

Schedule	
Week	Updates
1	Status-1/Report-1/Demo-1
2	Status-2/Report-2/Demo-2
3	Status-3/Report-3/Demo-3
4	Status-4/Report-4/Demo-4
5	Status-5/Report-5/Demo-5
6	Status-6/Report-6/Demo-6
7	Status-7/Report-7/Demo-7
8	Status-8/Report-8/Demo-8
9	Status-9/Report-9/Demo-9
10	Status-10/Report-10/Demo-10
11	Status-11/Report-11/Demo-11
12	Status-12/Report-12/Demo-12
13	Dry Run-1/Draft Report-1
14	Dry Run-2/Draft Report-2
15	Final Draft Presentations/Draft Report-3
16	Final Report

6 References

[1] L. L. Mentz, W. J. Loch, and G. P. Koslovski, *Comparative experimental analysis of Docker container networking drivers*. 2020, pp. 1–7. DOI: 10.1109/CloudNet51028.2020.9335811.

[2] T. Combe, A. Martin, and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016. DOI: 10.1109/MCC.2016.100.

[3] T. with Nana. "Docker tutorial for beginners [full course in 3 hours]." Retrieved on 2023-01-30. (2021), [Online]. Available: https://youtu.be/3c-iBn73dDE.

[4] docker docs. "Docker overview." Retrieved on 2023-04-28. (2023), [Online]. Available: https://docs.docker.com/get-started/overview/.

[5] S. Singh. "Lxd vs docker." Retrieved on 2023-04-23. (2023), [Online]. Available: https://www.educba.com/lxd-vs-docker/.

[6] aws. "Amazon elastic container registry." Retrieved on 2023-04-23. (2023), [Online]. Available: https://us-east-1.console.aws.amazon.com/ecr/get-started?region=us-east-1.

[7] ProjectPro. "Openstack vs aws - is aws using openstack?" Retrieved on 2023-04-23. (2023), [Online]. Available: https://www.projectpro.io/article/openstack-vs-aws/482\#:\~:text=AWS\%20vs.,-OpenStack\%20\%2D\%20A\%20Head\&text=OpenStack\%20offers\%20easier\%20management\%20of,cloud\%20management\%20through\%20Amazon's\%20infrastructure..

[8] L. containers. "What is lxd?" Retrieved on 2023-04-23. (2023), [Online]. Available: https://linuxcontainers.org/lxd/introduction/.

[9] D. Docs. "Docker and iptables." Retrieved on 2023-04-12. (2023), [Online]. Available: https://docs.docker.com/network/iptables/.

[10] D. docs. "Puse bridge networks." Retrieved on 2023-04-12. (2023), [Online]. Available: https://docs.docker.com/network/bridge/.

[11] Silicium14. "Change default route in docker container." Retrieved on 2023-04-12. (2016), [Online]. Available: https://stackoverflow.com/questions/36882945/change-default-route-in-docker-container.

[12] D. Inc. “Docker official image.” Retrieved on 2023-02-06. (2023), [Online]. Available: https://hub.docker.com/.

[13] T. Tim. “Build your own dockerfile, image, and container - docker tutorial.” Retrieved on 2023-03-06. (2022), [Online]. Available: https://www.youtube.com/watch?v=SnSH8Ht3MIc\&t=306s.

[14] D. Inc. “Manage data in docker.” Retrieved on 2023-03-06. (2023), [Online]. Available: https://docs.docker.com/storage/.

[15] D. Inc. “Volumes.” Retrieved on 2023-03-06. (2023), [Online]. Available: https://docs.docker.com/storage/volumes/.

[16] docker. “Docker community forums.” Retrieved on 2023-03-06. (2020), [Online]. Available: https://forums.docker.com/t/how-to-access-docker-volume-data-from-host-machine/88063/6.

[17] Microsoft. “Dockerfile on windows.” Retrieved on 2023-03-27. (2023), [Online]. Available: https://learn.microsoft.com/en-us/virtualization/windowscontainers/manage-docker/manage-windows-dockerfile.

[18] BMitch. “Provide static ip to docker containers via docker-compose.” Retrieved on 2023-03-29. (2016), [Online]. Available: https://stackoverflow.com/questions/39493490/provide-static-ip-to-docker-containers-via-docker-compose.

Appendix A - Weekly Status Updates

First To Do List - 1/27/2023

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

i. research, understand Kubernetes + its capabilities in the public and private cloud space, owned by both, for next week

ii. list different private cloud options, owned by both, for next week

Status #1 To Do List - 1/30/2023

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

i. decide private vs public cloud space

ii. now understand that Kubernetes cannot host a cloud environment on its own, acts as a supplicant with docker to cloud

iii. Docker, Kubernetes, Cloud

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

i. YouTube video - Docker video [3]

ii. create a copy environment from video

Status #2 To Do List - 2/6/2023

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

i. Watching the YouTube video - Docker Video is still in progress as the video is taking longer than expected.

ii. Creating a copy of the environment from the video is in progress.

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

i. Have a full, functioning docker environment

Status #3 To Do List - 2/13/2023

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

i. The Docker information video is halfway finished

ii. Have a Docker environment set up on Linux VM.

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

i. Create a baseline VM including Docker and all other apps wanted to make clones of in the future

ii. Develop lab instructions that enable you to do the demo; starting with docker already downloaded (20 mins)

iii. Demonstrate a working mongo-express GUI + application as shown in video

Status #4 To Do List - 2/20/2023

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

i. The Docker information video is fully finished

ii. We both have a Docker environment matching the full demo from the video

iii. We both have an instruction page detailing the demo process

iv. We both have a Baseline VM to work the demo from

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

i. Analyze mongo and mongo-express images that we pulled from the public Docker repo to understand them

ii. Apply volumes to our Docker environment to enact Data Persistence

iii. AWS Completion (Katie)

Status #5 To Do List - 2/27/2023

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

   i. Some research was done, but unable to find what truly was requested. Unable to find information about the actual pieces inside the mongo docker image or mongo-express docker image.

   ii. We applied volumes to enable data persistence within the Docker environment

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

   i. Study Dockerfile for an image; tutorial to create simple Dockerfile (kerberos, mongo, redis, postgres, nodejs)

   ii. Research ways to enable data persistence: whether in the Dockerfile to create the image, in the docker-compose file once in the container, etc.

   iii. Research where docker volumes saves data: in the container, on the host, both

   iv. Can instances of data persistence be accessed and seen: on the host, in the container (actual data changed to be seen)

   v. Attempt aws using Casey's baseline and with Casey's credentials then use the same baseline with Katie's credentials for AWS access (Katie)

Status #6 To Do List - 3/6/2023

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

   i. Short demo conducted on creating a Dockerfile + building the image (nginx) to host a web page

   ii. Data persistence researched for options (volumes, bind mounts, tmpfs mounts) + where to implement this

   iii. Website with information on how to access volumes data found, not implemented

iv. Casey's baseline and aws credentials work great, never got Katie's to work so she just completed all the remaining docker activities in Casey's baseline.

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

i. Full demo from Dockerfile (more complex), build image, run with docker image/up with docker-compose, access utilization.

ii. Attempt Dockerfile FROM line locally.

iii. Each of us will work on a different image.

Status #7 To Do List - 3/20/2023

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

i. Casey has completed redis simple Dockerfile, image build, ran with docker-compose + attempted FROM line locally

ii. Katie has completed nginx simple Dockerfile, image build, and ran with docker-compose as well as attempted FROM line locally.

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

i. Focus on creating a more complex Dockerfile from an existing image or from scratch + demo understanding

ii. Discuss + brainstorm overall scope of Capstone project

Status #8 To Do List - 3/27/2023

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

i. Katie and Casey were able to add more working commands to a Dockerfile that created directories and replaced some files that were already loaded into the nginx system.

ii. Limited real complexity was added to the Dockerfile

iii. Progress was made in the docker-compose files by controlling ports and assigning IPs and subnets which is aligned with the overall goal of project.

iv. End goal was discussed and is thought (as of now) to be re-writing some labs in classes (IT215, 333) to have them run from Docker and not from VMs. This would also require a whole lab on the basics, in-outs of docker before any students are able to use it for other things. This would need to be given before any rewrote lab (IT101, 215 (beginning)).

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

i. Experiment with networks section in docker-compose: must it be version 2, try many services all with IPs, try more than one network, more than one subnet.

ii. Experiment with "docker network inspect *" commands: creating new networks, changing variables in networks, capture on wireshark from virtual interface on host that aligns with network.

iii. Capture packets on Wireshark virtual interface on the host: see what has to be done to capture packets, see the IPs, ports, other identifiable information from the network.

iv. Work together on the above items.

Status #9 To Do List - 04/05/2023

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

i. We experimented with networks; had two subnets, two different networks (by name), set an IP to each, set more than one IP to one container from different subnets

ii. We used docker network inspect * to view the networks created (two subnets)

iii. We captured packets on the virtual interfaces on the host and were able to ping from the inside of each container

iv. All was done together to increase productivity

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

i. Continue current setup with subnets, networks, IPs - introduce routes, iptables, treat middle container as a router

ii. Start on proposal - preliminary draft of proposal, problem statement, proposed approach towards solution, certain number objectives, deliverables, from which IT courses; separate section to include approach we took thus far

Status #10 To Do List - 04/12/2023

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

i. The current subnet, network, IP setup was expanded, no solid solution found to turn the second container into a router

ii. Were able to find a way to map different subnets to each other and have connectivity

iii. We used a number of sites to spark our findings: [9], [10], [11]. We also utilized ChatGPT to come up with options to try.

iv. Wrote a beginning proposal including certain sections

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

i. Continue to try iptables and IP forwarding in and outside of containers; find a solid solution to having second container be a router

ii. Expand on proposal; expand problem statement, include IT460, focus less on full removal of VMs and more so on the integration on Docker, expand on Docker intro lab (tools, networking, what would be necessary), 5 total labs instead of 3, integrate rough timeline and milestones (2.5/3 weeks per), start to think about certain gains we'd get through docker and which details we get through vms that we'd lose (from scratch sudo apt) but does speed outweight?, people learning to create docker images replicates creating VM?

Status #11 To Do List - 04/19/2023

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

   i. Extended the proposal draft (turning in what we had done at time of meeting, improvements will be turned in next week)
   ii. Continued attempt towards creating routing container

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

   i. Finish routing example with own environment + files, write instructions (finished experimentation)
   ii. Reevaluate conversion/creation of labs - decide direction
   iii. Rework timeline
   iv. Continue/finish draft of proposal; include decided direction of labs, purpose for decision
   v. Create project name

Status #12 To Do List - 04/26/2023

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

   i. Completed Draft 1 of Project Proposal + decided direction
   ii. Successfully implemented static ip rules in own environment (seen in atts/4.23.2023-Casey and atts/4.23.2023-Katie)
   iii. Created project name

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

   i. Fill out + edit intro.tex
   ii. Fill out + edit obj_del.tex
   iii. Fill out rel_works.tex
   iv. Remove constraining statements
   v. Research cloud computing
   vi. Research OpenStack more

Status #13 To Do List - 05/04/2023

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

i. Wrote our introduction (room for improvement)

ii. Expended on/cleaned up our overall proposal (obj and del)

iii. Wrote the related works section (can expand as well - were a little unsure how to go about)

iv. Wrote the research journey section (maybe is too much like a story instead of more factual based?)

Appendix B - Weekly Write Ups

Status #1 - Write Up 1/30/2023

Kubernetes

- Standard for deploying + managing software in the cloud
- A single node is one machine in your cluster - physical machine in data center or virtual machine hosted on cloud
- Kubernetes cluster made up of many nodes
- Persistent volume is same as external hard drive providing a file system that can be mounted to the cluster without being associated to any one node
- 82% of enterprises have a hybrid cloud computing strategy

Kubernetes + public/private

- Data centers are expensive to maintain
- Public clouds are more expensive than people think and the bills make it hard to understand what you are actually paying for - therefore how can you mitigate the cost
- Kubernetes is used to manage/organize systems, operations, applications, files, CPU, RAM, etc. not used to startup/host a cloud environment itself. Either public or private.

Main private cloud providers

- AWS - Amazon Virtual Private Cloud
- Cisco - Cisco Quickstart Private Cloud
- Dell
- Google
- HPE - Hewlett Packard Enterprise (IaaS) Greenlake
- IBM - can connect to existing infrastructure
- Microsoft - uses Azure Stack

- OpenStack
- SAP - enterprise planning software, S/4HANA Cloud (private)
- VMware (SaaS)

Status #2 - Write Up 2/6/2023

Youtube Video Notes

- Container - way to package applications with all dependencies and current configuration
- Entire container can run on its own, does not need to remain in its location or OS
- Easily moved and shared
- Very efficient
- Container storage stay in a repository (companies private) (public for docker, DockerHub) [12] hundreds thousands docker images stored in this repository
- Each main application has an official docker container image, search bar for something else too, non official container images from developers or jenkins (creator)
- This site is a great reference to get started with, much less scary to have things to start with instead of from scratch.
- To run applications, (before) must install processes straight onto OS (installation process different per OS)
- Multiple steps = many places for something to go wrong
- Complex application = more tedious work, how many things to download on each OS
- (After) with containers, do not have to install anything directly to OS, container acts as it's own OS (or OS layer)
- Already have all parts needed for execution, even most complex, everything put together already, also could already exist in repository (just one docker command to fetch and start it same time)
- Same docker command for every OS***
- Easier and more efficient

Status #3 - Write Up 2/13/2023

- Before - everything separate, harder to install + make work
- Now - everything already bunched together in one environment + no need to configure, one command to pull docker image
- Beginning install
- If you have downloaded an image before, then want to download newer version, only the layers with differences will download
- Docker image - actual package, with configuration and dependencies, movable
- Container - pull and start image on local machine, creating container environment.
- Want to see "database system ready for connections"
- Docker images much smaller than VMs, they don't include OS Kernel
- Docker containers can run much faster
- Can run VM host OS on any host OS
- Cannot run linux based docker image on windows, can run linux VM on windows
- See if kernel is compatible with docker image
- Installation differs on OS and OS version
- Container is running environment of image
- Host can run multiple containers at once
- Laptop has ports available for certain apps, make binding from port to container, port 5000, you bind laptop port 5000 to containers port 5000, cannot connect another container to 5000 on host, but can listen on 5000 w same port number from containers

Status #4 - Write Up 2/20/2023

Created a Detailed Step-by-Step Instruction Page

- We both created our own Docker environment fitted to connect to a node server, a mongo-express GUI, to make a Docker image, to connect to a private repository, and more.

- We presented this display to Professor Salib and demonstrated full understanding of the topic.
- We were given some tasks to accomplish for next week including adding data persistence to our Docker environment
- We will also be analyzing the default mongo and mongo-express images from the public Docker Repository

Status #5 - Write Up 2/27/2023

Research about Docker Images & Containers

What is in a docker container?

- A docker container contains everything, start to finish, needed to run an application
- This includes:
  (a) Code
  (b) Runtime
  (c) System tools
  (d) System libraries
  (e) Settings
- When they are pulled, they are considered images, when they are ran, they are considered containers
- Containers isolate software from the host environment The containers are standard for all OS', they are the same inside, they run the same, and the commands to pull, run, use them are the same.
- Containers are meant to virtualize the OS hardware to be portable and efficient
- You can run many containers on one host environment while VMs include their own new OS
- Running containers inside a VM allow for the greatest flexibility
- Using docker-compose allows us to run multiple different containers with just one command and the containers are fully configured inside the file.
- The container has its own file system that it puts on the host and we are able to access the stored data

- As default, everytime a container is stopped and restarted, all data is gone
- To combat this, we included volumes in the docker-compose file and the data remains between stops and starts.

Final Steps in Docker Demo Instructions

- To finish off the Docker demo, we added a volumes section to our docker-compose (mongo.yaml) file to implement data persistence
- After this section was added, a change was made the in the HTML file, an ID was added to mongo-express, and the containers were turned on and off
- Usually, the ID would be gone after the container was restarted like this, but this time it was not and it remained

Status #6 - Write Up 3/6/2023

Understanding Docker images, volumes

Creating a Dockerfile

- Entering vscode to an empty text file named ”Dockerfile”
- Will list full code below with descriptions
  FROM nginx:1.10.1-alpine
  COPY . /usr/share/nginx/html
- Simply, the nginx:1.10.1:alpine is used
- Using the copy command allows us to add our own files to the container in order to make changes to the function
- File path is specific - every image has one
- Create an index.html file
  ¡html¿ ¡body¿ ¡h1¿Hello!¡/h1¿ ¡img src=”wave.png”¿ ¡h1¿I hope this works!¡/h1¿ ¡img src=”whale.png”¿ ¡/body¿ ¡/html¿
- Have whale and wave downloaded to the folder
- Open in browser with vscode tools
- Build image with docker build -t hello-casey .

- Run docker images to see the created image from this dockerfile
- Using the ID from above command, docker run -d -p 80:80 (ID)
- Enter in browser localhost:80 to see the page
- Show other edits made
- [13]

Data Persistence

- Volumes are stored in a part of the host filesystem that is managed by Docker
- Can use ”docker volume create” or add volumes to the Dockerfile at creation
- ”When you create a volume, it is stored within a directory on the Docker host. When you mount the volume into a container, this directory is what is mounted into the container.”
- ”This is similar to the way that bind mounts work, except that volumes are managed by Docker and are isolated from the core functionality of the host machine.” 1

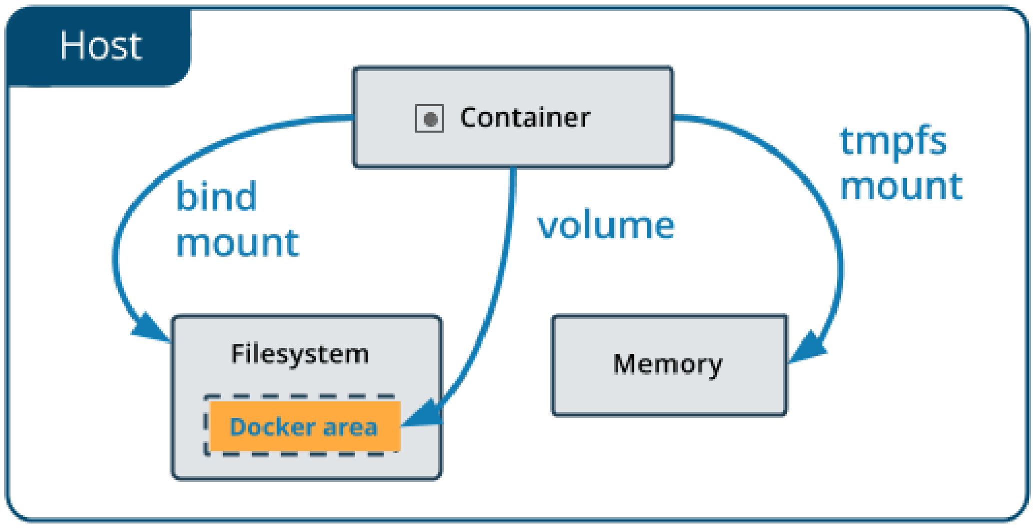

Figure 1: Data Persistence Options

- Bind mounts are the lesser counterpart to volumes

- Volumes are almost always best practice, but it is personal preference
- Websites from Docker docs [14] [15]
- Looks best to me to have volumes in docker-compose file

Accessing Volumes

- [16]

Status #7 - Write Up 3/20/2023

Dockerfile Image, Build, Run: Local

- We created a Dockerfile for the images nginx and redis.
- We added basic commands such as FROM, COPY, and RUN.
- We also created docker-compose files to run containers from our images that were built.
- We made second Dockerfiles pulling from our local image repository.
- Instructions:
  (a) Make sure Docker is up by running "docker ps"
  (b) Using VSCode, start creating a Dockerfile by creating a regular text file and naming it "Dockerfile"
  (c) Select an image to pull from, in my case, redis:latest, in Katie's case, nginx:latest
  (d) Create the first Dockerfile; seen in /atts/Redis+Nginx/DockerFiles-Casey as Dockerfile (1)
  (e) Turn the Dockerfile code into an image using a build command: "docker build -t hello_casey:latest .".
  (f) Check this image was created using "docker images"
  (g) Start creating a docker-compose file (.yaml ext); seen in /atts/Redis+Nginx/DockerFiles-Casey as redis.yaml
  (h) Bring this image up using the docker compose file (image to container) with "docker-compose -f redis.yaml up -d"
  (i) Create another Dockerfile with it's FROM line being the name of the image you created with the first Dockerfile; seen in /atts/Redis+Nginx/DockerFiles-Casey as Dockerfile (2)

(j) Now you have created your own Dockerfiles and used your own image to pull from local repository

Status #8 - Write Up 3/27/2023

Dockerfile Alterations

(a) RUN commands: This command with execute any command on the pre-existing image and commit the changes in a new image we create using the Dockerfile.

(b) COPY command: This command allows us to copy files from our own host directory into the docker image.

(c) WORKDIR command: This command "sets a working directory for other Dockerfile instructions, such as RUN, CMD, and also the working directory for running instances of the container image" [17]. This command is suggested to be used in a Docker file over "RUN cd".

(d) FROM command: This pulls the base image you will be manipulating and changing in the dockerfile.

(e) We used these commands to add files, create directories and work with in directories without yet creating the docker image.

Docker-compose (complex) Instructions

Alpine - Casey

- Begin by going to Docker Hub and finding the Alpine Docker Official Image to read about it
- I discovered that Alpine is a lighter and smaller Linux distribution
- I began creating the Dockerfile with a "FROM alpine" line and then planned to add many RUN lines containing updates, upgrades, and many tools such as wireshark, openssh-server, net-tools, vim, and other things which can be seen in atts/CaseyDockerfileComplex-3.29/Dockerfile
- The Dockerfile was built into an image using the command "docker build -t casey-1 ."
- I built the image, and named it casey-1 as an image in my local repository.

- I was contemplating whether to continue advancing the Dockerfile or to attempt changes in the docker-compose file instead.
- The docker-compose file can be seen in the folder as well.
- In this particular environment, something must have been configured wrong, because after the ”docker-compose up” command that brings up the images configured inside, only one of the two would run. The one that would not was the one I had created, letting me know I had a problem
- To create the compose file, I used the image I created as an image for one service and decided to use mysql as an image in another service.
- In this file, I configured the ports and the networks. Here I could also add data volumes to have data persistence.
- We used the default bridge as our driver which is the link layer that handles forwarding traffic between the IPs in a network
- ”ipam” stands for IP Address Manager/Management and marks where we begin to configure the IPs/subnets for the network
- This was the first time we had touched on IP Addresses, subnets, and gateways within our Docker containers.
- After configured, I entered the shell inside the container. I did this with ”docker exec -it (container ID) /bin/sh” and was returned with ”#”.
- I checked the container ID with ”docker ps”
- The only container running was the mysql official docker image container that was in my compose file, so the casey-1 image was not there running to go into
- To check the IP address of each container, I entered ”docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' (container ID)”. I was returned the addresses that I assigned for that container on each. This long command is to be used as is (no slashes) and to be used on the host, not in the shell of a container. Again, the container ID is found with ”docker ps”
- After all of this, back on our host, enter docker network ls. This will return the names and IDs of each network present. At least one should be the network that has been created in the compose file
- Another command that displayed a lot of information was ”docker network inspect (network ID)”. This has to be the network ID that

aligns with the virtual interface on the host. This showed me all of the information about the entire network that contains all of the services that are up with docker-compose.

- Katie's section below details the troubleshooting we did with the IP address networks section of docker-compose.

Nginx - Katie

- Begin by going to Docker Hub and finding the Nginx Docker Official Image
- I created a Dockerfile and pull form the nginx image
- I looked at a lab 3 from 460 and added RUN commands with installs of the application that we used in that lab see atts/KatieDockerfile Complex-3.29/Dockerfile.
- I then wanted to test the ability to change the ip address of the docker container on the inside using Docker compose.
- I created a file named "docker-compose.yaml" and used the yaml code I found from [18]; see atts/ KatieDockerfileComplex-3.29/docker-compose.yaml. This will be the docker-compose file to bring my images up into running containers
- Container ID can be found with "docker ps"
- After running this yaml code using the command "docker-compose -f docker-compose.yaml up" which started a mysql conatiner, we used the command "docker inspect -f '{{range .NetworkSettings.Networks}} {{.IPAddress}} {{end}}' (container ID)' " to see the ip address assigned to the docker container. The ip address assigned to the were not the ip addresses we meant to assign to the container it was the next ip after the gateway of the ip.
- We went back to the website and found that the way to assign specific ip's to the individual containers is to add the code:
  networks:
  vpcbr:
  ipv4_address: 10.20.0.5
  after we declare which port we want to use for each separate container we wish to assign a specific IP too.
- We then stopped the container using the command "docker-compose -f docker-compose.yaml down" and restarted the container using the

command "docker-compose -f docker-compose.yaml up -d" and re-examined the ip given to the inside of the docker container using the command "docker inspect -f '{{range .NetworkSettings.Networks}} {{.IPAddress}} {{end}}' container ID' " again. This time we were given the specific ip of 10.20.0.5 that we assigned to this container.

- Now that we have tried this with a standard docker image we now wanted to try changing ip addresses of the containers built from our Dockerfiles.
- We start by building an image from the Dockerfile called attempt6
- Then we create a new Docker compose file called "docker-compose.yaml" we used the same code that was successful for running mysql container but changed the service we are using to nginx and the image we are using to create the container that will be named attempt6 see atts/KatieDockerfileComplex-3.29/composeNginx/docker-compose.yaml.
- We start the container using the command "docker-compose -f docker-compose.yaml up" which starts the container attempt6. We used the command "docker inspect -f ' {{range .NetworkSettings.Networks}} {{.IPAddress}} {{end}}' container ID' " on the host to see the ip address that we assigned to the docker container in the compose file and are given the ip address of 10.20.0.5.

Status #9 - Write Up 4/5/2023

Docker network Testing Instructions

Dockerfile + Docker-compose set-up for current IP, subnets: Casey

- I will be describing the steps I took to create two networks in my docker environment
- I will be using VSCode and my Ubuntu terminal
- I went to VSCode and created a folder to contain all of my files, named prac, as well as three folders inside, named image1, image2, and image3
- In each image folder, I created a new file and named it Dockerfile (in all 3, this is why I created the folders)
- I added a FROM line in each Dockerfile, each with a different Docker Official Image, the image itself is not really the point here

- In each Dockerfile, I tried to add a few more lines just to make it differ from the official image somewhat
- Each Dockerfile can be seen within the image* folders in the 4.5.23-Casey folder in the atts folder.
- I will now be building each image in my docker environment; "docker build -t image* ." is the command to do this (cd into image1 folder, *=1; cd into image2 folder, *=2, etc.)
- Now we have three built images in our own local repository
- I will now begin to build my docker-compose file and will be putting it straight in the main prac folder, not inside an image folder
- I decided to name it docker.yaml (giving purple ! instead of pink whale) and began with version: '2'
- The file can be found in the folder mentioned before in atts, but to explain
  (a) services: - lists all the images that I want to turn into containers
  (b) each container section has an image, restart, ports, and networks; could also add volumes just was not necessary here
  (c) the image - what to pull from, must be in local repo, or in docker's repo
  (d) the restart: always tag - sets to always restart the container if it stops
  (e) ports - maps port from the host to a port inside the container with a : in between
  (f) networks - newest feature we've added and made the most progress with; defines which network to be a part of and assigns an IP address from this networks subnet
  (g) as seen in image2, can have an IP address from each network/subnet defined in the file
  (h) more importantly - the actual network definition section - lists, in this order, network name:, drive:bridge, ipam: (this is IP address management), config:, - subnet: {list desired IP subnet with /nn}, (optional) ip_range: {list range of IPs} (see example in file), gateway: {list desired IP gateway within subnet}
  (i) I did this twice in order to set two networks
- Above should have deeply explained the lines in the files so the user knows what to keep and what they can change

- Now that you have images and a docker-compose file, we need to bring the compose file up and start these images into containers
- To start the compose file, I went to the prac directory and entered "docker-compose -f docker.yaml (name of my compose file) up -d"
- This not only starts all three images into containers but also starts the two networks that I configured
- I can first enter "docker network ls" and see a short list of networks in my environment, likely the bottom two will be the ones just created in the compose file
- If you take the ID of one of these new networks and enter "docker inspect {network ID}, you will see the config of the whole network, including its subnet, gateway, and containers that are assigned IPs in this subnet, also the config of these mentioned containers and their IPs, MAC addresses, IDs, and more
- Enter docker ps to see the list of running containers and their IDs
- Now that we have seen the network specs, we enter this command to make sure the ip address assigned has actually applied to the container at run-time: "docker inspect -f '{{range .NetworkSettings.Networks}} {{.IPAddress}} {{end}}' container ID (we found this in command before) (this command will need to be entered without the slashes that are added here)
- This should return the ip address that was assigned under the network section on each container/service
- When done on the container ID for image2, two IP addresses should appear, one from each subnet
- One last thing to try to see the networkings; start wireshark with sudo wireshark in command line
- Select the interfaces that have the format "br-(ID of network)"; select both
- Enter the interactive terminal of one or all of the containers; "docker exec -it (container ID in question) /bin/sh" - this should return a # to you if you are in the shell now
- Here, type ping (set IP address of the other container in this network)
- Likely, ping will not have been installed yet, so enter "apt-get update" and "apt-get install iputils-ping"

- Re-enter ping command and see connection through the network from the inside of docker containers
- Check wireshark to see ICMP packets and likely some other packets that have been captured unintentionally

Dockerfile + Docker-compose set-up for current IP, subnets: Katie

- I am going to build docker images that have two network interfaces that are from separate Ip subnets.
- First I created a folder called Tester that containers a Dockfile that will build an image from nginx image, that we will use to create the image server1 and a docker-compose.yaml file. Inside of that folder is another folder named next that container a Dockerfile that pulls from the image called server1 to create a new image called server2.
- I teh created a docker compose file that gave each of the images that I would create (nginx, server1, and server2) an static ip address from each subnet that I created.
- This can be seen in 4.5.2023Katie folder in the docker compose file.
- Now was time to create the docker images I started by "cd Tester" then the command "docker build -t server1 ." once that was built I executed the command "cd next" while in next I executed the command "docker build -t server2 ." to build the server2 image off of the server1 image I made.
- Now i go back to the Tester file and execute the command "docker compose -f docker-compose.yaml up -d" to build the containers from my images and pull the nginx image from docker that I will also be using.
- I then checked that each conatiner I started had the 2 static IP adresses I assign them by executing the command "docker inspect -f '{{range .NetworkSettings.Networks}} {{.IPAddress}} {{end}}' Containers number' " in each container.
- Once each has established it has an ip address I go into each of the containers using the command "docker exec -it 'container number' ./bin/bash" sense there are three docker containers I opened each on a separate command terminal window.

- On the host I opened a wireshark and started capturing on all the interfaces that my two docker interfaces were running on.
- Then back on the nginx container terminal I start pining server1 and sever2's containers ip addresses. I do the same on server1 and ping server2 and nginx containers. These should all be successfully as seen in 4.5.2023Katie folder file Server1-Redis-Images-ping-eachother-on-both-Subnets.pcapng with these successful pings.

Status #10 - Write Up 4/12/2023

Draft For Capstone Write-up; Very Basic Report

Problem statement: We are attempting to combat the fact that VMs are used solely in the networking aspect of the major without integration of the other options that exist.

We are attempting to introduce Docker (and possibly Kubernetes) in IT215/IT333/IT460 in order to expand the knowledge and experience of the future IT students. This would include storing/pushing created Docker environments into private AWS cloud repositories for mobile usage and some understanding of cloud computing. This would remove the usage of VMs for the labs we choose to convert, and can (hopefully) have all the same functionality within the Docker containers.

Proposed approach towards solution: Our proposed approach towards a solution to this problem is to teach ourselves the ins and outs of Docker in preparation while beginning to think about the integration of Kubernetes. From the experience we have gained, we plan to convert x* number of labs to use only Docker containers, networks, etc. instead of multiple VMs like is done in many labs. We believe an intro to Docker lab would be necessary/beneficial for them to go onto dive deep into Docker and its functions. Some other solution ideas can include a mixture of VMs with Docker, a mixture of Docker with Kubernetes, or docker with GNS3 and the inclusion of hardware as well.

Certain number of objectives: A x* number of 215, 333 and 460 labs that we would fully convert into using Docker. Which would include a lab to teach docker from a basic form to help students gain understanding of Docker and how it works.

Deliverables: Converted x amount of 215, 333, and 460 labs to be fully functional using Docker (and possibly Kubernetes) instead of VMs. As well as the lab we would create to teach how to use Docker (and possibly Kubernetes).

** At the moment, we are thinking of 5 total converted 215 and 333 labs, and 1 fully written by us, Docker introduction lab.

Status #11 - Write Up 4/19/2023

Continued Draft For Capstone Write-up

Problem statement:
We are attempting to combat the fact that VMs are used solely in the networking aspect of the major without integration of the other options that exist. There is much value to be gained from not only introducing Docker, but also integrating Docker and VMs into a working environment together. We are interested in experimenting with the possible integration of Kubernetes to increase the overall functionality of the environment as well. Introducing a topic such as Docker in a lower level class is most effective as students are still open to brand new methods. If new IT students are introduced to Docker before they are fully stuck on VMs (IT101, IT215), then the students would likely be more receptive to the new concepts of networking and be able to adapt more quickly. These adaptation skills will be important for their futures, as it is likely that brand new things will be introduced to them often and even in the following IT classes they will take.

We are attempting to introduce Docker (and possibly Kubernetes) in IT215/IT333/IT460 in order to expand the knowledge and experience of the future IT students. This would include the usage of Docker (with or without VMs) for the labs we choose to convert, and likely will have similar functionality with the addition of what it would take to use the containers.

Proposed approach towards solution:
Our proposed approach towards a solution to this problem is to teach ourselves the ins and outs of Docker in preparation while beginning to think about the integration of Kubernetes. From the experience we have gained, we plan to convert x* number of labs to use only Docker containers, networks, etc. instead of multiple VMs like is done in many labs. We believe an intro to Docker lab would be necessary/beneficial for them to go onto dive deep into Docker and its functions. Some other solution ideas can include a mixture of VMs with Docker, a mixture of Docker with Kubernetes, or docker with GNS3 and the inclusion of hardware as well.

The lab that we would create start to finish would include general and introductory tools to Docker. This (roughly) could include adding yourself as a user, how to pull from docker hub, checking your images, running images, creating your own images, starting them, starting them from docker-compose, alterations in docker-compose, pushing to AWS repository, pulling from AWS repository. These topics can and likely will change/expand. As we go about transferring the 4 other labs, we will make sure what is done in the intro lab prepares them for what is needed in the rest of the Docker labs. This would include storing/pushing created Docker environments into private AWS cloud repositories for mobile usage and some understanding of cloud computing.

Certain number of objectives:
A x* number of 215, 333 and 460 labs that we would fully convert into using Docker. Which would include a lab to teach docker from a basic form to help students gain understanding of Docker and how it works.

Deliverables:
Converted x amount of 215, 333, and 460 labs to be fully functional using Docker (and possibly Kubernetes) instead of VMs. As well as the lab we would create to teach how to use Docker (and possibly Kubernetes).

** At the moment, we are thinking of 4 total converted 215, 333, 460 labs, and 1 fully written by us, Docker introduction lab.

Status #12 - Write Up 4/26/2023

Draft 1 Proposal

Opportunity statement

We are attempting to combat the fact that VMs are used solely in the networking aspect of the major without integration of the other options that exist. There is much value to be gained from not only introducing Docker, but also integrating Docker and VMs into a working environment together. We are interested in experimenting with the possible integration of Kubernetes and AWS to increase the overall functionality of the environment as well. The integration of AWS provides an aspect of cloud computing that would be beneficial to know.

AWS and OpenStack are two cloud services that we are looking into to integrate into the creation of our labs [5]. AWS is a public and private cloud computing service that allows the user to setup an account and start using their different cloud services such as ECR and Kubernetes. The benefit of this includes being able to push Docker containers to the cloud service via the Elastic Container Registry (ECR) as well as use the cloud in different locations. If we only want to use the public cloud service of AWS, we will be using the ECR mainly because of its ability to store containers for use [6]. If we want to experiment creating and configuring our own private cloud, then we would do so with the guidance and permission of Dr. Salib in hopes of utilizing this cloud for the whole class, or even whole department. We decided on using AWS over OpenStack because AWS is more secure through Amazon's infrastructure [7].

We believe it would be efficient to use the cloud in conjunction with our Docker environment because the containers would be able to be pushed into the ECR as well as pulled back into the Docker environment. This can happen from any computer and any location which is what makes this efficient as mobility is important. The files that create Docker images would be stored with the student and therefore not needed to be pushed anywhere. Docker-compose files are able to bring up these containers after they are pulled from the ECR as well.

We have decided do use Docker as our container system over LXD because we have spent a significant amount of time using, learning, and experimenting with Docker at this point. After researching LXD [8], we found that Docker is more portable, can be integrated with tools like Kubernetes and almost all public clouds while LXD cannot, as well as that Docker is more widely known and used as the industry-leader. We have grown to feel very comfortable using Docker that it would not be advisable for us to change paths now [5].

Introducing a topic such as Docker in a lower level class is most effective as students are still open to brand new methods. If new IT students are introduced to Docker before they get to the real VM heavy courses such as IT460 and IT333, then the students would likely be more receptive to the new concepts of networking and be able to adapt more quickly. These adaptation skills will be important for their futures, as it is likely that brand new things will be introduced to them often and even in the following IT classes they will face. We are planning to introduce Docker in a possible new course that focuses on cloud computing using Docker (and possibly Kubernetes), in order to expand the knowledge and experience of the future IT students. With this new class, we'd also be enabling the instructor to introduce to undergraduate students curriculum that includes hands-on learning and exercises focused on topics that may not have been possible if the current solely VM approach had remained.

Proposed Solution

Our proposed approach towards a solution to this problem is to teach ourselves the ins and outs of Docker. From the experience we have gained thus far and will continue to grow upon, we plan to create educational labs to use Docker containers, networks, images, etc. instead of multiple VMs like is done in many labs currently. We were planning on taking 3-5 labs that are used in IT major classes currently and convert them to be Docker based instead of virtual machine based. We have strayed from this idea because we felt that having to work off of pre-established topics would box us in too much. It would reduce the full understanding of both VMs and Docker as it would limit the capabilities of both. We felt that during the transition a lab would lose too much of its value and of its material that would make it less beneficial to the student. We want the Docker based labs to focused around topics that it works well with and was made for, not just topics we have done in VMs before. Since we will be creating new labs from scratch, we will not be losing function from the lab as we convert it from VMs to Docker. Instead, the student will now be able to go into just as deep of function in both the VM based learning and Docker ones as well.

We believe creating an Introduction to Docker lab would be necessary/beneficial for the students to go on to dive deep into Docker and its functions. Some other solution ideas can include a mixture of VMs with Docker, a mixture of Docker with Kubernetes, or docker with GNS3 and the inclusion of hardware as well.

The first lab that we would create start to finish would include general and introductory tools to Docker. This (roughly) could include adding yourself as a user, how to pull from docker hub, checking your images, running images, creating your own images, starting them, starting them from docker-compose, alterations in docker-compose, pushing to AWS repository, pulling from AWS repository. These topics can and likely will change/expand.

Another idea we could expand on is creating Docker images that enable certain networking tasks and deploy them to a public Github, allowing other curious students, teachers, and professors to experiment using these images or teach using them in actual implementation.

For the labs we will create after the intro, we have a wide range of topics we can choose from and experiment with. The topics are not chosen yet, but we plan for them to be selected from researching other projects that have been done, along with the help of Dr. Salib for topic creation. We can say for sure that we want to include basic networking tasks within these labs such as setting and checking IP addresses/MAC addresses, monitoring traffic within containers + on the created interfaces for containers on the host, checking logs of built containers as they start + as they are altered, altering networks within docker-compose, adding, changing iptables rules/static ip routes, etc.

Project Goal & Objectives

The project goal is to create brand new labs with the intention of them being taught in a new IT course focusing on Docker and Kubernetes containers and container orchestration.

To achieve this goal, we set out to accomplish the following objectives:

- Teach ourselves Kubernetes functions in order to plan to supplement labs with the integration of it
- Implement a introductory lab to Docker processes and Docker commands
- Provide teachings to create and use private AWS repository
- Create labs to expand on knowledge of Docker processes and go deeper into available functionality

We plan to set specific deadlines and goals for ourselves in order to stay on track and monitor our progress. We will closely follow the calendar that we have created (still to be edited) and have planned deliverables per week. Setting smaller goals on the path to the final goal will help us make sure that we are consistently making progress.

Deliverables

We plan to submit educational labs regarding Docker; each with Exercises, Steps, Tasks, and Questions to help students better grasp the topics. We have not yet decided the number of labs this will be as we are unsure the scope or length it will require per lab. We will also be submitting a final report that acts as a summary of both our design and implementation sections. More informally, we will be submitting what we set for ourselves each week.

Status #13 - Write Up 5/4/2023

Draft 2 Proposal

This week, our work can be seen on the intro, objectives & deliverables, related works, and research journey (so far) pages.