Ryan Morlando
Dominic Rey
Deliverable 3 Technical Document

We have made considerable progress since deliverable two. We now have the containers automated so that all you have to do is deploy the docker branch of the cluster template, $ git clone https://github.com/brolando94/496-group-project, move into the “dockercompose” folder and then run $ docker-compose up command. From then on the files in the “dockercompose” folder automatically build the web server and database. The main file is the docker-compose.yml file which sets up our two containers to our specific needs. The yml file has the services laid out and then has the network initiated that both containers will communicate on.

The first service is the db. This consists of the mariadbtest container. It runs off the latest version of MariaDB, is on the projectNetwork and exposes port 3306. This container also accesses an environment variable file that connects it to the MariaDB, logging in as the root user and going into the test database.

The second service is defined as web. This is where the webserver container is. It runs off the latest version of ubuntu, exposes port 80 and is also on the projectNetwork. This service is dependent on the db that was defined in the previous paragraph. That means the webserver will not run unless the database is up and running. This service also utilizes a separate docker file that helps build the custom image. This separate docker file is in the “webapp” folder. This file allows us to run the commands that we would normally have to type into the command line to get the correct service installed on the image. Named “Dockerfile”, it runs the commands to update ubuntu, install apache2, and starts the apache2 service. Last it exposes port 80 for communication between containers.

This is what we consider to be the substantial progress that was made in the last week. At the time of deliverable two, we were still creating our containers in the command line were not able to get them to connect to each other. We knew we needed the network setup but had not made it to the port exposure. Now we have all of that

automated and have docker acknowledging that the mariabdtest container is attaching to the webserver container. After that is where the project halted, and we ran out of time to figure out the problem. The MariaDB runs through its initialization after the two containers connect but gets stuck in an infinite loop with the output “webserver exited with code 0”. This was frustrating because we feel like we are so close to the finish line and completing the goals.

In the “webapp” folder there is also an index.html file that we created to have on the webserver to communicate to the database. It is really simple html/php and just sets up a table with a couple entries and then sends a select all query and shows the table contents on the web page. Though it is a simple concept, to get there would have been a big win. We think the problem we are having with not being able to view the website is how we have the webServer container dependent on the mariadbtest container. That leads into the bigger problem of something going wrong in the setup of the maraidbtest container. One theory that we want to try out but have run out of time is that the database might be just sitting there waiting for commands and our index.html file is in the wrong location. We know from previous setups that apache has a test html file that they put up so that you can see if you server is live. we tried to put our html file in the spot preemptively so that when you download the project folder from github it is right there ready to go. But if the html file we created is in the wrong spot. Then the php is not executing and the table that is created from it does not exist. That is the last key to finishing this puzzle. Though we didn’t fully make it to the end, we still learned a lot and were able to automate what we had accomplished from deliverable 2 and more.

So just to recap a little, we fell short of the final deliverable in the end but learned a lot in the mistakes all along the projects path that really hung us up. The biggest thing that we learned was starting from the bottom up. If you can not walk. You probably are not going to be able to run. You really need to fully understand docker on a command line basis before you dive into the automation of yml files. Even after you feel like you mastered the command line. The yml file still throws you for a loop. There was very little indication of what the error might be. Just that the error came from before or at the spot

it was reported at by docker. We spent a lot of time looking for solutions to various little problems that mostly came down to formatting. Yml files need to be carefully written and the rules closely followed, or you could be spending a lot of time fixing the little mistakes. When it came to working on this project our main access to docker was through cloud lab. So, to test changes to files we either had to reload the node to wipe the cloned git repository or change the files in the nano editor. Making the changes on GitHub were faster but took longer to get to the test phase. Changing the files in the nano editor took a good bit longer than it would on GitHub but could be tested right away. We are still not sure which way is faster, but both got tested out a lot. That would have not been such an issue if we had the knowledge that we do now of how to set up the containers and all the picky yml syntax rules. The endgame was to have a fully automated cloud lab profile that could be deployed. Unfortunately, we never even stepped out of the original cluster-template and just focused on pulling from our git repository and using the docker-compose up command as the start of our automation.

Overall this project was more challenging than we originally thought but we are happy with the progress that we have made since deliverable two and overall on the project. From a time saving standpoint, what took us about 15 mins to set up via the command line now takes around 3 with the various files that are in the "docker-compose" folder. We are so close and have made a lot of progress. With another week or so we could have fulfilled the requirements set out in the project.
List of commands to get to where we are starting with docker and all the necessary repositories downloaded(cluster-template, docker branch):

```
$ git clone https://github.com/brolando94/496-group-project
$ cd 496-group-project/dockercompose
$ docker-compose up
```