Technical Report
5/8/2019
Installation of LAMP Stack Server utilizing Docker Toolbox
Group 4 – Jeffrey Jones, Andre Ibarrondo, Jacob Hansen

Deploying an Automated LAMP Stack Profile onto Cloudlab's Cloud Infrastructure

Authors: Jeffrey D.Jones, Andre Ibarrondo, Jake Hansen

West Chester University, CSC496: Cloud Infrastructures

Technical Report
5/8/2019
Installation of LAMP Stack Server utilizing Docker Toolbox
Group 4 – Jeffrey Jones, Andre Ibarrondo, Jacob Hansen

Introduction

With the goal of building a website with the ability to print out the various professors' at West Chester University's identification number, name, and picture, all obtained from information stored in a backend database, the development of a LAMP Stack Service, deployed onto CloudLab's cloud infrastructure was determined to be the best way forward.
Throughout this report the various steps taken to achieve our goals of developing an automated profile on CloudLab which displays a webpage with information gathered from a database will be outlined. First, we will be summarizing the results and our conclusions made during our development process. Next, we explain the various tools used throughout our experiment, then moving onto the various experimental details which we encountered throughout the development of the LAMP stack and automated profile. Finally, we will conclude, in more detail, about our results and how we can plan for the future to resolve any issues we encountered along the way.

Summary

The results of the initial development process of a LAMP Stack profile on a local machine achieved great results. Utilizing Docker and Docker Compose, the manual development of the LAMP Stack server was successful. Docker Compose allowed for the making of highly configurable dockerfiles which were decoupled and connected back to one another utilizing ports. With the various dockerfiles configured by our docker-compose.yml file, we were able to have a fully deployable LAMP Stack on a local machine. This allowed for the connection to the localhost:8000 on a web browser and signified that the various containers were connected to one another, most importantly the database.

Some issues encountered during the development of the LAMP Stack was providing a way for the containers to listen for one another on a port. The solution to this was creating a virtual host which listened on a designated port, allowing for the connections needed. Another issue encountered was the creation of duplicate entries within the database and our inability to utilize phpMyAdmin to its full extent. Our workaround for this was to manually create Databases, Tables, and insert data through the index.php file to create our database for the website. An issue we foresee in the future with our database architecture is the creation of duplicate entries with every run, along with the possibility that the images being displayed will be lost in time due to them being grabbed by a URL instead of from the database like the names and identification documents are.

When it came to deploying an automated profile, the usage of docker compose as our method for running multiple containers at once proved to be greatly beneficial. With much of the work being done by our docker-compose up function, the automation of our profile was made easy by just using a bash script to direct to our folder containing our project, and indicating to run the command "docker-compose up."

Technical Report
5/8/2019
Installation of LAMP Stack Server utilizing Docker Toolbox
Group 4 – Jeffrey Jones, Andre Ibarrondo, Jacob Hansen

Trouble found during our process was the length of time it takes to instantiate a profile in CloudLab. This proves very time consuming and detrimental when testing profile functionality. Even with CloudLab's webhooking feature to automatically update from a github repository, many hours of development were spent waiting for instantiations to test small lines of code.

Lastly, an abrupt issue occurred at the end of our development stage with CloudLab's infrastructure and our own local machines. An issue with launching a webpage and database was discovered after terminating an expiring instance of our automated profile. Due to time constraints we were unable to solve this issue as to why it was occurring.

Experiment

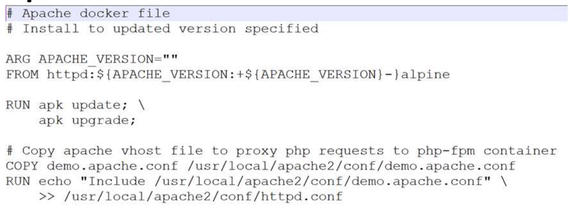
Operating Systems

Windows 10 Home used for local development.
Ubuntu 16.04 used for development within docker and CloudLab's shell service

Docker

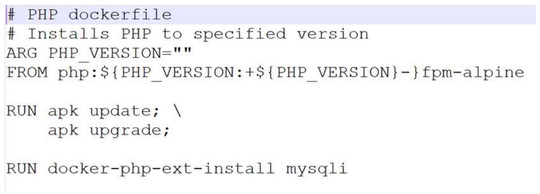
Utilizing the technology of Docker Toolbox for Windows 10 v.18.09.03 allowed for the setting up of LAMP Stack Service onto CloudLab's cloud infrastructure in an organized, containerized, and configurable way. By containerizing images and pushing them to DockerHub, a database with all of our containers and images were available to pull and push at any time, allowing for increased organization and testing functionality. Various Dockerfiles were utilized to install the services required to run a LAMP Stack Service (See figure1).

Apache Service Dockerfile



```
# Apache docker file
# Install to updated version specified

ARG APACHE_VERSION=""
FROM httpd:${APACHE_VERSION:+${APACHE_VERSION}-}alpine

RUN apk update; \
    apk upgrade;

# Copy apache vhost file to proxy php requests to php-fpm container
COPY demo.apache.conf /usr/local/apache2/conf/demo.apache.conf
RUN echo "Include /usr/local/apache2/conf/demo.apache.conf" \
    >> /usr/local/apache2/conf/httpd.conf
```

PHP Service Dockerfile

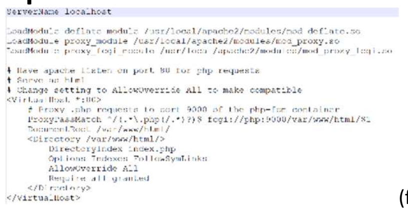


```
# PHP dockerfile
# Installs PHP to specified version
ARG PHP_VERSION=""
FROM php:${PHP_VERSION:+${PHP_VERSION}-}fpm-alpine

RUN apk update; \
    apk upgrade;

RUN docker-php-ext-install mysqli
```

Figure1

Needing the ability to decouple container, while still having them be able to communicate with one another when requested, the scripting of an apache.config file was required (See figure2)

Apache.conf



```
ServerName localhost

LoadModule deflate_module /usr/local/apache2/modules/mod_deflate.so
LoadModule proxy_module /usr/local/apache2/modules/mod_proxy.so
LoadModule proxy_fcgi_module /usr/local/apache2/modules/mod_proxy_fcgi.so

# Have apache listen on port 80 for php requests
# Serve as html
# Change setting to AllowOverride All to make compatible
<VirtualHost *:80>
    # Proxy .php requests to port 9000 of the php-fpm container
    ProxyPassMatch ^/(.*\.php(/.*)?)$ fcgi://php:9000/var/www/html/$1
    DocumentRoot /var/www/html/
    <Directory /var/www/html/>
        DirectoryIndex index.php
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

(figure2)

By looking more in depth into the apache.conf file, we demonstrate our effort to make sure that both the apache and php containers were able to listen to one another through port 80, while remaining completely independent of one another, ie each container is a value of PID 1. Port 80 is specified with VirtualHost *:80 and the settings are changed to AllowOverride All as during our experiment the issue of connectivity was expressed while that setting was not into effect.

With all of these files aiding in the installation and connecting of various containers together through port listening, Docker Compose was utilized to automate and configure the various files into containers and images.

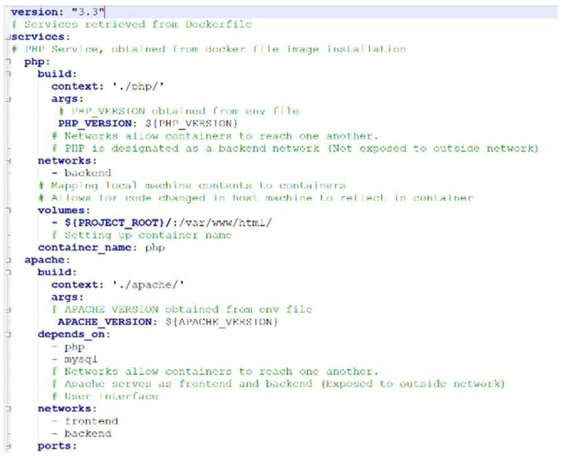
Docker Compose v3

Docker-compose.yml

With the functionality of Docker Compose, the development of a docker-compose.yml file was necessary. Docker compose utilizes the dockerfiles discussed earlier to run them simultaneously, while also configuring services, volumes and frontend/backend specifications.

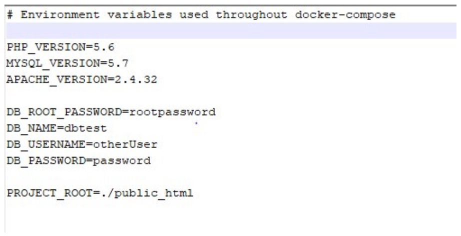
Docker Compose Services

Our services specified in the docker-compose.yml file are apache, php, MySQL, and phpMyAdmin (see figure3). Within these service specifiers, each is given a name, port, and version specification based off our .env file (see figure4)



```
version: "3.3"
# Services retrieved from Dockerfile
services:
# PHP Service, obtained from docker file image installation
  php:
    build:
      context: './php/'
      args:
       # PHP_VERSION obtained from env file
       PHP_VERSION: ${PHP_VERSION}
      # Networks allow containers to reach one another.
      # PHP is designated as a backend network (Not exposed to outside network)
    networks:
      - backend
    # Mapping local machine contents to containers
    # Allows for code changed in host machine to reflect in container
    volumes:
      - ${PROJECT_ROOT}/:/var/www/html/
      # Setting up container name
    container_name: php
  apache:
    build:
      context: './apache/'
      args:
      # APACHE VERSION obtained from env file
       APACHE_VERSION: ${APACHE_VERSION}
    depends_on:
      - php
      - mysql
      # Networks allow containers to reach one another.
      # Apache serves as frontend and backend (Exposed to outside network)
      # User interface
    networks:
      - frontend
      - backend
    ports:
```

Figure3



```
# Environment variables used throughout docker-compose

PHP_VERSION=5.6
MYSQL_VERSION=5.7
APACHE_VERSION=2.4.32

DB_ROOT_PASSWORD=rootpassword
DB_NAME=dbtest
DB_USERNAME=otherUser
DB_PASSWORD=password

PROJECT_ROOT=./public_html
```

Figure4

Docker Volumes

By utilizing volumes we are able to make sure that any code edited within the local host is updated into our containers within Docker. This provided us with a significant boost to productivity and testing functionality.

Within the docker-compose.yml file (figure3), the volumes for php and apache are mapping local machine contents to containers, while MySQL is mapping data towards var/www/html.

Docker Frontend/Backend Specification
Choosing to make a secure development stack, opting to make the apache service the only one exposed to the outside by being designated “Frontend”, while keeping php and MySQL “Backend” aided in this endeavor.

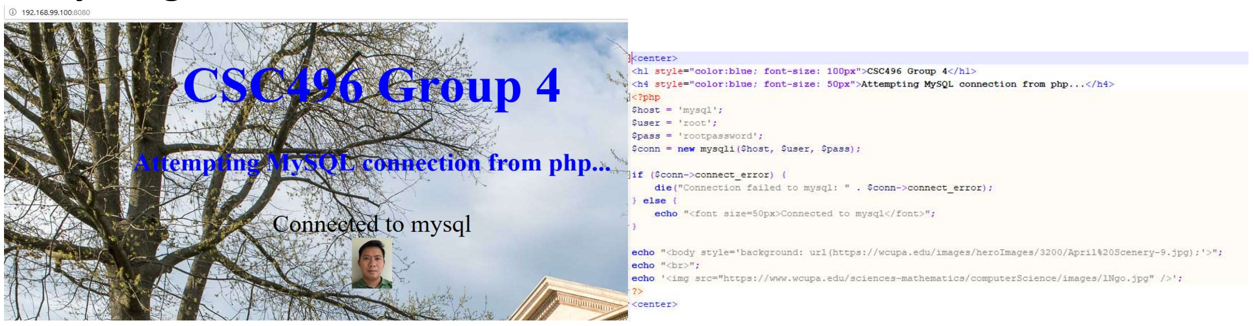
LAMP Stack Service

The components of the LAMP Stack Service are made up of Linux, Apache, MySQL, and PHP. By combining these components a fully configurable development environment is achieved, allowing for the launching of a web service with an apache frontend and a MySQL backend.

PHP

To illustrate the effectiveness of Docker and its various functionalities, an index.php file is required. Connecting to a database and illustrating our LAMP Stacks functionality was achieved through the use of the index.php by first using a command to connect to a database, then display if that connection was established, and then finally displaying a picture of a professor. This experiment conveys the early functionality of our LAMP Stack, but comes with a few limitations. It is first off, not deployed onto a cloud, it does not gather data from the database backend except for connectivity, and it thus does not serve the complete purpose of creating a website displaying professors, and their information collected by the database, while also being deployed onto CloudLab’s cloud infrastructure.

Early Progress on LAMP Stack Service



CSC496 Group 4

Attempting MySQL connection from php...

Connected to mysql

```
<center>
<h1 style="color:blue; font-size: 100px">CSC496 Group 4</h1>
<h4 style="color:blue; font-size: 50px">Attempting MySQL connection from php...</h4>
<?php
$host = 'mysql';
$user = 'root';
$pass = 'rootpassword';
$conn = new mysqli($host, $user, $pass);

if ($conn->connect_error) {
    die("Connection failed to mysql: " . $conn->connect_error);
} else {
    echo "<font size=50px>Connected to mysql</font>";
}

echo "<body style='background: url(https://wcupa.edu/images/heroImages/3200/April%20Scenery-9.jpg);'>";
echo "<br>";
echo '<img src="https://www.wcupa.edu/sciences-mathematics/computerScience/images/1Ngo.jpg" />';
?>
<center>
```

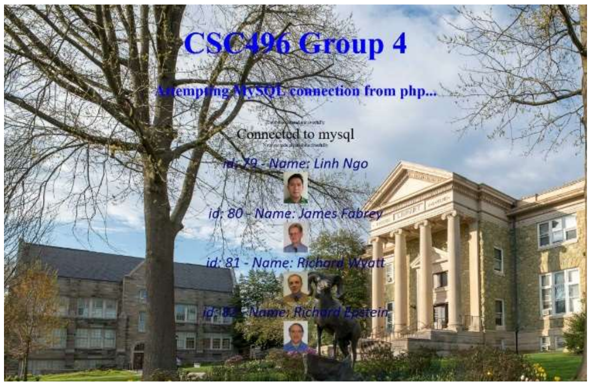
Figure6

Figure 6 illustrates the ability for us to connect to a database from our apache webservice and displayed connectivity and basic php commands onto a webpage.

Technical Report
5/8/2019
Installation of LAMP Stack Server utilizing Docker Toolbox
Group 4 – Jeffrey Jones, Andre Ibarrondo, Jacob Hansen

Finished PHP Version and Webpage

With the inclusion of conditions on what data to display, and the manual creation of a database, table, and data insertions; we were able to display the final version of our product, hosted on our local machines.


CSC496 Group 4

Attempting MySQL connection from php...

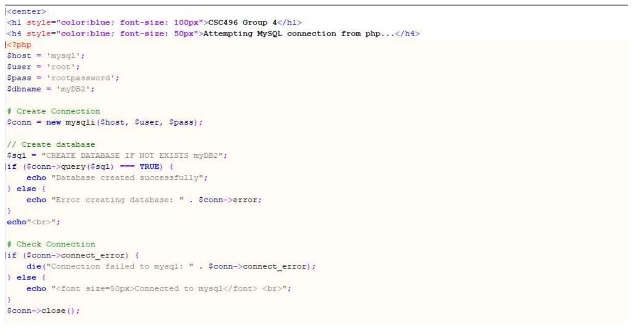
Connected to mysql

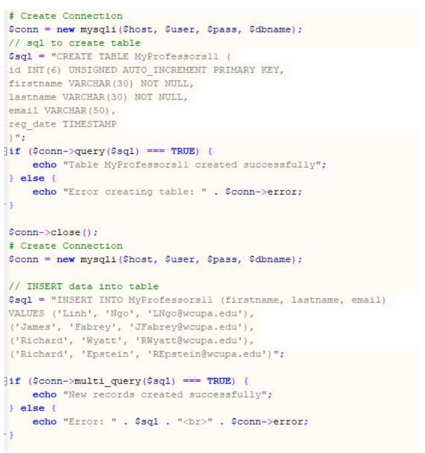
id: 79 - Name: Linh Ngo

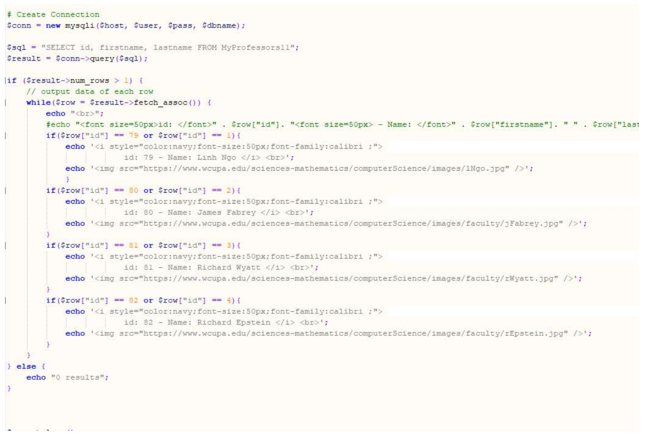
id: 80 - Name: James Fabrey

id: 81 - Name: Richard Wyatt

id: 82 - Name: Richard Epstein


```php
<center>
<h1 style="color:blue; font-size: 100px">CSC496 Group 4</h1>
<h4 style="color:blue; font-size: 50px">Attempting MySQL connection from php...</h4>
<?php
$host = 'mysql';
$user = 'root';
$pass = 'rootpassword';
$dbname = 'myDB2';

# Create Connection
$conn = new mysqli($host, $user, $pass);

// Create database
$sql = "CREATE DATABASE IF NOT EXISTS myDB2";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn->error;
}
echo"<br>";

# Check Connection
if ($conn->connect_error) {
    die("Connection failed to mysql: " . $conn->connect_error);
} else {
    echo "<font size=50px>Connected to mysql</font> <br>";
}
$conn->close();
```


```php
# Create Connection
$conn = new mysqli($host, $user, $pass, $dbname);
// sql to create table
$sql = "CREATE TABLE MyProfessorsll (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";
if ($conn->query($sql) === TRUE) {
    echo "Table MyProfessorsll created successfully";
} else {
    echo "Error creating table: " . $conn->error;
}

$conn->close();
# Create Connection
$conn = new mysqli($host, $user, $pass, $dbname);

// INSERT data into table
$sql = "INSERT INTO MyProfessorsll (firstname, lastname, email)
VALUES ('Linh', 'Ngo', 'LNgo@wcupa.edu'),
('James', 'Fabrey', 'JFabrey@wcupa.edu'),
('Richard', 'Wyatt', 'RWyatt@wcupa.edu'),
('Richard', 'Epstein', 'REpstein@wcupa.edu')";

if ($conn->multi_query($sql) === TRUE) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
```

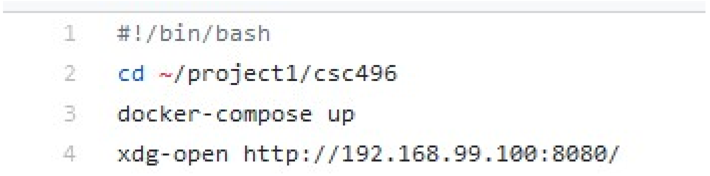

```php
# Create Connection
$conn = new mysqli($host, $user, $pass, $dbname);

$sql = "SELECT id, firstname, lastname FROM MyProfessorsll";
$result = $conn->query($sql);

if ($result->num_rows > 1) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "<br>";
        #echo "<font size=50px>id: </font>" . $row["id"]. "<font size=50px> - Name: </font>" . $row["firstname"]. " " . $row["last
        if($row["id"] == 79 or $row["id"] == 1){
            echo '<i style="color:navy;font-size:50px;font-family:calibri ;">
                        id: 79 - Name: Linh Ngo </i> <br>';
            echo '<img src="https://www.wcupa.edu/sciences-mathematics/computerScience/images/lNgo.jpg" />';
            }
        if($row["id"] == 80 or $row["id"] == 2){
            echo '<i style="color:navy;font-size:50px;font-family:calibri ;">
                        id: 80 - Name: James Fabrey </i> <br>';
            echo '<img src="https://www.wcupa.edu/sciences-mathematics/computerScience/images/faculty/jFabrey.jpg" />';
        }
        if($row["id"] == 81 or $row["id"] == 3){
            echo '<i style="color:navy;font-size:50px;font-family:calibri ;">
                        id: 81 - Name: Richard Wyatt </i> <br>';
            echo '<img src="https://www.wcupa.edu/sciences-mathematics/computerScience/images/faculty/rWyatt.jpg" />';
        }
        if($row["id"] == 82 or $row["id"] == 4){
            echo '<i style="color:navy;font-size:50px;font-family:calibri ;">
                        id: 82 - Name: Richard Epstein </i> <br>';
            echo '<img src="https://www.wcupa.edu/sciences-mathematics/computerScience/images/faculty/rEpstein.jpg" />';
        }
    }
} else {
    echo "0 results";
}
```

Deploying onto CloudLab's Cloud Infrastructure

After successfully deploying our service locally, it was time to set up our service onto CloudLab's Cloud Infrastructure. With the help our Dr. Ngo's docker profile, we were able to set up customized profile that contained scripts to automate the process of docker-compose up..

Automation

Automating the process of docker-compose composed of two steps. The first step was to create a python file that selected the various services that we needed to run docker successfully. These files installed docker onto ubuntu and prepared CloudLab's environment for us. Furthermore, the python file ran our custom bash script "automateComposeUp.sh" which changed the directory of the ubuntu path to our main folder "csc496" and then proceeded to run the command “docker-compose up” to finish the installation of our services, just as it would on a local machine (figure7).



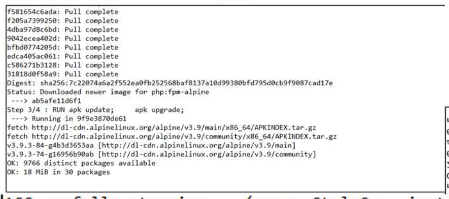
```
#!/bin/bash
cd ~/project1/csc496
docker-compose up
xdg-open http://192.168.99.100:8080/
```

Figure7

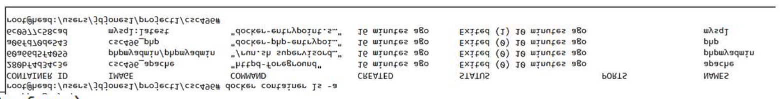
Problems Encountered with Automation

Our initial deployment of our services onto CloudLab’s cloud infrastructure was a success, displaying the full functionality of our website. Figure 8 shows the functionality of the commands inputted manually for our first deployment.

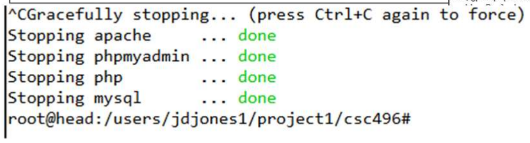
Docker-compose up **docker container ls -a Displaying all containers (stopped)**



```
f581654c6ada: Pull complete
f205a7399250: Pull complete
4dba97d8c6bd: Pull complete
9042ecea402d: Pull complete
bfbd0774205d: Pull complete
edca405ac061: Pull complete
c586271b3128: Pull complete
31818d0f58a9: Pull complete
Digest: sha256:7c22074a6a2f552ea0fb252568baf8137a10d99380bfd795d0cb9f9087cad17e
Status: Downloaded newer image for php:fpm-alpine
 ---> ab5afe11d6f1
Step 3/4 : RUN apk update;     apk upgrade;
 ---> Running in 9f9e3870de61
fetch http://dl-cdn.alpinelinux.org/alpine/v3.9/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.9/community/x86_64/APKINDEX.tar.gz
v3.9.3-84-g4b3d3653aa [http://dl-cdn.alpinelinux.org/alpine/v3.9/main]
v3.9.3-74-g16956b90ab [http://dl-cdn.alpinelinux.org/alpine/v3.9/community]
OK: 9766 distinct packages available
OK: 18 MiB in 30 packages
```



```
root@head:/users/jdjones1/project1/csc496# docker container ls -a
CONTAINER ID        IMAGE                     COMMAND                  CREATED             STATUS                      PORTS               NAMES
280bf4d34c36        csc496_apache             "httpd-foreground"       16 minutes ago      Exited (0) 10 minutes ago                       apache
e0aeed5f4059        phpmyadmin/phpmyadmin     "/run.sh supervisord…"   16 minutes ago      Exited (0) 10 minutes ago                       phpmyadmin
a0efd70de543        csc496_php                "docker-php-entrypoi…"   16 minutes ago      Exited (0) 10 minutes ago                       php
ec0977c58cad        mysql:latest              "docker-entrypoint.s…"   16 minutes ago      Exited (1) 10 minutes ago                       mysql
root@head:/users/jdjones1/project1/csc496#
```



```
^CGracefully stopping... (press Ctrl+C again to force)
Stopping apache     ... done
Stopping phpmyadmin ... done
Stopping php        ... done
Stopping mysql      ... done
root@head:/users/jdjones1/project1/csc496#
```

Graph of productivity

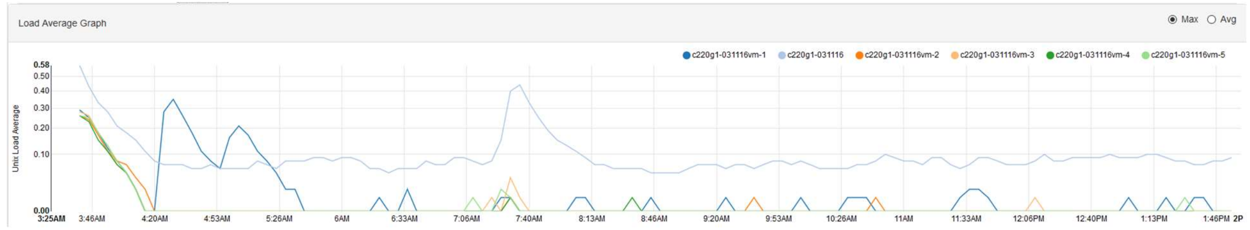
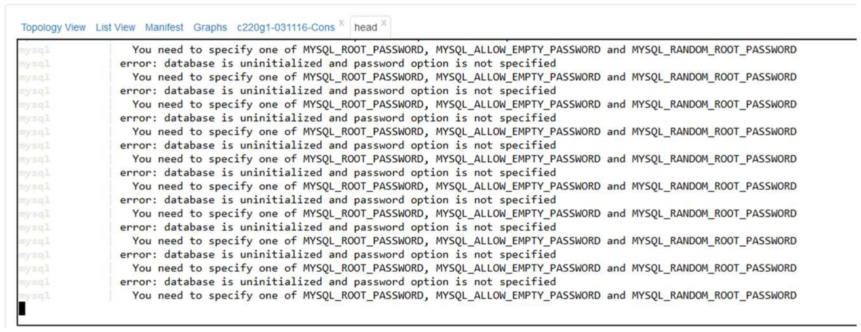


Figure8

However, for an unknown reason that we are still looking into while we write this report, even when we turned off the various services within CloudLab’s shell, we could still receive information from the website.

Furthermore, as time ran out on our second instantiation of our profile, we ran into issues. A major issue appeared which we still are trying to fix now. When we deployed our new instantiation with the same profile and the same settings, we could no longer connect to the database and errors about MySQL not being initialized occurred. This broke all functionality of our website and can be seen in figure 9



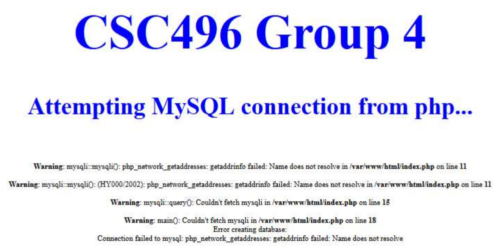
```
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
mysql          |   error: database is uninitialized and password option is not specified
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
mysql          |   error: database is uninitialized and password option is not specified
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
mysql          |   error: database is uninitialized and password option is not specified
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
mysql          |   error: database is uninitialized and password option is not specified
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
mysql          |   error: database is uninitialized and password option is not specified
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
mysql          |   error: database is uninitialized and password option is not specified
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
mysql          |   error: database is uninitialized and password option is not specified
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
mysql          |   error: database is uninitialized and password option is not specified
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
mysql          |   error: database is uninitialized and password option is not specified
mysql          |     You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
```

Figure9

Conclusion

Deploying a LAMP Stack on the local machines being developed in the initial stages of our experiment proved to be successful. We achieved everything we set out to deliver, that being a fully function LAMP Stack Service demonstrating its functionality through a web service displaying content retrieved from a backend database.
However, when it came to deploying onto CloudLab's cloud infrastructure, we ran into hours of being slowed down by the slow instantiation speed of our automated profile, which hindered our testing capabilities on why our database was failing to initialize after the second automated run.
Furthermore, an issue arouse on two of our developers machines that made docker no longer functional on their host machines, slowing down progress further.
With the set backs, we believed that, with enough time and resources, we would be able to successfully debug our MySQL service code to gain access to display the proper website again.

Error displayed



CSC496 Group 4

Attempting MySQL connection from php...

Warning: mysqli::mysqli(): php_network_getaddresses: getaddrinfo failed: Name does not resolve in **/var/www/html/index.php** on line **11**

Warning: mysqli::mysqli(): (HY000/2002): php_network_getaddresses: getaddrinfo failed: Name does not resolve in **/var/www/html/index.php** on line **11**

Warning: mysqli::query(): Couldn't fetch mysqli in **/var/www/html/index.php** on line **15**

Warning: main(): Couldn't fetch mysqli in **/var/www/html/index.php** on line **18**
Error creating database:
Connection failed to mysql: php_network_getaddresses: getaddrinfo failed: Name does not resolve