

Assignment 1
Coordinating Virtual Machines

Isaac Caruso, Jack Kearney, and Reid Smith
COSC 349
September 2, 2019

Introduction

The intention of this application is, through the coordination of three virtual machines, to create a web server to play video files. One of the virtual machines will act as the web server that uploads the files, the second will be a database to store the video files, and the third will play the video files. The virtual machines will start up simultaneously and interact in a coordinated fashion. The source code and other files were compiled in GitHub.

Initially, we decided to use the provided code and work on coordinating the three virtual machines; ideally, we wanted video files to be able to be uploaded through a select-and-insert method. Once selected and uploaded, the user would receive a message confirming the file of their choosing was uploaded. Once this file had been uploaded, the application would transfer the file to a database that would be run by the second VM. After a user had added video files to their database, one would be able to run a web server, hosted by the third VM, in which they could select the desired file and have the server play it back for the user.

The following sections outline the problems and difficulties we encountered in a bare-bones approach to a three VM solution. After multiple attempts, we decided to change our avenue of approach, instead opting to use a YouTube embedded link framework within a three VM solution. The final video display provides a playlist of uploaded videos, with a clear playlist button. This became our final product and the path to such a solution is documented below.

Initial Problems

Originally, we intended on using the provided code and a MySQL database server. However, after working on this pathway for a while, we decided to change our methods. Using the video “PHP: Video Upload and Playback (Using Database)”, [here](#), as a model, we decided to attempt a different strategy. The aforementioned video does not use VMs, rather it is just done on a local host. However, we thought that if we were able to successfully complete a local version of an upload and playback model, we would be able to split the three different tasks in between three different VMs. After that, our last task would be coordinating the VMs to communicate; this would be done with assistance from Lab 6 and other online resources.

After creating three different .php files, one for the uploading, one for the database, and one for the playback, we stepped through the aforementioned video in an attempt to successfully complete a local version of our desired final product. We encountered several difficulties in initializing the upload framework.

An essential part of working this model on a local scale is connecting the MySQL database with the upload architecture. In a debugging effort and intermediary step, we decided to migrate a copy of the file to a local repository, a home base from which the files could be fetched. This local folder acted in principle as the database that would be generated by our second virtual machine. We were successfully able to upload files through the upload architecture, yet several problems arose. Namely, we could not locate where the files had gone. Despite using a move-uploaded-file call in the index.php file, the video files were not moving

into the local directory. Furthermore, once a file was uploaded, we wrote script such that a successful upload message would be posted. However, this message proved to be non-volatile, such that even when the web server was reloaded, the previous message still remained.

Initially, we decided to tackle the problem of where the files were sent. Using a `mysql_connect` call, we attempted to link the database and the upload framework. However, this connection continuously failed. We attempted to use a PDO connection to link the two components, yet we continuously failed. After exhausting multiple solutions to the aforementioned problem, we decided to change avenues and explore the multi-VM architecture right away.

Our first step was reconfiguring the already-given database. After slightly tweaking our approach, we worked on adding three components to our database, such that ID, name, and location, URL, could all be stored. By modifying the given code, the database architecture was adjusted to the following.

```
vagrant-multivm-master — vagrant@dbserver: ~ — ssh • vagrant ssh dbse...

mysql> FLUSH PRIVILEGES;
[ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the right syntax to use n
ear 'PRIVELEGES' at line 1
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW databases;
+-----+
| Database |
+-----+
| information_schema |
| fvision      |
| mysql        |
| performance_schema |
| sys          |
+-----+
5 rows in set (0.00 sec)

mysql> USE fvision;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW tables;
+-----+
| Tables_in_fvision |
+-----+
| videos             |
+-----+
1 row in set (0.01 sec)

mysql> DESCRIBE videos;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| id     | int(11)   | NO   | PRI | NULL    | auto_increment |
| name   | varchar(255) | NO   |     | NULL    |              |
| location | varchar(255) | NO   |     | NULL    |              |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

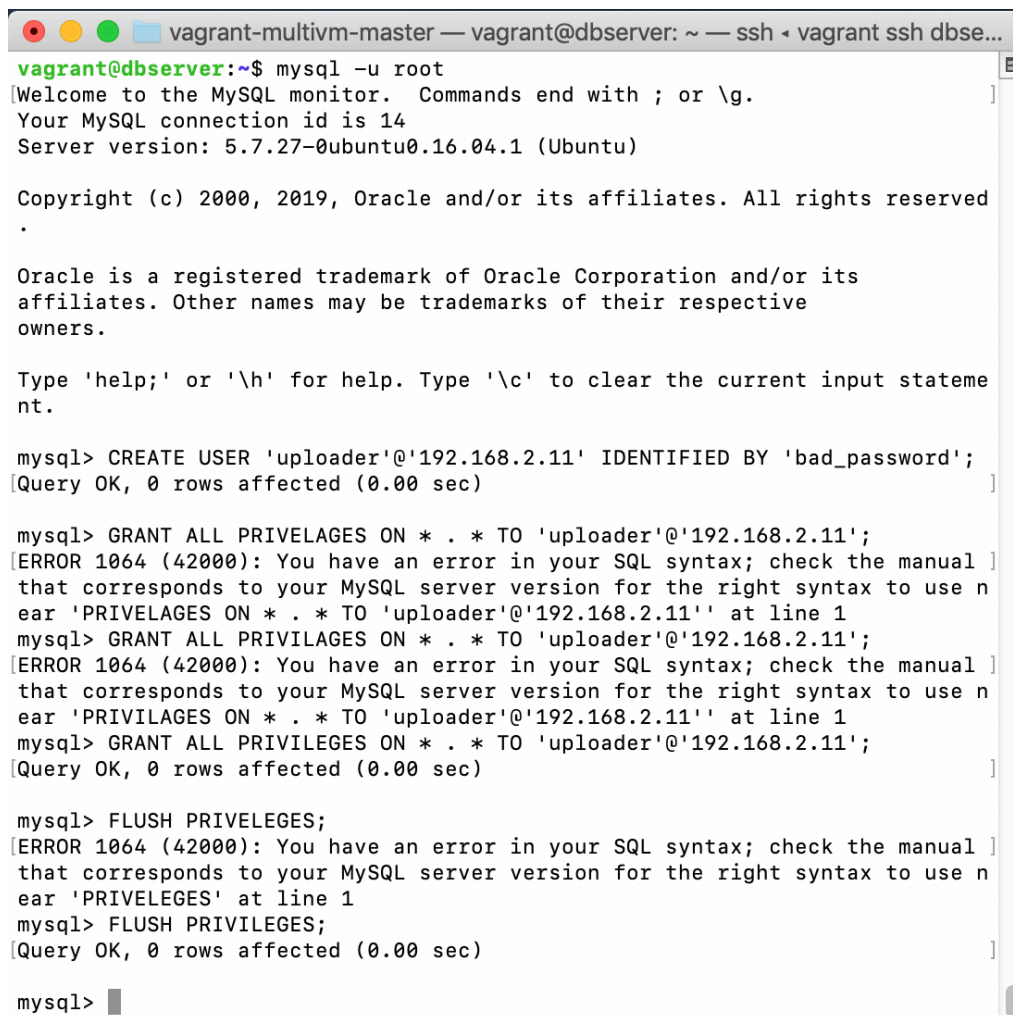
After restructuring the architecture of the database, we wrote an index and configuration file, such that the index.php file called the config.php file, and the config.php file would coordinate the connection between the two different VMs. The following screenshots demonstrate the changes made. Using a `mysqli_connect` call, we attempted to link the database and the upload components. After examining the VagrantFile to find the desired password and username, we attempted to run the connection.

```
index.php
1 <!DOCTYPE HTML>
2 <html>
3 <head>
4 <?php
5     include("config.php");
6
7     if(isset($_POST['but_upload'])){
8         $maxsize = 5242880; // 5MB
9
10        $name = $_FILES['file']['name'];
11        $target_dir = "videos/";
12        $target_file = $target_dir . $_FILES["file"]["name"];
13
14        // Select file type
15        $videoFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
16
17        // Valid file extensions
18        $extensions_arr = array("mp4","avi","3gp","mov","mpeg");
19
20        // Check extension
21        if( in_array($videoFileType,$extensions_arr) ){
22
23            // Check file size
24            if(($_FILES['file']['size'] >= $maxsize) || ($_FILES["file"]["size"] == 0)) {
25                echo "File too large. File must be less than 5MB.";
26            }else{
27                // Upload
28                if(move_uploaded_file($_FILES['file']['tmp_name'],$target_file)){
29                    // Insert record
30                    $query = "INSERT INTO videos(name,location) VALUES('".$_FILES["file"]["name"]."','".$target_file."')";
31
32                    mysqli_query($con,$query);
33                    echo "Upload successfully.";
34                }
35            }
36        }else{
37            echo "Invalid file extension.";
38        }
39    }
40
41 }
```

```
Config.php
1 <?php
2
3 $host = "localhost"; /* Host name */
4 $user = "root"; /* User */
5 $password = ""; /* Password */
6 $dbname = "tutorial"; /* Database name */
7
8 $con = mysqli_connect($host, $user, $password,$dbname);
9 // Check connection
10 if (!$con) {
11     die("Connection failed: " . mysqli_connect_error());
12 }
```

However, when the VM was run and the connection was tested, the following error message was produced: *Connection Failed: Connection Restricted*. However, after examining the code and the lab note regarding communication between a server and a virtual machine, it was noted that the incorrect IP address was being used. After adjusting the IP address, the connection was retested, and the following error message was produced: *Connection failed: Access denied for user 'root'@'192.168.2.11' (using password: YES)*.

After exploring multiple avenues to avoiding such a problem, it was decided the best solution would be to create a new user, where we had explicit control over the username and password. All privileges were granted. The following syntax was used.

A screenshot of a terminal window titled 'vagrant-multivm-master — vagrant@dbserver: ~ — ssh - vagrant ssh dbse...'. The terminal shows a user logging into a MySQL database as 'root'. The user then attempts to create a new user 'uploader' with a password 'bad_password' and grant all privileges to it. The terminal shows three attempts to grant privileges, each resulting in an 'ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'PRIVELEGES ON * . * TO 'uploader'@'192.168.2.11'' at line 1'. The user then attempts to flush privileges, which also results in the same error. The terminal ends with the prompt 'mysql>' and a cursor.

```
vagrant@dbserver:~$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 5.7.27-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE USER 'uploader'@'192.168.2.11' IDENTIFIED BY 'bad_password';
[Query OK, 0 rows affected (0.00 sec)]

mysql> GRANT ALL PRIVILEGES ON * . * TO 'uploader'@'192.168.2.11';
[ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
]
that corresponds to your MySQL server version for the right syntax to use n
ear 'PRIVELEGES ON * . * TO 'uploader'@'192.168.2.11'' at line 1
mysql> GRANT ALL PRIVILAGES ON * . * TO 'uploader'@'192.168.2.11';
[ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
]
that corresponds to your MySQL server version for the right syntax to use n
ear 'PRIVILAGES ON * . * TO 'uploader'@'192.168.2.11'' at line 1
mysql> GRANT ALL PRIVILEGES ON * . * TO 'uploader'@'192.168.2.11';
[Query OK, 0 rows affected (0.00 sec)]

mysql> FLUSH PRIVELEGES;
[ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
]
that corresponds to your MySQL server version for the right syntax to use n
ear 'PRIVELEGES' at line 1
mysql> FLUSH PRIVILEGES;
[Query OK, 0 rows affected (0.00 sec)]

mysql>
```

After making the above changes, the connection successfully worked. Our next step was coordinating the third virtual machine. After granting all privileges, we noticed that this procedure would have to be done every time. Instead, we used the user and password from the given VagrantFile. The connection worked successfully and avoided the above redundancy problem.

Our next step was initializing a third virtual machine. Modifying the VagrantFile to create a separate web server, we were successfully able to run three VMs. One webserver for uploads, one web server for playback, and one database for storage. Additionally, we were confident that the upload web server and the database were connected and communicating, as no error message was displayed when a file was uploaded. Furthermore, both web servers were up and displaying content.

Modeling our code off of the PHP video, we added a hyperlink on the upload server that brought someone to the playback server. Furthermore, we added more HTML code to the upload server to clean it up. Our largest problem at this stage was the connection between the database and the upload web server. Despite the fact that they were connecting, we were unsure as to where the files were being sent; there seemed to be an error in the `move_uploaded_file` line in the `index.php` file. This was our largest obstacle in completing the assignment.

While we worked on successfully storing the video files, we were able to successfully update both webserver, adding more text and links that connect the upload architecture to the playback architecture, and vice versa. The major problem we worked to tackle was how to successfully display the files on the playback web server; underlying this problem was the issue of correctly storing the video uploads in a spot that could be accessed, written to, and read to, by both the upload and playback servers.

After exploring multiple solutions to the database problem, we decided to change avenues. Still exploring a three VM solution, we decided to coordinate three virtual machines such that an uploader took a YouTube embedded link and a name, stored them in the database of a second virtual machine, and a third virtual machine coordinated the video selection and playback.

Because of our limited knowledge of MySQL and HTML code, the two following links, seen in the relevant sources section, were heavily relied upon. The HTML skeleton was mimicked, yet the internal components were adjusted for our own adaptation. The following sources are listed: <https://www.tutorialrepublic.com/php-tutorial/php-mysql-insert-query.php> and <https://www.tutorialrepublic.com/php-tutorial/php-mysql-select-query.php>.

The architecture of the solution was largely similar to what we had previously attempted. There are three .php files, namely `index` (upload architecture), `insert` (database storage and verification), and a second `index` file (video playback). The upload `index` and `insert` file is in the `www` directory, and the second, `playback index` is stored in the `xxx` directory. Users are able to insert a name and embedded URL, which are in turn pushed into the database, and the video playback architecture grabs and displays files from the database.

Opportunities for Improvement

Despite the fact that we were pleased with the construction and final product of our application, there are certainly avenues other developers could undertake to improve and build upon the pre-existing architecture.

For example, one possible addition could be abandoning the need to name and provide an embedded URL. Instead, the application could be modified to simply require a URL from YouTube, and internally, the program would fetch the embedded URL and name of the video. Furthermore, the program could exist outside the scope of YouTube, rather accepting video files from a multitude of other online resources.

Further adaptation to the program could include a genre-specific playlist, such that when videos were uploaded, the program would be able to unpack the tags inherent in the video, sorting a user's uploads into songs, trailers, instructional videos, comedy videos... The playlists could also become shareable, such that one user of the application would be able to quickly and easily acquire a shareable link that would allow other users, on different machines, to access their playlist.

Brief Instructions

The following steps outline the instructions for which a 3rd party developer could follow to interact with our application.

A video screen capture of the instructions can be viewed [here](#).

*Important: once the GitHub directory is downloaded, from the steps below, it is crucial that users do work inside of the folder COSC349_1-youtube. Everything outside of that folder is simply a record of progress and previous attempts.

- 1) Using GitHub, navigate to the following webpage:
https://github.com/icaruso21/cosc349_1
- 2) Using the clone or download tab, select clone with HTTPS and copy the link
- 3) Navigating to a directory of your choice through the terminal shell, type the following commands...
 - a. `git clone insert https link here`
 - b. `vagrant up`
- 4) Once vagrant up has been run, the three virtual machines will be initialized. Navigate to the page 127.0.0.1:8080 – this represents the upload architecture
- 5) Once a user is at the aforementioned page, they will be able to upload their own videos. Five videos will be present in the playlist at the beginning, for demonstrative purposes.
- 6) Once on YouTube, users are free to select a video of their choosing. Once a user has found a video of their choice, follow the steps below to upload the video...

- a. Click *Share*, located near the thumbs up and thumbs down buttons
 - b. Click *Embed* and copy the link that appears to the right of the video thumbnail
 - c. Navigate back to the aforementioned upload framework, create a unique name, paste the embed link, and click upload
 - d. If a video has been successfully uploaded, a message reading “Records added successfully” will appear
 - i. Of note: if the database is not connected, a message reading “ERROR: Could not connect” and pertinent information will appear
- 7) After uploading a video, users will be able to click a button that allows them to upload another video or view their playlist. Users are also able to navigate directly to their playlist from the start of the upload architecture
- 8) Users are now free to upload and view their videos

The following steps outline where a developer could go to make their own changes...

- A) The files that would be changed are the following: two index.php files and one insert.php file.
- B) One folder contained in the git clone procedure is www; inside this folder is insert.php and index.php. This insert file is in reference to the database and storage and the index.php file is in reference to the upload architecture. Any php or HTML changes could be made inside these files.
- C) Another folder obtained from the git clone procedure is named xxx. Inside this folder is an index.php file and a clear.php file. The clear file relates to clearing the playlist, and the index.php file handles the video playback feature. Any changes looking to be made to playback or playlist management would be done in the aforementioned files.

For first initialization, the download volume is 92KB. The initialization time is 6 minutes and 18 seconds.

Download volume and time for subsequent builds is also expected to be similar.

Relevant Sources

- Abell12. “PHP: Video Upload & Playback (Using Database).” *YouTube*.
https://www.youtube.com/watch?v=MpRRckA2Bo8&feature=share&fbclid=IwAR1yW4OhJ2M-7lfdmc8VSynLP_kWrg_LJGkHovg4qbZgoo_hSfG9CKs08vA
- B., Rashmi. “Setting Up PHP 7 on Mac with Vagrant and VirtualBox (OSX and macOS).” *Ryadel*. October 29, 2017. <https://www.ryadel.com/en/php-7-mac-vagrant-virtualbox->

osx-macos/

Eyers, David. "Lab 6: Further Exploration of Vagrant." *COSC349 Laboratory Assignments*. August 14, 2019. <https://hackmd.io/@dme26/H17w3aIXB>

"HTML Audio Player." *w3schools.com*. https://www.w3schools.com/html/html5_audio.asp

"HTML File Selector." *w3schools.com*.
https://www.w3schools.com/tags/att_input_type_file.asp

"HTML Forms." *MDN web docs*. <https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms>

Megía, Alberto. "Access Denied for User 'test'@'localhost' (using password: YES) Except Root User." *Stack Overflow*. December 3, 2013.
<https://stackoverflow.com/questions/20353402/access-denied-for-user-testlocalhost-using-password-yes-except-root-user>.

"PHP File Upload." *w3schools.com*. https://www.w3schools.com/php/php_file_upload.asp

"PHP MySQL SELECT Query." *TutorialRepublic*. <https://www.tutorialrepublic.com/php-tutorial/php-mysql-select-query.php>

"PHP MySQL INSERT Query." *TutorialRepublic*. <https://www.tutorialrepublic.com/php-tutorial/php-mysql-insert-query.php>

"Sending HTML Form Data." *MDN web docs*. https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Sending_and_retrieving_form_data

Singh, Yogesh. "Upload and Store video to MySQL Database with PHP." *Makitweb*. May 14, 2018. <https://makitweb.com/upload-and-store-video-to-mysql-database-with-php/>

Screen Capture Instructions: <https://www.youtube.com/watch?v=MAi2lfXbXg4>