Assignment 2
Coordinating Virtual Machines with the Cloud

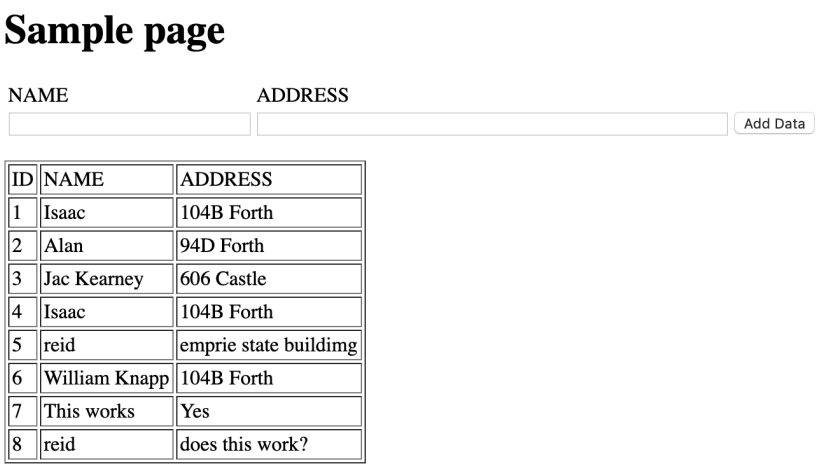
Isaac Caruso, Jack Kearney, and Reid Smith
COSC 349
October 4, 2019

Introduction

In beginning Assignment 2, our group decide to expand upon our deployment and implementation of a video storage and playback application. The application ran through the coordination of three virtual machines, a database and two webservers. For this cloud-oriented deployment of the application, we planned to move the database to the cloud, using Amazon RDS. The two web servers will still function as they did in Assignment 1.

Process

Our first step was conducting research on transitioning from a virtual machine database to an RDS database. We also started by pushing our old Assignment 1 files into the new Assignment 2 repository. Following an Amazon RDS tutorial, we were able to construct a database with a storage table.[1] Furthermore, we were able to connect this database, with the corresponding storage table, to an EC2 instance. The figure below demonstrates the implementation and testing of the aforementioned connection.



Sample page

NAME ADDRESS

Add Data

ID	NAME	ADDRESS
1	Isaac	104B Forth
2	Alan	94D Forth
3	Jac Kearney	606 Castle
4	Isaac	104B Forth
5	reid	emprie state buildimg
6	William Knapp	104B Forth
7	This works	Yes
8	reid	does this work?

[1] The Amazon RDS tutorial is linked here: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateDBInstance.html?fbclid=IwAR1drrgv_tHL-dGyCjAGq1NzYgXwXznPfYH7_Bne8PD2wxUYLkLyqE8K-xk

The above screenshot is indicative of a successful connection between the database and the webserver. Our next step was to successfully integrate Vagrant into the assignment, allowing for the coordination of three virtual machines. Upon beginning this step, it was made apparent that coordinating Vagrant and AWS instances would be quite excessive given the nature of the project.

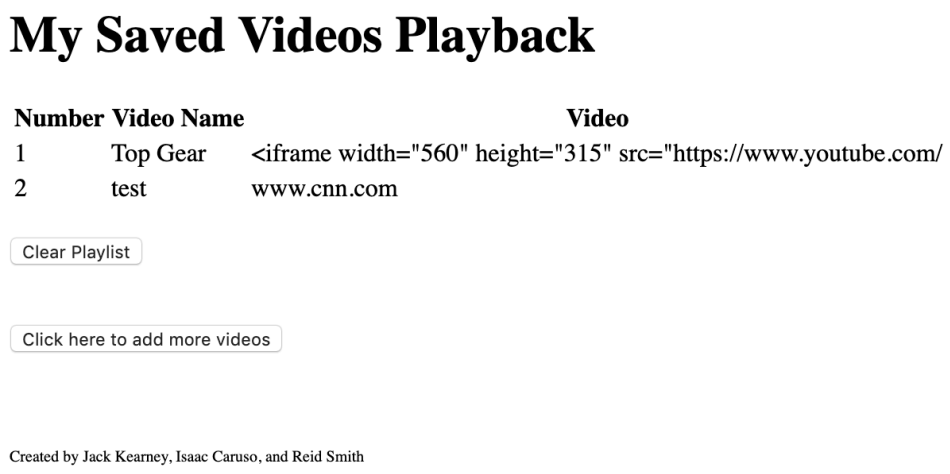
We decided the best way to tackle this problem would be to instead use three Amazon instances: two EC2 webservers and one RDS database. Another Amazon tutorial was used to build and connect two EC2 webservers.[2] The driving reason behind this decision was the efficient manner in which Amazon instances can communicate; all being from the same provider, Amazon provided extension tutorials on both initializing and connection their cloud infrastructure. We were also interested in exploring how RDS and EC2 instances work in conjunction. After initializing and connecting three Amazon instances, our next step was to transfer in our own architecture from the previous assignment, namely the insert and index files in the corresponding www and xxx folders.

In transferring over the index and insert files, it was necessary to alter the code for our redirect buttons, namely the button to upload another video and the button to view one's playlist. Previously, these buttons took action by switching users between the two Vagrant instances. Because we abandoned such an implementation, the connections were changed such that the buttons brought users either back to the first EC2 instance, the upload architecture, or the second EC2 instance, the playlist architecture that was supported by the RDS database. Of note is that

[2] The tutorial for the webserver implementation is linked here: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateWebServer.html

both web servers do communicate with the database; the upload webserver pushes to the database, whereas the playback webserver pulls from the database.

After changing the addresses and implementing buttons, we were able to get the two webservers bouncing back and forth between each other and the database. Our next step was to implement our database architecture from the previous assignment into the RDS instance. After editing the appropriate files, we were able to add our own architecture into the database. One problem we began to run into was the correct displaying of our videos, despite the fact that none of the code was changed. The below image demonstrates the current problem with our implementation.



Evidently, there was an apparent disconnect when an embed link was inserted. Frame characteristics were included, and a thumbnail of the video was not displayed. In troubleshooting the problem, we decided to attempt to clear the table. After clearing the table and adding data, the following image was displayed.

My Saved Videos Playback

Number	Video Name	Video
1	Top Gear	<iframe width="560" height="315" src="https://www.youtube.com/
2	test	www.cnn.com

Clear Playlist

Click here to add more videos

Created by Jack Kearney, Isaac Caruso, and Reid Smith

After deleting the table, we attempted to re add the index.php file. The table was successfully deleted, yet we still could not get the video to display correctly.

My Saved Videos Playback

Number	Video Name	Video
1	Top Gear	<iframe width="560" height="315" src="https://www.youtube.com/embed/r3i-HRxf8z4" frameborder="0" allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

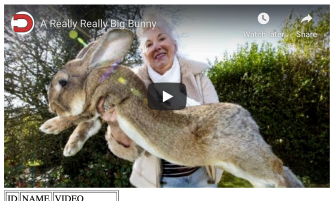
Clear Playlist

Click here to add more videos

Created by Jack Kearney, Isaac Caruso, and Reid Smith

After messing around with several different solutions, we attempted to solve the problem by hard coding in the actual YouTube embed link in the code, outside of the table itself. The following screen capture demonstrates that the above logic was successful.

My Saved Videos Playback



ID	NAME	VIDEO
1	test	RMh7BcqzDsY

Number	Video Name	Video
1	test	RMh7BcqzDsY

Clear Playlist

Click here to add more videos

Created by Jack Kearney, Isaac Caruso, and Reid Smith

Evidently, this is an impractical solution as hard coding videos in will not work for an application. However, this did indicate that if we can isolate the embed YouTube variable, we will be able to "hardcode" in this URL, outside or inside of the table, such that it will display. We began working on attempting to implement this logic.

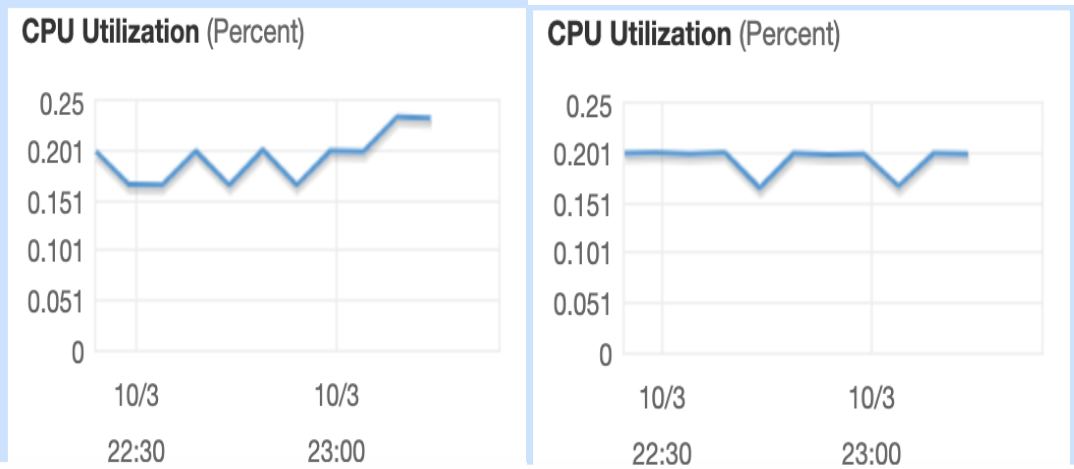
One of the other avenues we pursued was using PLYR, an open-source HTML player that provides clean and effective use. The video player is simple and beautiful, and we thought it would add nicely to the project. Upon further research, we began weighing our options in deciding the best avenue to pursue.

One of our noted areas for improvement was being able to not just upload YouTube embed links, but websites in general. In pursuing this goal, we decided to attempt an implementation in which the application would function as a bookmarking machine, a way to save favorite videos and websites into one location. We decided we wanted a table, with a name and an ID number, along with a URL for the user to save to revisit a page.

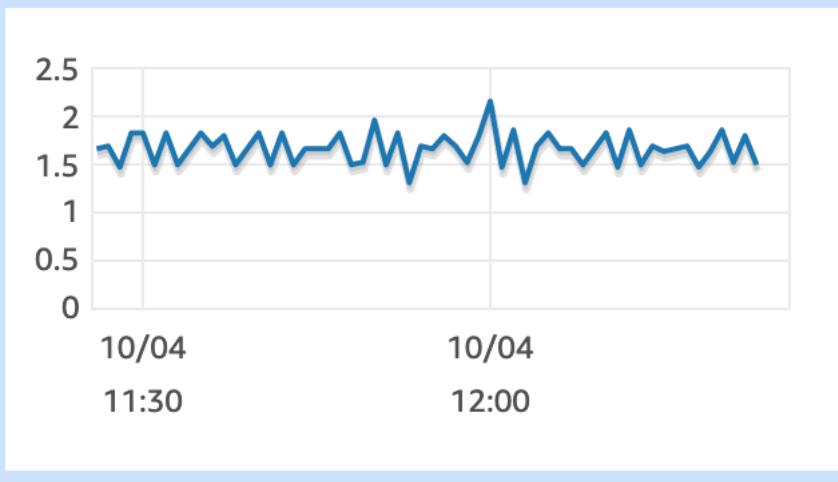
The main problem we encountered was having clickable links inside of a table. We were successfully able to add links outside of the table, but not inside of it. For some reason, despite hours of research and attempts, were unable to hyperlink the URLs in the table. Despite this shortcoming, we were able to add IDs, names, and URLs into the table; a bookmark application for users to revisit their favorite videos and sites. Eventually, we discovered a period was our only problem. After inserting this, we were able to obtain clickable links!

One of the key elements we wanted to understand was how much each Amazon instance was costing us, along with the CPU load required to implement and deploy our application. Our first EC2 instance was running an approximate CPU load of 0.151% to 0.24%. Our other EC2 instance was running slightly lower, with CPU utilization between 0.151% and 0.201%. The

below figures are respective to the aforementioned descriptions.



The RDS was using considerably more CPU power, hovering between 1.5% and 2.0%. This finding is reasonable, as the database is involved in both web servers and stores all the necessary data. The figure below represents the CPU load.



Along with CPU load, it is both practical and interesting to understand the real coast behind the deployment. Since beginning the project on 9/30, $5.27 has spent on deploying the three instances. As the RDS instance requires roughly 8.333333 times more computational power, it is reasonable to assume that each EC2 instance has cost $0.316 and the RDS instance has cost $4.637. We sought more precise data, as it may be incorrect to assume that an RDS and

EC2 cost the same, yet this information was unavailable given our current AWS permissions. Billing and invoice statements were all blocked.

Of note is how Vagrant, an automated setup, would have functioned within this architecture. Had we used Vagrant, those instances would have functioned as the two web servers, one being the upload page and the other being the playback page. Much like Assignment 1, the web servers would have communicated in conjunction with the database to coordinate the deployment of the application. A Vagrant setup would have provided for automation of the deployment of the application, inherent in this is the notion that the Vagrant instances were communicating with the Amazon RDS. As aforementioned, our motivation for using all Amazon instances was the ease with which they communicate, along with the in-depth tutorials provided by Amazon.

Brief Instructions

*several videos are already contained within the application for demonstrative purposes
*an instructional video will be contained within the GitHub repository

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

1) Navigate to the following webpage: http://ec2-18-208-199-159.compute-1.amazonaws.com/?fbclid=IwAR3b2Yok8CDmOSmJykahhBDKWgkzQIU-_UaHLGQ77Xlv0dm7OrJFiLkwVAw

2) Once a user is at the aforementioned page, they will be able to upload their own videos.

3) Once on YouTube, or any other website, users are free to select a video or site of their choosing. Once a user has found a video or site of their choice, follow the steps below to upload the video…

    a. Navigate back to the aforementioned upload framework, create a unique name, paste the link, and click upload
    b. Of note: if the database is not connected, a message reading “ERROR: Could not connect” and pertinent information will appear

4) 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

5) Users are now free to upload and view their videos

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

*all files are contained within the var folder
*also of note, in order to ssh into the below files, users need a key pair. As it is not safe to include this information in a public GitHub, it has been excluded.

A) Inside of the www folder is an inc folder. Navigate into this folder and access the dbinfo.inc file. Any developer who wishes to modify the database connections would take action here.

B) If a developer wanted to change the web servers, the necessary files are contained within the html folders, on web servers 1 and 2. The user can access these two separate servers by sshing into them. If a user wanted to change the upload page, this could be done in the index.php file contained within webserver1. If a user wanted to change the playback page, this could be done within the same index.php file, however this file will be contained within webserver2.

References

"Adding hyperlink to a variable in html table." https://stackoverflow.com/questions/20363408/adding-hyperlink-to-a-variable-in-html-table?fbclid=IwAR2nZ81fL0CHn93YAxGY5V6vgMfnOqC7YUNJW8HUCNa6yHEfB2zQWFSRFyU

"Step 1: Create an RDS DB Instance." https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateDBInstance.html?fbclid=IwAR1drrgv_tHL-dGyCjAGq1NzYgXwXznPfYH7_Bne8PD2wxUYLkLyqE8K-xk

"Step 2: Create an EC2 Instance and Install a Web Server." https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Tutorials.WebServerDB.CreateWebServer.html

*The below reference section is from Assignment 1; It was included as our implementation is 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/