COSC349
Assignment 2
Darcy Knox
7712753

NBA Greatest players game simulator

This application allows you to select teams made up of NBA greats of your choice, and simulate what the outcome of the game would be. The outcome of the game is based on the selected players' ratings. Ratings are predetermined and do not change.

The application is built using two Apache2 web servers (one serving the player selection interface, and one other serving the game simulator), and a database server that stores player information and manages queries from the web servers.

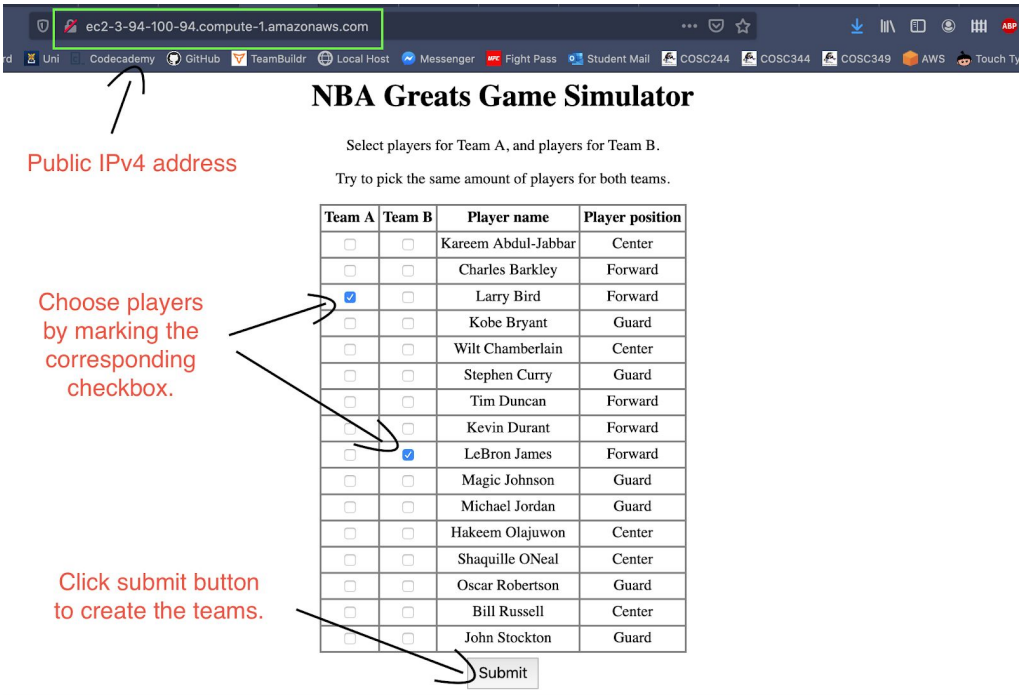
The web servers are Vagrant VMs running on AWS as EC2 instances, and provisioned using SSH. The database server is a MySQL engine running on the Amazon Relational Database Service (RDS). The front end and database querying is written in php and javascript.

User interaction

Note: As the public IP addresses are not-static and are created upon launching each web server instance. Therefore the hyperlinks that would link a user from one page to another have been removed, users must instead enter the addresses as follows.

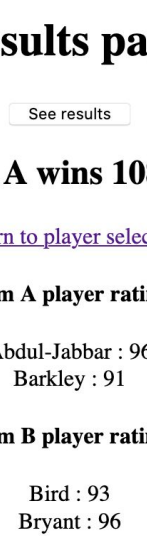
1. Go to the player selection interface by opening a web browser and entering the address http://ec2-3-94-100-94.compute-1.amazonaws.com/

2. Select two teams of players and click submit.



3. Go to the results page at the address ec2-54-91-114-105.compute-1.amazonaws.com

4. and click see results to see the winning team and score.



Deployment process

Choosing cloud technologies

I started off by first evaluating my options in terms of what cloud technologies I could use for deploying an application that I was most familiar with. Initially I looked at CloudCannon, which is a CMS for static sites built on Jekyll. I decided against using CloudCannon, as my existing application design wouldn't have worked as CloudCannon's static site technologies don't support the backend technologies I used in the application.

I wanted to test both Docker and AWS to see which would be a more fluid workflow for the project. I ran into several dependency issues with installing and using the vagrant-aws plugin. I worked for a while with Docker but slowly figured out that the learning curve for the technologies involved with cloud deployment was quite steep. I managed to solve my dependency issues, and decided to use AWS as the process was the most straight-forward and I was able to revisit lab material that was quite helpful.

I decided to continue using Vagrant as I found the configuration process was relatively simple, also the CLI is easy to use. Vagrant has well written docs and many resources that help with troubleshooting.

Configurations

The VMs are created within AWS as EC2 instances. EC2 features and support had all the necessary requirements for deployment.

Security credentials and key-pairs were set up to allow log-in and access of the instances. I had one issue with this where I mistakenly had my security credentials stored in a file of the project repo that I pushed to github. After that I made sure that credentials were only stored in environment variables.

Setting up two security groups, used by both instances, allow for ssh access as well as HTTP and HTTPS access.

Availability zones and subnets were created with us-east-1. I was able to find the corresponding subnet using the aws-cli. I had some trouble with availability zones where the subnets weren't working as my aws console was set to ap-southeast-2 (Sydney) which gave different information than when I set the console to N.Virginia.

I used the Ubuntu Amazon EC2 AMI Locator to get an appropriate AMI. My search criteria was my selected region and "HVM" (hardware virtual machine). The AMI is the machine image that provides the virtual hardware and operating system.

The instance type t2.micro, performed very slowly in stages. I changed this to use the default m3.medium instance type. This reduced page load times significantly and made the testing process much easier.

I had omitted the database to separate and further simplify the deployment. After testing the web servers correctly displayed the application, I decided to use the Amazon RDS (relational database service) to serve the MySQL database.

I created an RDS database instance with basic lightweight configurations, and assigned it a security group that allows open access (not very secure, but also low risk). Then applied the public IP and credentials to the php files that query the database. I then used MySQL workbench (a database management tool) to test the connectivity of the database, and to execute SQL commands. Once this was done, the application was behaving as expected.

Reflection

If I was to do this again from the start, I might instead try to create an application that could be used with static web technologies like CloudCannon. I also think that Docker would have been easier to use if the application was in an early state of development. I found AWS to be very powerful, and supportive in terms of tutorials, documentation, and support. I will look at using more of the AWS services in the future.