Jenna Abbott

James Ruh

Christopher D’Agostino

Quinn Huang

Deliverable 3

We used our deliverable three to make improvements on our Minecraft server and the progress that was made in deliverable two.

One of the goals we had was to set up an installation script that allowed us to quickly install all the necessary dependencies for Minecraft server. We wanted to make the setup process as easy as possible. This script is created and pushed to the GitHub repository for the project. It was tested on the instance of the virtual machine on Google Cloud several times. This script sets up the operating system and installs all the latest updates and all the necessary dependency using the `apt` package manager.

We start by using the command `apt update & apt upgrade`. This one line is actually two commands (as indicated by the ampersand). The first command updates the package list. It does not install any new software updates, it simply finds out which version the software installed on the system is and whether new packages have been released for these software. If there are new packages available, the package manager will now know about them. Then it installs java and the GNU terminal multiplexer screen. It also will create a minecraft directory in the home directory. It will then install the latest Minecraft server package from Molang using wget. It will then run the server.jar file using java. This first time the server will not be completely set up, because the user needs to agree to an end-user agreement. For Minecraft server, this is done by setting the eula variable in the eula.txt file to be true. This script also does this step automatically. After that the script will run the java command again and this time Minecraft world will be spawned and a Minecraft server will be set up.

A script was created that included

There were some challenges when we were writing the script. For one, there is still some level of manual work involved. Secondly, the script automatically spawns the world and sets up the server and does not use screen. Therefore, this means that when you exit out of the process or disconnect from the Google Cloud instance, your server will no longer be running. We were unable to get screen and then run java inside the screen session working just using a script.

Therefore, once it shows you the Minecraft server is up and the script is working. You can exit out of the server using Ctrl-Z or the ‘/stop’ command. This will stop the server and the java process. You then need to look up the process using the bash command `ps`, which will show a list of current running processes, this will also show you the PID of the process. By killing the java process using the command `kill -9 *PID`*, you can stop the java process and thus clear up the port so that you can run the Minecraft server again on that same port. Alternatively, use the other script which stops after it edits the eula.txt file and simply run the screen command and java command on the server.jar file manually.

With Google Cloud you can add different levels of administrators/moderators who are allowed to have ssh access under the IAM & admin page. We just followed the direction in the Google Cloud documentation and were able to have this set up.

We thought about changing the game’s rendering distance so that it would save resources within our instance. After multiple stress tests, we realized that the default render distance wasn't going to be that much of a strain on the system. Google allows us to customize the size of the memory and hard drive we want to use. For this project we didn’t need to have such a large threshold. In the future if we wanted to increase the amount of users on the server we could easily add more “hardware”.

After getting the server up and running, we decided to invite a few friends on to test it. We started with 10 players. Minecraft only renders active chunks. Chunks are 16 blocks wide, 16 blocks long, and 256 blocks high, which is 65,536 blocks total. Chunks are generated around players when they first enter the world, and as they wander around the world, new chunks are generated as needed. The largest draw on memory is when a player creates any automated farms. The more entities that are spawned in the more memory is needed to handle it. If we set the hardware specs too low a player could crash the server by breeding too many cows. For our stress test, we told our friends to explore, load in as many chunks as possible, and attempt to break the server by doing various resource intensive activities. Also, as admins, we spawned in all types of entities. This may have been overboard, as the amount of hardware we initially set up was more than adequate to handle 1000’s of cows, sheep, mobs, and other entities. Overall, our stress test proved that the server was functional and operational.

The comments from our second deliverable allowed us to reflect on our project for deliverable three. Our project had some hardships when we were creating the Minecraft server. There were issues within the real-world as well as problems in the virtual workspace. One of our team members had to get through power outages from thunderstorms and wi-fi issues.

When we were first setting up the game, some players had issues getting into our Minecraft world. We figured out that this was because the player’s settings were not the same as our server’s settings. The versions of the game had to be the same for players to connect. We checked our server to make sure that it was properly running before assuming that the problem was on the player’s side.

The Java version of our player’s had to be the same version of Java or a newer version of Java as our Minecraft server for it to run properly on their end. The initial setup process was tricky figuring out the functions and modification options. We were able to find the correct settings and versions so that the server would be running from the correct instance within Google Cloud.

The learning curve of Google Cloud was not too difficult, but there were still the beginning challenges of learning any new application. However, the process of setting up a Google Compute Engine instance was easier to understand than the process of setting up a CloudLab OpenStack environment, because everything was readily available to us in one location with an easy to understand user interface. Our main issues were the technical problems we were encountering when working with CloudLab to set up an OpenStack cloud compute platform, but once we changed over to Google Cloud, our second and third deliverable were easier to dive into.

We thought that it would be costly to go from CloudLab to Google Cloud. Amazon Web Services would have made us pay as we were using their resources. We decided to change course and try Google Cloud. There were lots of questions regarding the Google Cloud payments. Google Cloud was the best route for us because we were able to receive free trial money in our account and expand our project for the third deliverable. Google Cloud shows the real-time costs. While experimenting, the Google Cloud engine charged us $7.00 for creating and deleting instances, having multiple instances up for a long period of time, and periodically testing out the server with numerous players. The charge amount depends on the hardware that would be used, how many people are utilizing the server, and how long the instances would be running. Our project cost analyzer predicted that it would be $250.00 per month to continue our project, but we were doing lots of adjustments to our Minecraft server as well as experimenting with other instances within Google Compute Engine. We averaged that a basic running of our Minecraft server would be about $35.00 per month realistically.

In addition to why we switched from CloudLab/OpenStack to Google Cloud, we were also asked about why we didn’t turn to Minecraft Realm. At the beginning of the creation of our server, we did not look into this option. Generally, it is more beneficial to host your own Minecraft server because you have more freedom to control how you want the game to function. The costs of Minecraft Realm would be cheaper than the cloud option because it would instead be hosted locally. It is the creator’s preference of whether or not they want the freedom to customize their server than to pay less. Our group would have chosen to have more freedom with our game even if we did consider using Minecraft Realm in the beginning. However, you do have more complications with hosting the server like moderating and administering it.

A benefit of administering your own Minecraft server is that you are able to share it with whomever you want to. It was easy for others to use our Minecraft server because all the player would have to do is connect through Minecraft multiplayer mode via our server IP. As previously stated, the game version would have to be the same as our server.

There are changes that could be made to our Minecraft game like installing plug-ins or modifications. Modifications can be made by installing Forge 8nstalled on the server and the client’s computer. Google Cloud makes modifications easier because we can update the server with the changed modifications in the instance. As well as modifications, plug-ins are also incredibly easy to install. Bukkit can allow for more console commands. DigitalOcean can be used for small projects and only charge $5.00 per month.

Other games can also be created from our initial project idea. We had a lot of comments that students wanted to do similar games on Google Cloud. Any multiplayer game (Counter Strike, Garry’s Mod, Arma, Rust, Stardew Valley, and Terraria) can go through the same process and have a server created.

One of the reasons we felt like the third deliverable was achieved was because it opened doors for other players to be curious about having their own favorite games turned into personal servers. It was great seeing how our project made other students feel like the cloud interface for games can be useful to them and they can use it in their free time. Cloud services can be used for real-world purposes and be extremely useful for the average person. It takes out the hardship of having the hardware infrastructure.

We felt that our project had reached deliverable three because it achieved the goal of a Minecraft server running and being functional for other players to use it. The goals of our deliverable three were to improve the server that we created. We were able to create a script that would make the process of our project easier to recreate. This was our top priority because the core of our project was to have the Minecraft server functioning and creating a script was the most direct way to improve that process.

Our stress test was to ensure that our server would not crash while it was running. This was the second most important task for us because we wanted our server to be running without us having to worry about it crashing. The stability of our game was important to us and for the players that were using our server.

The questions and comments from our deliverable two helped us think about our deliverable three because we were thinking more long-term about our project. It gave us perspective on how much money it would cost us to keep the project running, future projects with other games, and how much we had learned about Google Cloud. Google Cloud came with its own challenges because it was new to us. We did not learn about this from any other previous classes. After we navigated through Google Cloud, we were able to make changes for the new script and make any alterations to the server if necessary.

We have definitely met our goal that we set out to achieve from the beginning of the project and from the Second Deliverable.