CS 385: Tap-Tap-Adventure

Devin Brown, Eric Green, Justin Ramos

Overview

Tap-Tap-Adventure is an online 2D MMO set in medieval times. You can play the game through a web browser or using the iOS or Android apps. The project uses HTML5 WebSockets and node.js to provide the user with a smooth and enjoyable experience with the game.

The services that our project will be expanding on are the database end and bringing up more instances of the application itself to allow for greater numbers of players to play the game. The Google Cloud Platform and its various services were selected to help fulfill these goals. Google Spanner is the service of choice for scaling the database system. To bring up more instances of the web application, docker will be used as well as provisioning more virtual machine instances on the Google Compute Engine if needed.

The Database - Google Spanner

Several options were looked at in regards to providing a scalable database. Tap-Tap-Adventure is designed to work with a single MySQL database. Because of this, options were initially explored in setting up a MySQL Cluster. Both this option and others, such as using Google's CloudSQL, ended up being very complex to implement. To try and solve the problem of scaling the database, we looked towards using an existing solution: Google Spanner. Google Spanner is Google's globally-distributed strongly consistent relational database service designed for horizontal scaling. Using Google Spanner instead of MySQL for this project involved converting all queries from MySQL syntax to Google Spanner syntax. It also meant that all connections to the database had to be re-done as well, as interacting with a Google Spanner databases is more akin to interacting with a REST API. Work on this part halted when we encountered issues writing to the spanner database as well as realizing that the MySQL integration was deeper than we had originally thought.

Advantages

- Very easy to perform reads/(some writes): Uses a REST API.
- Very easy to scale up and down: Also uses a REST API.
- Everything is a REST API.
- All of the back-end management of active read/write nodes is handled by Google.

Disadvantages

- Turns out writes aren’t as easy as we’d hoped.
- ***EXTREMELY*** expensive. Once created, Spanner instances are continuously running until they are deleted. At a cost of $0.90/hour for a single node, this cost quickly adds up. A lot. $670/month minimum lot. This made testing difficult.

The Web Application - Docker/Google Compute Engine

The initial idea for scaling the web application part of the project was to create more docker instances of the web application and/or more Virtual Machine instances on the Google Compute Engine with docker contains as needed. The game world only accepts a maximum amount of players. If a world fills up, we can look at our metrics and decide to either create another docker container or spin up a new VM and deploy more docker containers there. Connections to the game world would first be load balanced between VMs and then within the VM the connection would be load balanced again to an available docker container. Due to issues in other parts of the project, the scaling of the web application had minimal work completed for it and thus the web application does not currently scale.

Advantages

- Very easy to deploy new containers or instances using docker.

Disadvantages

- Having multiple levels of load balancers that need to be managed.

Miscellaneous Services

To help facilitate the automatic scaling of Google Spanner and the web application, several helper services were used to monitor and perform actions as necessary. All of the following were deployed in docker containers (except Stackdriver).

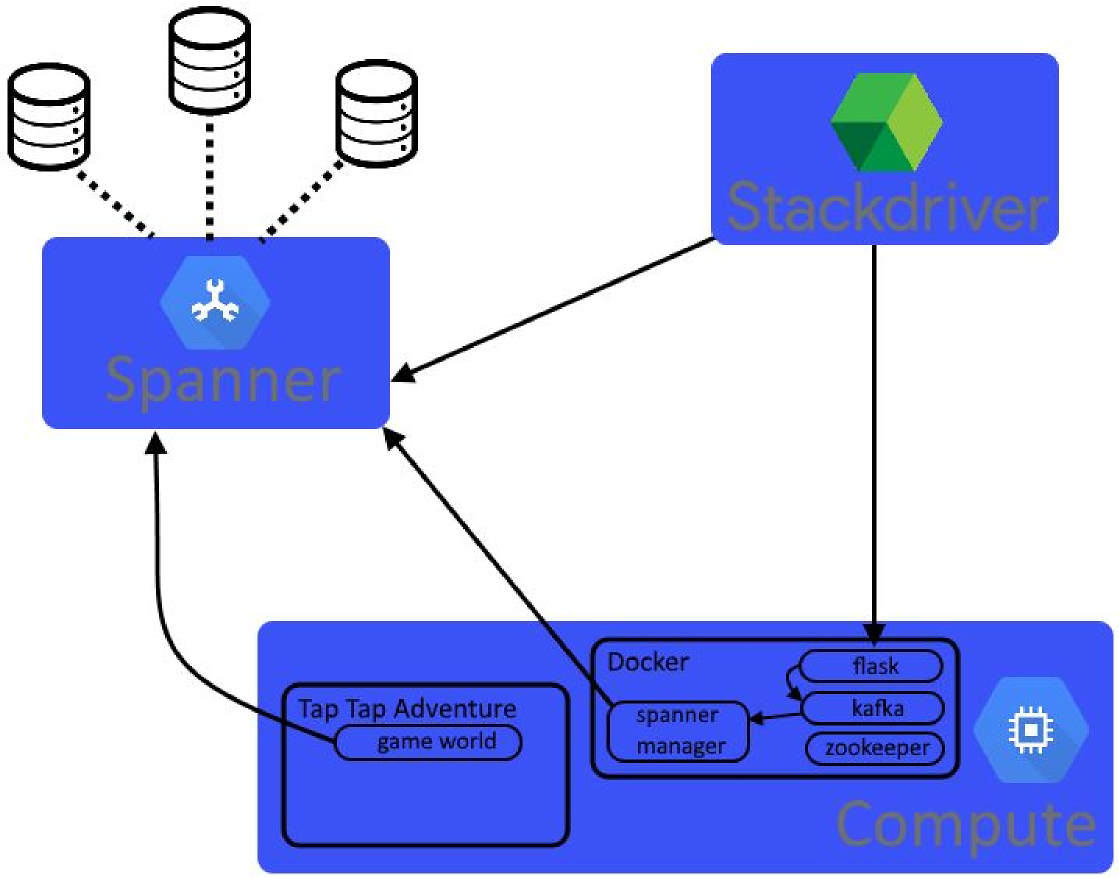
1. Google Stackdriver
2. Zookeeper/Apache Kafka
3. Flask app running small REST API; Kafka Producer

4. Python script for scaling the Google Spanner database; Kafka Consumer

Google Stackdriver was used to monitor the health of the Google Spanner database. It will monitor metrics and send an alert to the Flask app if the database has been at a high load for an extended period of time. In this instance, the flask app will post to an Apache Kafka topic a message that simply says “scale up” (or down if load decreases). The final python script is a kafka consumer and will be waiting for these messages to determine if it should allocate or deallocate a node for Google Spanner.

The reason to separate the final python script from the rest is that interacting with the spanner database will lock up the final script until the operation finishes. The intent was to have multiple topics and multiple consumers, one for scaling the spanner database and another for the web application.

Below is a diagram of what the current application structure looks like.



Retrospective

Because the web application itself did not fully work due to issues in figuring out how to use Google Spanner, there is not much demonstration for the project. Testing the webhooks for the stackdriver alerts did produce the desired scaling on the spanner instance nodes. Given what we know now from having worked with the services that we did, a few conclusions come to mind:

- Don’t use Google Spanner: Too expensive. The amount of testing that could be done was limited because of how much it cost to run the instance.
- Try to use a NoSQL system instead. Would be much easier to scale up/down. The data stored for this game is not relational. We only used a relational database because the game already used one.
- Stackdriver webhooks aren’t the best way to receive metrics on the system performance. Perhaps try to find a better alternative.