Technology Research Document

Name: Tony Jiang

Semester: 6

Class: RB04

Contents

[Introduction 3](#_Toc181539620)

[Technology for backend (individual) 3](#_Toc181539621)

[Database selection (individual) 3](#_Toc181539622)

[CAP theorem 3](#_Toc181539623)

[Polyglot persistence 4](#_Toc181539624)

[Text file store (group) 5](#_Toc181539625)

[Cloud technology (individual) 6](#_Toc181539626)

# Introduction

This document contains all the research I’ve done on the technology and methods I use, along with the reasons for my choices. This way, I can showcase to the teachers that I have explored the options and justified my decisions. This research can be for the individual project or the group project.

# Technology for backend (individual)

This semester, I have the opportunity to explore a new programming language. A friend recommended that I try Golang, so I did some research. Initially, I was planning to use Java because of its vast library options, built-in security features, high scalability, and strong support for microservices.

However, Golang (Go), which was designed by Google, also offers several appealing features. It has a simple and easy-to-read design, fast compilation, strong support for microservices, and is excellent for cloud-native development. Additionally, it can easily scale horizontally and is highly favored for DevOps and infrastructure tasks. The only problem is that I need to learn how to code in Golang.

The obvious choice is Golang because it adheres to all the learning outcome of this semester.

# Database selection (individual)

Choosing the right database for a project can be challenging. To make this decision, I use the CAP theorem and it is reference Canvas.

## CAP theorem

The CAP theorem is a fundamental concept in distributed database systems. It states that it is impossible for a distributed data store to simultaneously guarantee all three of the following properties:

* Consistency (C): Every read receives the most recent, correct, and up-to-date information, regardless of which server (or node) you ask.
* Availability (A): Every request (read or write) receives a response, even if it doesn't guarantee that the response contains the most recent write.
* Partition Tolerance (P): The system remains operational despite network partitions or communication breakdowns between nodes.

According to the CAP theorem, you can only guarantee two out of these three:

* CP (Consistency + Partition Tolerance): The system remains consistent and tolerates network partitions, but this comes at the cost of availability—some requests may fail.
* AP (Availability + Partition Tolerance): The system prioritizes responsiveness and handles network issues, but consistency may be sacrificed, meaning you might not always receive the latest data.
* CA (Consistency + Availability): The system ensures both accurate, up-to-date data and immediate responses, but it cannot tolerate network failures or partitions.

A diagram of a system

Description automatically generated

Based on the CAP theorem, my project should prioritize AP (Availability + Partition Tolerance) to ensure quick response times and resilience to network issues. My users prefer faster responses over strict data consistency since they are playing a game, and they value immediate feedback on their answers more than having perfectly consistent data. Therefore, my system will need an AP-focused database. A reference to AP databases is provided in the image above.

## Polyglot persistence

In microservices architecture, it's common to use Polyglot Persistence, where each microservice can use the database best suited to its specific needs. You don't need to rely on a single type of database for the entire system. Different microservices can use different databases based on their requirements.

# Text file store (group)

I was tasked with creating a text file storage solution for the botlist service for the group. Since I had no prior knowledge or experience in this area, I conducted some research on how to implement it. The language used to create the botlist service is Java, and the text file can be stored in JSON format.

I explored possible solutions for storing data in a text file using Java and identified two dependencies that could meet my requirements:

the Gson library and the Jackson library.

I chose the Jackson library due to its advanced features, better performance, and support for polymorphism. This choice also allows for future scalability, ensuring high performance and accommodating larger datasets if the company decides to expand.

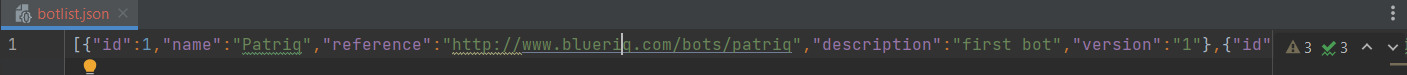
While the Gson library is simpler and more beginner-friendly, I opted for Jackson to better suit the project’s needs. All you need to do is add the Jackson dependency in build.gradle, and you can start coding.A screen shot of a computer

Description automatically generated

The text file is stored in the root folder of the botlist service. If the file does not exist, the system will automatically create it when a botlist is added. However, the file does not track IDs, as it lacks an auto-increment mechanism like a database, so an auto-increment feature will need to be implemented manually.

A screenshot of a computer

Description automatically generated



# Cloud technology (individual)

For the cloud technology that I’m going to use, I’m considering either Kubernetes or cloud functions. I have more experience with Kubernetes than with cloud functions. Cloud functions are new to me; I understand that, with cloud functions, instead of deploying a full microservice to the cloud, you can deploy individual functions in a serverless manner. For testing purposes, I’m going to use cloud functions to compare them with Kubernetes.

## Reason for deploying to cloud

I am using the cloud for my project because it allows me to scale my application depending on the number of users. My application is a music guessing game, where users challenge themselves to guess the daily song or guess random songs. Since the application should handle a large number of users, deploying it to the cloud is ideal for scaling the project.

## Cloud function

The main difference between Kubernetes and cloud functions is that with Kubernetes, you deploy and manage entire services (applications or containers) on the cloud. This often requires setting up and maintaining infrastructure for scaling, availability, and load balancing, which can lead to higher costs. The cost of hosting a service on Kubernetes starts as soon as you deploy the service, and it accumulates based on how long your containers or pods are running.

On the other hand, cloud functions are serverless, meaning you only deploy individual functions (specific pieces of code that perform tasks). These functions are event-driven and only run when triggered, such as when a user interacts with your app or a specific event occurs. The cost is based on the number of invocations (triggers) and the execution time of the function, not on continuous uptime like Kubernetes. This can make cloud functions more cost-effective for smaller, event-driven tasks since you only pay for what you use.

## Cloud provider

For the cloud provider I can use Google, AWS or Azure. Azure is out of the question because I’m using Golang, I have to configure a lot of things for Golang. Google and AWS is ideal because of their ease of use, extensive support, and ecosystem integrations.

### Pricing

Google offers a free tier with $300 in credits valid for 90 days. After that, you are billed for usage. The free tier includes 180,000 vCPU-seconds and 360,000 GiB-seconds per month. Google bills execution time in vCPU-seconds and GiB-seconds. Beyond the free tier, the rates are $0.000018 per vCPU-second and $0.000002 per GiB-second. These rates can get more cost-effective with committed-use discounts for long-term, high-volume usage.

AWS provides a free tier with 7.5 billion GB-seconds per month, which is generous and suitable for low-cost startups. After exceeding the free tier, AWS bills based on GB-seconds (a combination of memory allocated and execution time). AWS charges $0.0000133334 per GB-second initially, with rates decreasing at higher usage volumes.

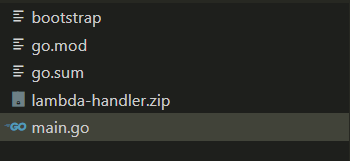
For testing purposes I’m using AWS, AWS is ideal due to its generous free tier. However, for long-term, high-volume scenarios, Google can be more cost-effective, especially with committed-use discounts.

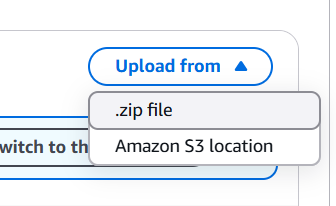
## How I did it

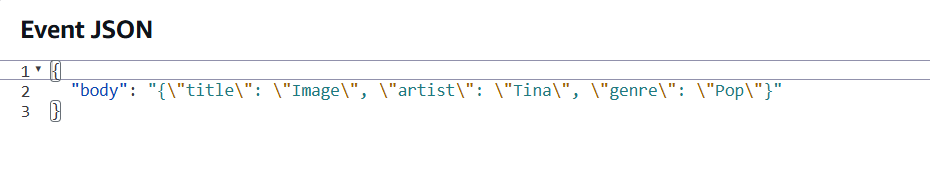
To deploy a Golang function to AWS Lambda, I started by importing the AWS Lambda Go SDK into my project using the package "github.com/aws/aws-lambda-go". I then created a handler for Lambda, utilizing my CreateSong function as an example. To connect the handler to the Lambda runtime, I added it to the lambda.Start() function, ensuring that the code was ready for deployment.A screen shot of a computer

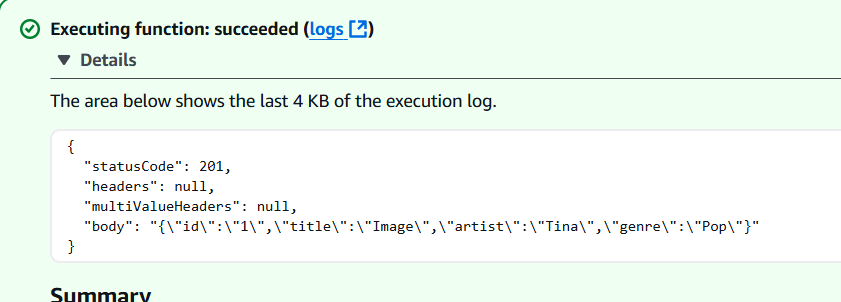
Description automatically generated

Next, I built a binary file from my main.go file. Since AWS Lambda requires the binary to be named bootstrap, I made sure to name it accordingly. I used the following command to create the binary: $env:GOOS="linux"; $env:GOARCH="arm64"; $env:CGO\_ENABLED="0"; go build -o bootstrap main.go. This step ensured that the binary was compatible with Lambda’s runtime environment. Additionally, I configured my environment file to match the environment instance I created in the Lambda UI. After creating the binary, I zipped it into a .zip archive and uploaded it to AWS Lambda.

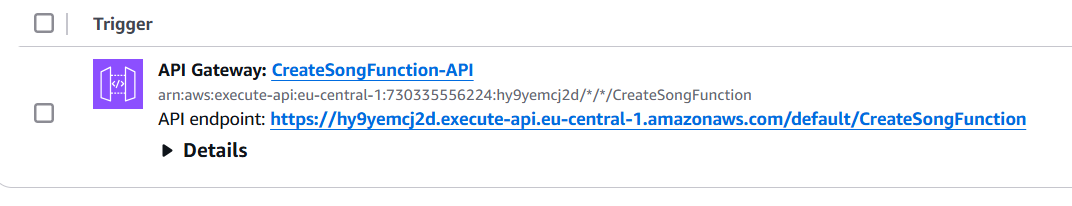


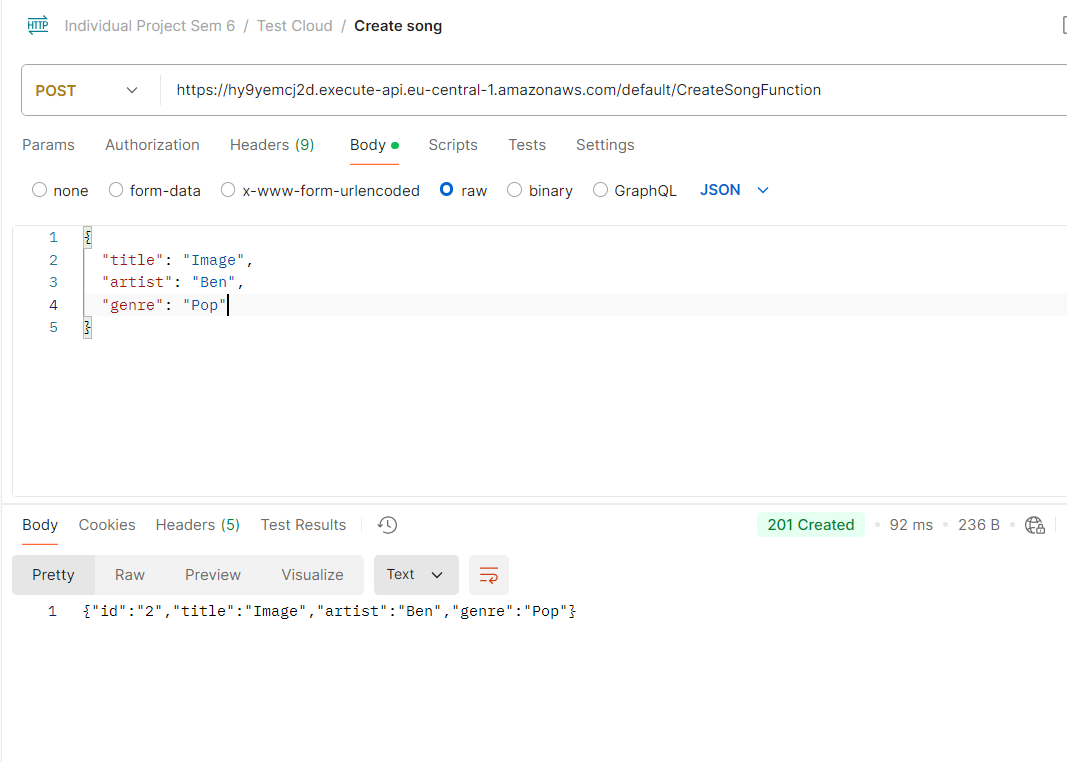
In the Lambda UI, I used the “Upload from” option and selected the .zip file for upload. 

Once the file was uploaded, I tested the function within the Lambda UI. I noticed that the request body format for Lambda tests differed slightly from Postman, so I adjusted the test input accordingly. The function ran successfully, and I received a response confirming its execution.



To make the function accessible, I added an API Gateway as a trigger. This setup generated a URL for the function, allowing me to send requests. In this case, I configured the API Gateway to handle POST requests. Finally, I tested the URL in Postman to verify that the function was triggered correctly and performed as expected.





That’s the process I followed to deploy a Golang function to AWS Lambda. By using the Lambda UI and some manual steps, I successfully deployed, tested, and triggered the function.

Summary of Deploying to AWS Lambda

To deploy a function to AWS Lambda, start by importing "github.com/aws/aws-lambda-go" into your project. Create a handler function for Lambda and pass it to the lambda.Start function from the AWS Lambda package. Next, build a binary file named bootstrap from your main.go file. Once the binary is created, zip it and upload the zip file to Lambda via the AWS Lambda UI. Test the function to ensure it works correctly. After successful testing, add an API Gateway to trigger the function. Adding the API Gateway will generate a URL that you can use to invoke your deployed Lambda function.