Capstone Project Report

Paulo Bolinhas - 110976

Rui Martins - 110890

Management and Administration of IT Infrastructures and Services

Introduction

The project's objective is to implement a simple and functional web application based on microservices. This type of architecture allows each service to manage its own distinct business logic while maintaining flexibility and scalability. The solution must be deployed on a cloud platform, ensuring it's easily reproducible. The project is divided into two stages, the first involves creating a basic functional system, while the second focuses on adding advanced features to enhance the solution.

That said, Bank46 is our cloud-powered banking web application, entailing the use of automation tools, designed to manage core microservices such as Insurance, Loan, and Transaction processing.

Video link: https://youtu.be/wXeHUM6cIso

System Design and Architecture

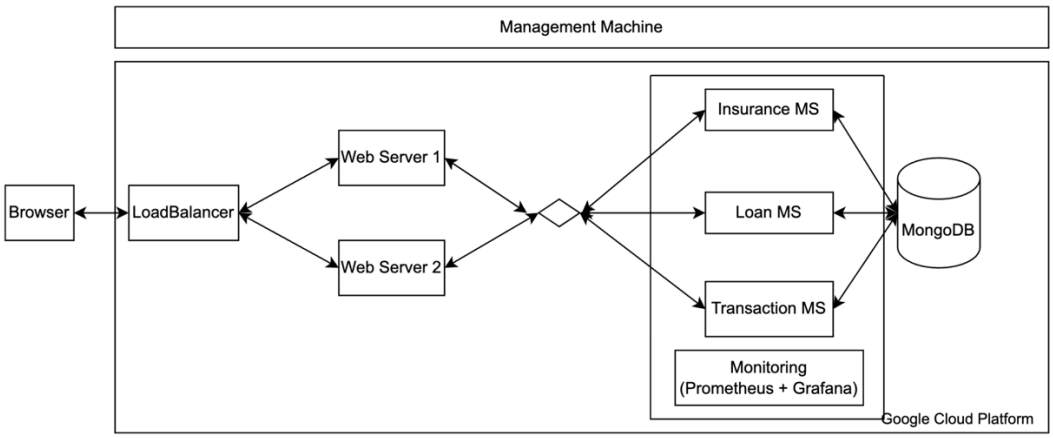

Image 1: Bank46 Architecture

The architecture of Bank46 follows a web microservices-based design **(Step 1 - Building a Web Application)**, hosted on GCP **(Step 2 - Deploying Foundational Resources)**, managed through a combination of tools including Vagrant, Terraform and Ansible **(Step 3 - Infrastructure as Code (IaC))**, data stored with MongoDB **(Advanced Component #2)** and monitored using Prometheus and Grafana **(Step 4 - Operations Tooling for Monitoring)**.

Vagrant is used to create a virtual management machine (mgmt), where Terraform initializes and deploys all the others, including the web servers, microservices, balancer, monitors and the database within GCP. Ansible then ensures the proper configuration of these machines running the respective playbooks. For monitoring, Prometheus and Grafana are used to track the system health and performance metrics.

The frontend is handled by 2 web servers, which load the microservices JavaScript frontend templates to the browser. A load balancer **(Advanced Component #2)** manages traffic distribution across these web servers, optimizing performance. It is through the ip of this balancer, that, depending on the balancing, the main application html page is loaded by the chosen web service. On this page, it is possible to interact with all the application endpoints, calling for the microservices backend logic.

The backend consists of three key microservices: Insurance, Loan, and Transaction. These are responsible for all the browser endpoints business logic (JavaScript) and each of them communicate with the MongoDB instance (Advanced Component #2) for proper data storage.

Every microservice have an endpoint for metrics exposure. With the Grafana + Prometheus setup, all the endpoints are easily tracked and monitored.

Our system is fully automated, since we use a unique ansible file that runs everything. This playbook initializes terraform, put the ips in the inventory file and in the environment of the machines (environment manipulation) so everything runs smoothly, and runs all the other configuration playbooks. Each playbook is essentially to install the necessary packages and assure services to run.

Deployment Method

(Advanced Component #2)
Bank46's deployment follows advanced practices to ensure high availability.

Part I – Load Balancing

Traffic is managed using load balancing across both web servers and microservices, implemented with HAProxy using the roundrobin mechanism. This is set up via an Ansible playbook on the load balancer machine, which is deployed using Terraform, that calls the haproxy config file. The load balancer distributes requests efficiently, ensuring optimal performance and availability.

```
#---------------------------------------------------------------------
# FrontEnd Configuration
#---------------------------------------------------------------------
frontend hafrontend
    bind *:80
    mode http

    default_backend web_backend

    # Monitoring Config URI
    stats enable
    stats uri /haproxy?stats
```

```
#---------------------------------------------------------------------
# BackEnd Configuration for Web
#---------------------------------------------------------------------
backend web_backend
  mode http
  balance roundrobin
  {% for host in groups['web'] %}
      server {{ host }} {{ hostvars[host]['ansible_default_ipv4']['address'] }}:80 check
  {% endfor %}
```

Image 2: Web Servers Balancing

Part II - Database backend service with persistent storage

A database backend service with persistent storage is employed to store data. Terraform deploys a MongoDB instance, which is then configured via an Ansible playbook, where it, by default, inserts mock values into the db instance. Each microservice connects to the database using individual Mongoose connections and schemas, since the business logics are different. This ensures that all endpoints are interrelated and can interact with the database.

```
async function connectToMongo() {
    try {
        mongoose.set('debug', true);
        await mongoose.connect(mongoURI);
        console.log('MongoDB connected');

        const db = mongoose.connection;

        db.once('open', async () => {
            console.log('Connection to DB is open');

            const collections = await db.db.listCollections().toArray();
            console.log('Collections in the database:', collections);
        });
    } catch (err) {
        console.error('MongoDB connection error:', err);
    }
}
```

```
const mongoose = require('mongoose');

const insuranceSchema = new mongoose.Schema({
    owner_id: String,
    title: String,
    monthly_payment: Number,
    currency: String
}, { collection: 'insurance' });

const Insurance = mongoose.model('Insurance', insuranceSchema);

module.exports = Insurance;
```

Image 3: Insurance ms & MongoDB instance connection and schema

Image 4: Insurance mock values

Evaluation

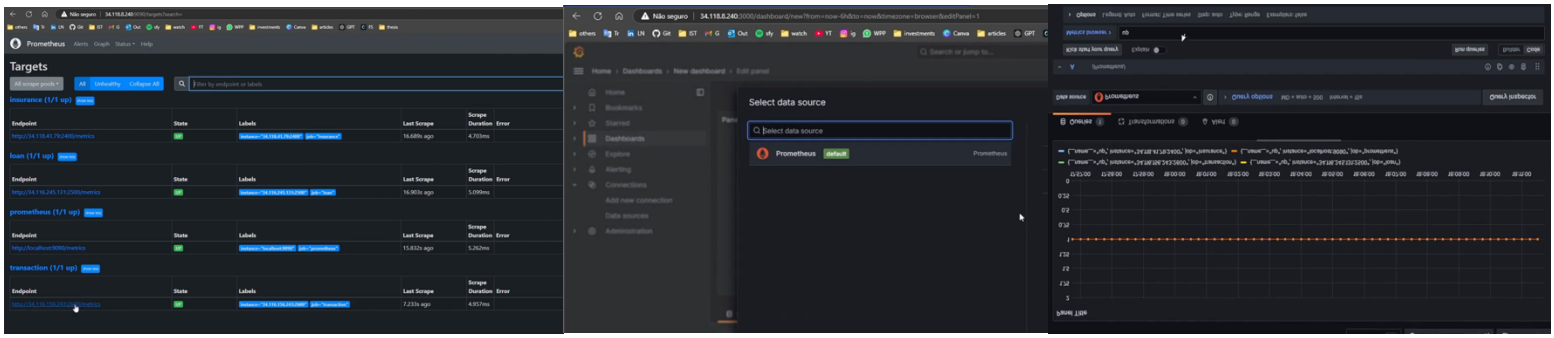

Image 5: Prometheus + Grafana

Microservices (Insurance, Loan, Transaction) are each assigned unique endpoints and ports, and Prometheus regularly scrapes them to collect metrics on requests, response times, and other performance indicators. Prometheus monitoring is integrated as part of the operations tooling (Step 4), working alongside Grafana for real-time visualization and alerting, as Data Source. The “Up” State, on both tools shows all services are online and accessible, with no errors in scraping, indicating stable performance at this monitoring point.

Throughput

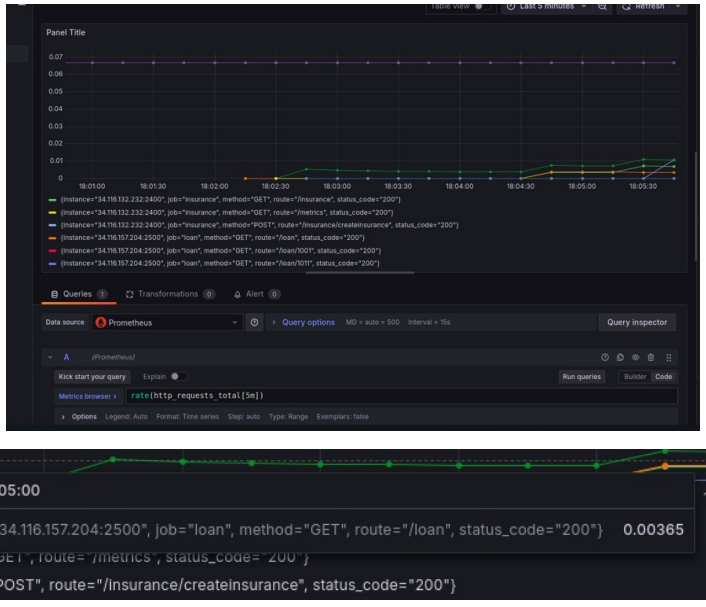

Image 6: Throughput analysis

This chart shows HTTP request rates for our microservices over a 5-minute window. The steady request rates across various routes (e.g., /insurance, /loan/1001) indicate that the microservices are receiving consistent traffic, with no noticeable spikes or drops in throughput, and all responses have a status code of 200 (success).

For example, the highlighted section shows that at 18:05:00, the "loan" microservice (GET request to the /loan route) processed a request successfully with a status code of 200, and the throughput at this moment was 0.00365 requests per second.

Latency

Image 7: Latency analysis

If the throughput (request rate) is consistently low (e.g., around 0.01 requests per second), then assuming the response time remains constant, we estimate latency indirectly. For example, if the request rate is 0.01 requests/sec, and each request completes in that interval, the inferred latency would be the reciprocal, approximately 100 ms/request.

Conclusion

In building and deploying Bank46, we've applied class knowledge in cloud infrastructure, microservices and automation to create a robust banking platform. The architecture is fully modular, with business logic broken down into microservices for Insurance, Loan, and Transaction, ensuring that each service can be developed and deployed independently. By using Vagrant for managing the development environment, Terraform for deploying resources on GCP and Ansible for configuration management, we've ensured that the entire system is automated and easy to manage.

The load balancing across the web servers ensures high availability and optimized traffic handling. The choice of MongoDB as a database backend provides persistent storage, ensuring that data durability and availability are never compromised.

To keep everything running smoothly, we've integrated Prometheus and Grafana to monitor system performance, offering insights into throughput, latency and overall health metrics in real-time.

Ultimately, the combination of these tools and design choices allows Bank46 to handle increased traffic efficiently, maintain high availability and provide a smooth user experience - all while being highly easy to manage.