Cloud computing

Fábio Gaspar, N.º 2220660

Polytechnic of Leiria

Wednesday 26th April, 2023

Contents

1 Introduction

Microservices architecture is a contemporary software development approach that entails decomposing a monolithic system into autonomous, smaller components that can be created, deployed and maintained separately. In contrast to a monolithic architecture, where all the functionality of the system is consolidated into a single codebase, microservices enable developers to work on distinct features or services independently without affecting other parts of the system. The goal of this project is to demonstrate the importance of decoupling services, as well as the use of consistent implementation strategies to achieve scalability, flexibility, and maintainability, by deploying and provisioning an application both locally and on a cloud platform.

2 Services and Architecture

The foundation of this project consists of a TicTacToe Multiplayer Game as a monolithic service with two components Figure 1. The first component is an HTTP Server that enables clients to access the game, serving a standard web application (HTML, CSS, and JavaScript files) being hosted with an express web server, while the second is a WebSocket server to handle all the game logic, room, and rules and using the Socket.IO library

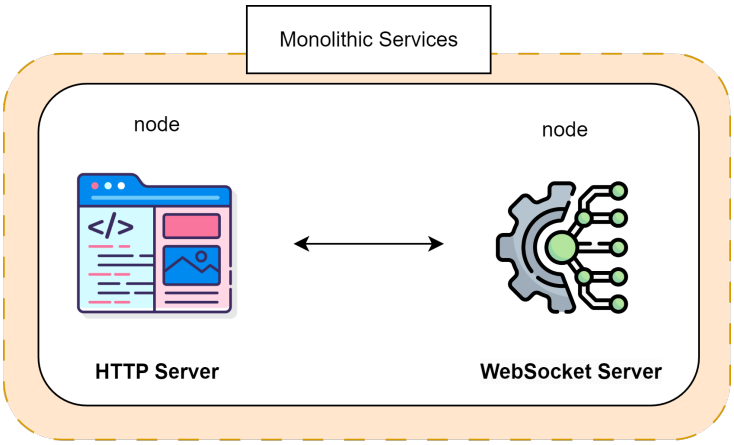

Figure 1: Original Monolithic implementation of Multiplayer TicTacToe

2.1 Local Architecture

To create these distinct services, it became essential to decouple the original monolithic implementation. The WebSockets component had several dependencies on the website, which made it difficult to separate the two services. Nevertheless, with some modifications, it was possible to retain the majority of the original implementation since in this type of work, it's crucial to preserve as much of the existing implementation as possible. In Figure 2 is represented the proposed architecture with the newly implemented services:

2.1.1 HTTP server

The HTTP Server is based on Node.js and uses the Express.js library in order to handle web application requests. When a user accesses the HTTP Server to play a game, the server retrieves

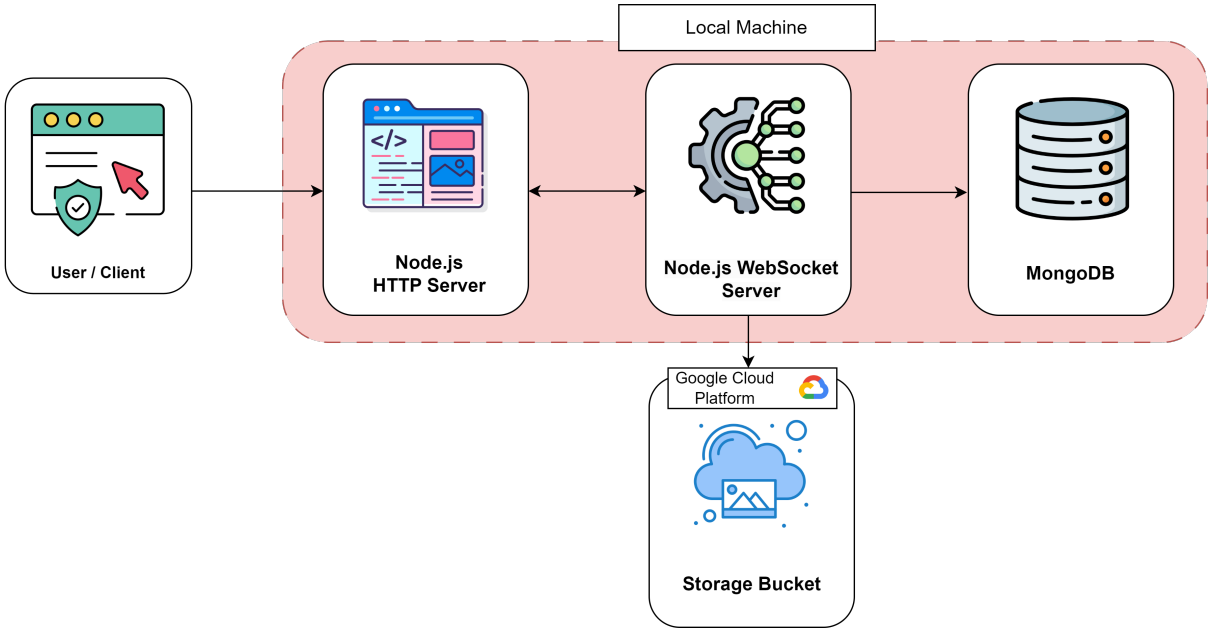

Figure 2: Proposed Local Architecture

an interactive website for the user and takes care of incoming requests between the user's browser and the application's back-end, where the game logic resides.

2.1.2 WebSocket server

The WebSocket server is where most platform logic resides. The service is created using a Node.js image and it uses the Socket.IO module to handle real-time communication between the client and server, handling the majority of the features.

The most important features of this WebSocket server are:

- **Room management:** The server has the ability to create two different types of game rooms: random player rooms and private rooms. Each room is assigned a unique room ID by the server, and players have the flexibility to join or leave a room at any point in time. It is also responsible for managing its players assigning them a player number and a letter (X or O) and also keeping track of the player's turn and the room information.

- **Game logic:** handles the game logic, including checking for win conditions, determining the next player's turn, and randomization of which player starts the game each time.

- **Score history:** The server is a crucial component in upholding the integrity of every game played since it performs the task of sending all game-related data to the cloud bucket and database, ensuring that the game's historical information is preserved and safeguarded.

2.1.3 Database

A new MongoDB database service was implemented to save the Win/Lose games data. The database records key information such as the winning player's name, the final game board state (represented in an image format), and the time when the game occurred. This data is seamlessly integrated into our platform and is readily available to users who are waiting to join a room.

2.1.4 Cloud Storage Bucket

A storage bucket is essentially a container for storing digital data in the cloud. It is a scalable and flexible solution that allows users to store large amounts of data. Creating a local storage system that could match the capabilities and convenience of cloud-based storage would require a significant amount of effort so since it is not the real objective of this project this was not developed. The integration of this new service is used for saving images of the game board after every game. This allows players to access and share their game board results with their friends by simply sharing a link.

2.2 Cloud Architecture

Cloud architecture (Figure 3) is similar to local architecture in many ways, but there are some important differences. One of the key differences between cloud and local architecture is that in cloud architecture, services are typically deployed remotely rather than on local servers resulting in some implementation differences, which will be covered in greater detail in the provisioning section. For the cloud computing platform, there are several options available, such as Azure, AWS, and Google Cloud.

After doing some research and evaluating each platform's features, I found that all three had extensive documentation and resources available, making it easy to get started and learn how to use each platform. However, I opted to use Google Cloud to host the client and server application as well as the storage bucket. This decision was taken since we had already been using it in our classes, which made the transition to using it easier, and also since it offers good pricing models. To achieve the goal of multiple platforms, I also have chosen to use Atlas MongoDB as my database service, hosted on Amazon Web Services (AWS).

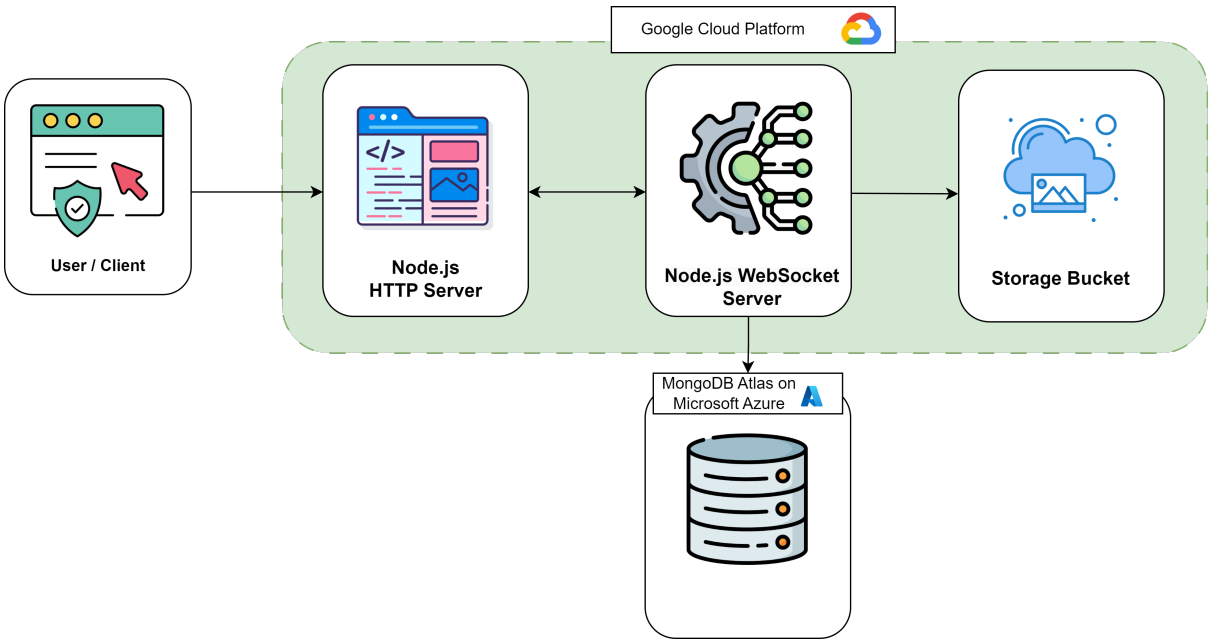

Figure 3: Proposed Cloud Architecture

3 Provisioning

In this section, we will delve into the infrastructures for the various services implemented on the architecture, discussing both cloud and local provisioning and also the reasoning behind these choices.

3.1 Local Provisioning

The local infrastructure provisioning Figure 4 is processed using Docker, with each service having its own Dockerfile to enable a streamlined deployment process. Additionally, a Docker Compose YAML was created to further simplify the process facilitating the containerization and seamless deployment of each service, while also providing the flexibility to scale and manage the infrastructure as needed.

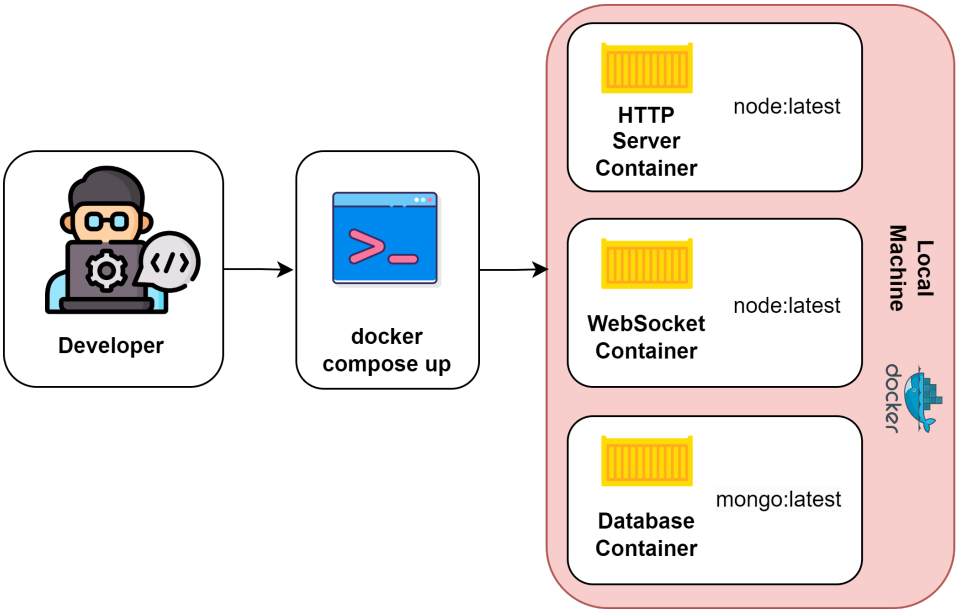

Figure 4: Local provisioning architecture

In this particular case, the process of containerization was carried out using an initial image of Node.js for both the HTTP Server and the WebSocket server, as well as a Mongo image for the database. When deploying the app locally (using docker-compose) a specific bucket was created solely for developing the application but its provisioning was done manually on the Google Cloud Platform since the bucket doesn't need to be accessed every time, only when deploying for testing, and allowing too for greater control over the setup and configuration of this.

3.2 Cloud Provisioning

To ensure seamless cloud provisioning of the platform, it is essential to have an efficient and connected infrastructure. The implemented infrastructure can be visualized on the Figure 5

To facilitate the deployment of the services, it was implemented a Terraform infrastructure which enables the automatic deployment of the image server, and cloud server with the updated images. In regard to having the most recent images used to deploy the application, a trigger was created in the Google Cloud Platform using the Cloud Build.

The trigger automatically allocates an image of each service in the Google Container Registry whenever a new push occurs in the master branch.

When destroying the infrastructure (”terraform destroy”), it was decided to clean the bucket every time and create it automatically whenever a new version is deployed (”terraform apply”). If we would like to maintain the data from the bucket when deploying a new version, we can modify the Terraform infrastructure by setting the ”force destroy” option to false, which would prevent the bucket from being deleted every time the application is deployed and make its creation only if it this doesn't exist. For storing the state information created by Terraform it was chosen to save it in the Terraform Cloud rather than locally providing better collaboration, ease of use, and backup/recovery options for infrastructure configuration.

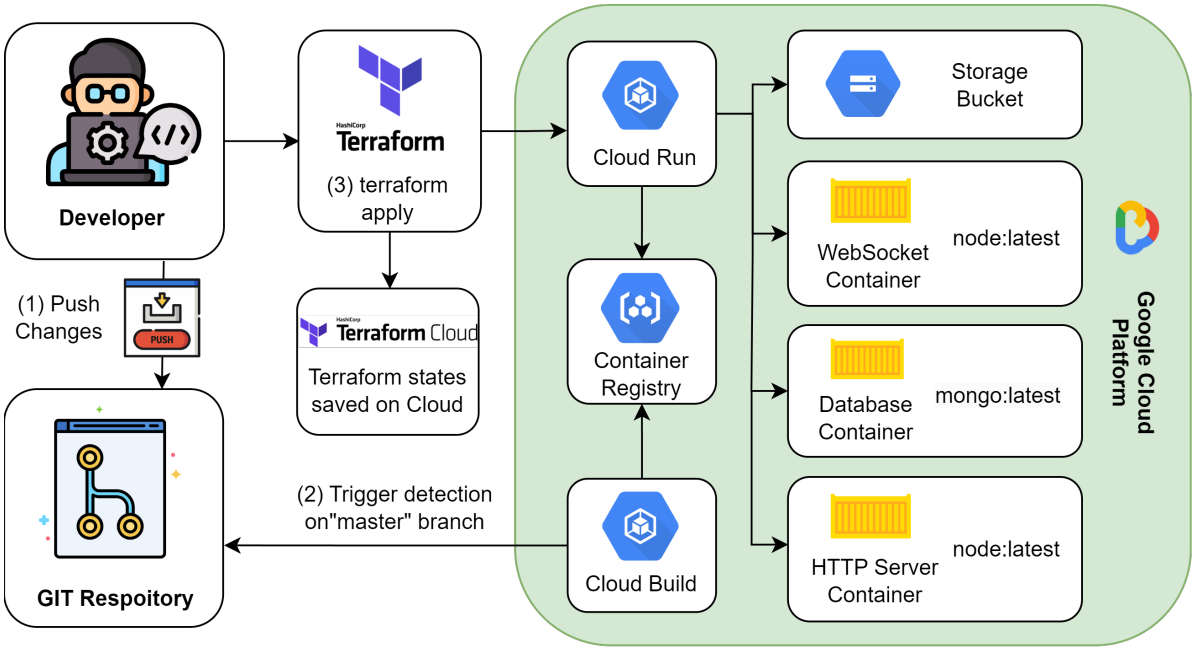

Figure 5: Cloud provisioning architecture

4 Conclusion

In conclusion, this project aimed to demonstrate the benefits of the microservices architecture approach to software development. By decomposing a monolithic system into smaller components, it was possible to achieve greater scalability, flexibility, and maintainability. This project specifically focused on decoupling the multiplayer TicTacToe game from its original implementation and the addition of possible useful services to it.

The project helped to gain a better understanding of cloud computing and its importance in the modern technology landscape. Through this project, it is possible to start understanding and discovering the agility, scalability, and cost-effectiveness that cloud computing offers, as well as the minimal management effort required to maintain infrastructures. The ease of provisioning and scaling of microservices in cloud environments can help rapidly respond to changing business needs, making cloud computing an essential technology for businesses and individuals alike.