Decentralized File Exchange Hub: A Cloud-Native Approach

Rafael Pereira[a]

[a]Computer Science and Communications Research Centre, School of Technology and Management, Polytechnic of Leiria, 2411-901 Leiria, Portugal

ARTICLE INFO

Keywords:
Decentralized file exchange
Microservices architecture
Containerization
Cloud-native deployment
Real-time communication
Scalability and resilience

ABSTRACT

In this report, we present a decentralized file exchange hub designed using a cloud-native approach that leverages containers, microservices, and cloud services. Our architecture comprises a React-based frontend served by Nginx, an Express API for managing file transfers, a real-time communication server using Socket.IO, and MongoDB as the primary database. The application stores files in Google Cloud Storage, and our deployment pipeline is streamlined using Google Cloud Build and Cloud Run. The result is a scalable, resilient, and efficient file exchange system that demonstrates the power and flexibility of modern cloud computing technologies for software applications.

1. Introduction

The rapid growth of cloud computing and microservices has revolutionized the way applications are developed and deployed. As a result, there is an increasing need for effective strategies to manage the complexity and scalability of these systems. This report explores the design and deployment of a distributed cloud-based application that utilizes containerization and microservices to achieve high availability, performance, and maintainability.

The application is divided into four main modules: Presentation Provider (Nginx), API (Express), Asynchronous Message Communication server, and Database (MongoDB). Each module is designed to serve a specific purpose and can scale independently to adapt to changing loads and requirements. The application also leverages Google Cloud Platform (GCP) services such as Cloud Storage Bucket for storing and managing files.

The report is structured as follows: Section 2 presents the architecture of the application, detailing the purpose and design of each module. Section 3 describes the deployment process, emphasizing the use of GCP's Cloud Build and Cloud Run services to automate and streamline the deployment of the application components. Finally, the report concludes with a summary of the key findings and insights gained from the project.

2. Architecture

This project follows a microservices architecture, which allows for improved scalability, resilience, and maintainability. The system is divided into four main components, each responsible for a specific aspect of the application. Figure 1 presents a diagram of the architecture, illustrating the relationship between each component.

1. Presentation Provider (Nginx): This component acts as the frontend server, hosting the React web application and serving static files. It is responsible for delivering the user interface to the clients and handling incoming HTTP requests.
2. API (Express File Management Service): This component is the backend server implemented using Express.js. It manages the file uploading and downloading processes, interacting with the Database and Cloud Storage Bucket to store and retrieve files.
3. Asynchronous Message Communication Server (Socket.io): This server enables real-time communication between the clients, ensuring that users in the same room receive instant updates when a new file is shared.
4. Database (MongoDB): The database stores information about the rooms, users, and file metadata, including the bucket URL and timestamp. MongoDB was chosen for its flexibility, scalability, and performance.
5. Cloud Storage Bucket (Azure Blob Storage): This component stores the uploaded files, providing a reliable and scalable storage solution.

The following UML diagram (Figure 1) depicts the relationships between the components:

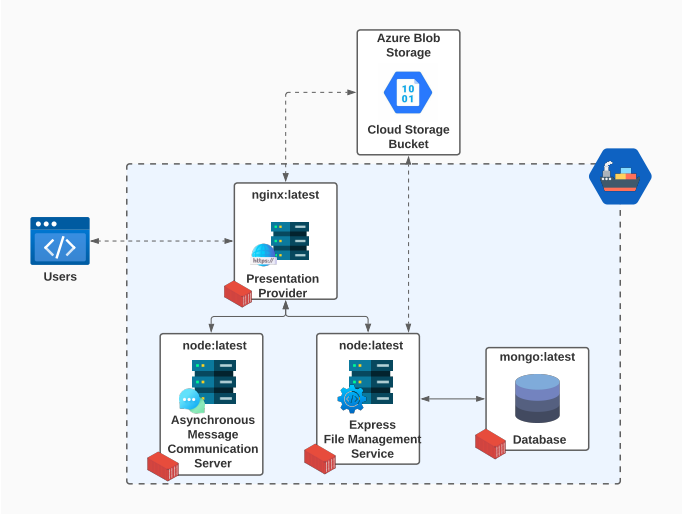

Figure 1: System architecture diagram

In this architecture, each component communicates with the others through well-defined interfaces, allowing for effi-

rafael.m.pereira@ipleiria.pt (R. Pereira)
ORCID(s): 0000-0001-8313-7253 (R. Pereira)
https://www.linkedin.com/profile/view?id='rafaelmendespereira' (R. Pereira)

cient collaboration and easy maintenance. The use of containerization and cloud-native deployment further enhances the system's scalability and resilience.

2.1. Presentation Provider: Nginx-based Frontend

The presentation layer of our file exchange hub is built using a React web application. We utilize Nginx as a reverse proxy and web server to serve the static content generated by the React application. Nginx offers excellent performance, stability, and low resource usage, making it an ideal choice for our frontend. Furthermore, its ability to handle a large number of simultaneous connections ensures a responsive user experience, even during peak loads.

2.2. API: Express-based File Transfer Service

Our file transfer service is implemented using Node.js with the Express framework, providing a robust and scalable API for managing file uploads and downloads. This service is responsible for handling file metadata, coordinating file transfers, and integrating with the cloud storage service for actual file storage. Express is chosen for its ease of use, extensive documentation, and compatibility with Node.js, which allows for efficient development and seamless integration with other components.

2.3. Asynchronous Message Communication Server: Socket.IO

To enable real-time communication and notifications within our file exchange hub, we employ Socket.IO, a popular library for real-time web applications. This server facilitates instant messaging between users in the same room, ensuring a seamless and interactive experience for all participants. Socket.IO is selected for its flexibility, reliability, and compatibility with various platforms and transports, making it a powerful solution for real-time communication in our application.

2.4. Database: MongoDB

MongoDB, a NoSQL database, is utilized to store information about rooms, files, and users. It's flexible schema and horizontal scalability makes it a fitting choice for our cloud-native application, as it can easily adapt to changing requirements and handle large volumes of data. Additionally, MongoDB provides a rich query language and high-performance indexing capabilities, allowing for efficient data retrieval and manipulation.

2.5. Cloud Storage Bucket: Azure Blob Storage

To store and manage the actual files exchanged by users, we leverage Azure Blob Storage. This fully managed object storage service offers high availability, durability, and scalability, allowing our file exchange hub to efficiently store, retrieve, and share files. The choice of Azure Blob Storage is driven by its integration with other Azure services, its security features, and its cost-effective pricing model, making it a suitable solution for our application's file storage needs.

3. Deployment

The deployment process is designed to ensure a smooth transition from development to production, leveraging the power of Google Cloud Platform (GCP) services. The deployment strategy uses GCP's Cloud Build and Cloud Run services for automating and streamlining the process. Figure 2 presents a diagram of the deployment process, illustrating the steps and relationships between each involved service.

1. Cloud Build: This service automates the build, test, and deployment of the application components. It is triggered when changes are pushed to the repository, and it builds the Docker images for the various services in the application.
2. Cloud Run: This service deploys the containerized applications on a fully managed platform, providing automatic scaling and load balancing. It ensures that the services are highly available and resilient.

The following UML diagram (Figure 2) depicts the deployment process and the relationships between the components:

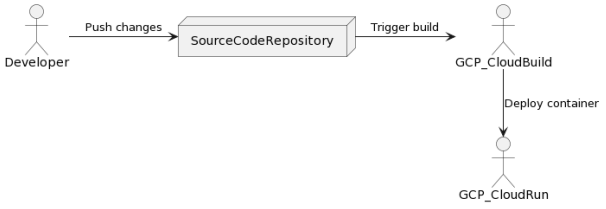

Figure 2: Deployment process diagram

The deployment process ensures that the application components are consistently built and deployed across environments, minimizing human error and facilitating continuous integration and delivery.

3.1. Cloud Build: Continuous Integration and Deployment

For a streamlined development and deployment process, we use Google Cloud Build, a managed service for building, testing, and deploying container images. This service automates the process of building Docker images from our source code and deploying them to Google Cloud Run, ensuring a consistent and reliable deployment pipeline.

3.2. Cloud Run: Scalable Containerized Services

Our application's microservices are containerized and deployed on Google Cloud Run, a serverless platform that runs Docker containers. This allows us to easily scale our services up or down based on demand, ensuring optimal resource usage and cost efficiency.

4. Conclusion

This report has presented a decentralized file exchange hub designed with a cloud-native approach. By leveraging

containers, microservices, and cloud services, we have created a scalable, resilient, and efficient system for exchanging files. Our use of modern technologies such as Nginx, Express, Socket.IO, MongoDB, and Google Cloud services demonstrates the power and flexibility of cloud computing for modern software applications.