Research Document

Movimingle

Contents

Introduction

This document describes the research process for my application this semester – Movimingle. It is designed to provide scalable and secure platform for collaborative movie selection using microservices architecture. The research is conducted and described using ICT DOT framework.

Research Topic and Objectives

The main objective of this research is to make and document the rational decisions that need to be taken for the development of a scalable and secure collaborative movie selection application. This research includes important points of the application, such as scalability, security, deployment processes, and the integration of external APIs.

Research Questions

Main Question

How can the microservices architecture be strategically implemented to provide a scalable user experience in the Collaborative Movie Selection Application?

Sub-Questions

1. What architectural approaches are recommended for handling user favorites and voting features to ensure scalability and responsiveness?
2. How can the deployment processes be designed to enable smooth operation and rapid scaling in response to varying user demands?
3. Which security strategies are most effective for ensuring the integrity and confidentiality of user data within a microservices architecture?
4. Is the use of an external movie information API needed in the scope of the project?

Detailed Research Analysis

Main Question: Strategic Implementation of Microservices Architecture

Research Method

- **Literature Study:** Reviewing current best practices and trends in microservices architecture from sources such as blogs, tech article, and case studies.
- **Prototyping:** Developing prototypes to test various architectural designs and their impact on scalability and performance

Analysis

Independent service creation, deployment, and scalability are made possible by microservices architecture. The capacity to scale individual components in response to user needs without harming the system as a whole is improved by this decoupled approach. Microservices need to have the management

of key tools like Docker and Kubernetes, which offer capabilities like fault tolerance, load balancing, and autonomous scalability.

Microservices can be deployed and scaled independently, which helps achieve better flexibility and efficiency. For example, Kubernetes has features such as auto-scaling, which automatically adjusts the number of running instances based on the current load. This way the optimal resource usage and performance is ensured. Docker is a tool for quickly building, testing and deploying software. It works by isolating the program in a container environment to assemble the code without risk to the larger application. After testing and fixing any bugs, developers can deploy the code safely and efficiently. This architecture style is supported by major cloud providers like Microsoft Azure, Amazon Web Services, and Google Cloud, which have extensive documentation and tools to implement microservices effortlessly.

The priority will be to Create the Kubernetes cluster locally, because it will be more cost-effective for when the load tests are being performed. After configuring everything, I will consider which parts of the software are going to be deployed in any of the cloud services providers.

(Deploying Microservices on Kubernetes, Mehmet Ozkaya, Jan. 2021)

(Microservice architecture - Benefits, how to implement, challenges, etc., Matvey Romanov, Jul. 2023)

(Scaling Microservices: A Comprehensive Guide, Chameera Dulanga, Jun. 2023)

Sub-Question 1: Architectural Approaches for Handling User Favorites and Voting Features

Research Method

- Design Pattern Research: Investigating design patterns suitable for managing state and ensuring responsiveness in microservices.
- Prototyping: Implementing and testing these patterns to evaluate their effectiveness in real-time synchronization of user favorites and voting features.

Analysis

In order for the system to manage user favorites and voting features efficiently, an event driven architecture would be very useful. When using this approach, the services communicate via events, which helps to decouple services and makes the system more scalable.

When a user adds a favorite movie to their favorites or creates a new watch party, an event can be generated and processed asynchronously by the relevant services. This will provide the user with a smoothly running system under high loads. This can be further improved by using message broker, such as RabbitMQ, which can help manage these events and ensure reliable communication between the services.

Another useful technology that can help the improvement of the performance during high loads is using a NoSQL database like Cassandra. With its high read and write throughput it is effective when I need the user interactions to be fast and responsive. Cassandra has the ability to scale horizontally, which adds more database nodes as needed to handle the increased load.

My initial idea was to use different database technologies for the different needs of the microservices. The chosen databases were Cassandra and PostgreSQL. I chose PostgreSQL for its transactional integrity

and the complex query support. However, after discussing it with my mentor and rethinking my decision, I have found that these features are irrelevant for my project, since the data I am saving is not really important and nothing major depends on it will not need this and is complicating the development unnecessarily. Although Cassandra has been struggling for a long time in the area of the distributed transactions, with the version 5.0 and newer, it will offer ACID transactions. This is very helpful for development of applications like mine, where the only database in use can be Cassandra.

(ACID Transactions Change the Game for Cassandra Developers, April 2023)

(Microservices and Message Queues, Jan. 2023)

(RabbitMQ: Concepts and Best Practices, Sep. 2023)

(The Complete Guide to Event-Driven Architecture, Aug. 2023)

Sub-Question 2: Designing Deployment Processes for Smooth Operation and Rapid Scaling

Research Method

- Prototyping: Setting up continuous integration and continuous deployment (CI/CD) pipelines using GitHub Actions to automate the deployment process.
- Field Testing: Conducting load testing to observe how the system behaves under different user loads and identifying bottlenecks.

Analysis

The implementation of CI/CD pipelines using GitHub Actions is essential for the quick deployment of new features and updates. GitHub Actions streamlines this process by automating the build, test, and deployment workflows. Using Kubernetes for orchestration allows for automatic scaling based on real-time user demands, ensuring that the system remains responsive under varying loads.

GitHub Actions automates the build, test, and deployment processes, reducing the risk of errors and ensuring that changes are deployed quickly and reliably. Kubernetes' auto-scaling features adjust the number of running instances based on the current load, ensuring that the system can handle high traffic periods without performance degradation. Load testing with tools like Apache K6 helps identify and mitigate performance bottlenecks, ensuring a smooth user experience.

CI/CD practices also enhance collaboration among development teams by providing a shared platform for integrating code changes, running tests, and deploying updates. This practice reduces integration issues and allows for faster delivery of new features.

Comparing the alternatives such as Jenkins, GitLab CI, and Azure DevOps, the reason for settling on GitHub Actions is its ease of integration with GitHub repositories, hence easier setup and maintenance of CI/CD workflows. GitHub Actions also provides a very generous free tier and extensive community support, which makes it equally cost-effective and robust in support for the current project.

(Build a CI/CD workflow with Github Actions, James Turnbull)

(Running distributed load tests on Kubernetes, Olha Yevtushenko, Jun 2022)

(Horizontal Pod Autoscaling, Feb. 2024)

Sub-Question 3: Effective Security Strategies for Microservices Architecture

Research Method

- Literature Study: Reviewing best practices for securing microservices from reputable tech websites and cybersecurity blogs.
- Security Testing: Performing vulnerability assessments and penetration testing to identify and address security weaknesses.

Analysis

Effective security strategies for microservices include using Auth0 for secure authentication and authorization. The implementation of API gateways such as Envoy and Kong to perform central security controls, including rate limiting, IP allowlisting, and API key management is another aspect of the security. An API gateway offers a single point of entry and control for all microservices, ensuring that any incoming request is filtered to allow only authorized users. As well as that, regular security testing from penetration to vulnerability scanning is needed to ensure the integrity and protection of user data within the system. Auth0 provides a robust and flexible framework for authentication with authorization, in which a user would only be authenticated once and be issued a token to use with subsequent requests. This kind of support from many identity providers allows users to impact proper management credentials securely.

The other security layers are network security, mutual TLS (mTLS) for service-to-service communication and implementing role-based access control (RBAC). Also, patches for known vulnerabilities must be regularly updated in all microservices. Grafana monitoring and logging will give real-time insight into the system's security posture, enabling detection and response to potential threats as early as possible in the kill chain.

(API Gateway Security Best Practices, Misbah Thevarmannil, Feb. 2023)

(Secure your API Gateway APIs with Auth0, Jimmy, Dahlqvist, Jul. 2021)

Sub-Question 4: Necessity of External Movie Information API

Research Method

- Literature Study: Researching available movie information APIs and their benefits from online resources.
- Prototyping: Integrating an external API into the prototype to evaluate its effectiveness and impact on the application.

Analysis

The use of an external movie information API, such as The Movie Database (TMDB), will be ideal, as such a database generally has to be updated very often. By using such a service, the application's architecture is kept simple, and the quality of the data on movies is already assured for the users. TMDB is an open and fully functional source of up-to-date data on films and TV shows. By integrating TMDB, the app can provide users with accurate and current information about movies, including details such as cast, crew, trailers, and user ratings. This integration will reduce the development and maintenance burden since

data updating and administration are handled by the external API provider. Prototyping this integration may highlight the potential for issues and, thus, fine-tune the implementation.

Alternatives checked were OMDB, Open Movie Database, and IMDb API. OMDB is a free, open-source database, but the data coverage and reliability remain inferior to those of TMDB. The IMDb API gives complete data but with more significant limits and is much more expensive. With the richness of data sources, ease of integration, and nurturing by a large community, TMDB was the best candidate for use in the project at hand. This is because it is the best.

Furthermore, external APIs can be exploited for better user engagement with features that include personalized content recommendations according to the history and preference of the viewer. What this means is that much data from TMDB is used, and so, in effect, users would have relevant material up their sleeves. The API use helps to reduce the chances of inconsistency of data and, therefore, the reliability of the application.

Conclusion

The decision to use microservices for the Movimingle application development is strategic and aims to create an app that is scalable, flexible and secure. The benefits of the microservice architecture are observed when using independent service deployment through Docker and Kubernetes. This allows the services to be scaled individually based on the load by the users. That is being monitored by tools and some load tests are performed before the application goes into production. The architectural decisions to use event-driven design or NoSQL databases support the efficient handling of user interactions. Continuous integration pipelines make updates easier and the use of API gateways protect the user and the software from ill intent people. The data management is simplified by using an external movie database like TMDB, which also offers extensive movie data which would be helpful for the users. All of this ensures that Movimingle remains responsive, secure, and scalable in the events of a rapidly growing user base.