Research Document TuneTurtle

Nieben van Sint Annaland
06-12-2024

Creating a Music Player using Microservice Architecture on Kubernetes in Azure

Introduction

The objective of this research document is to explore the best technologies for developing a music player application, akin to Spotify, using a microservice-based architecture hosted on a Kubernetes cluster in Azure. A critical aspect of this research is to compare gRPC and Spring Bot, two prominent frameworks for building microservices, and determine the most suitable option for my application. The document will conclude with a recommendation to use Spring Boot over gRPC, considering various factors such as complexity and integration with Kubernetes.

Microservice Architecture

Microservices architecture involves breaking down a monolithic application into smaller, independent services that can be developed, deployed and scaled individually. This approach offers several advantages, including improved scalability, flexibility and maintainability. Each microservice typically handles a specific business function and communicates with other services through APIs.

Key Requirements for the Music Player Application

1. **Scalability:** The application must handle a high volume of concurrent users and audio streaming requests.
2. **Resilience:** The system should gracefully handle failures and ensure high availability
3. **Maintainability:** Individual services should be easy to update, maintain, and deploy without affecting the entire system
4. **Performance:** The application must deliver low-latency streaming and quick response times for user interactions

Hosting on Kubernetes in Azure

Kubernetes is an open-source platform for automating deployment, scaling, and management of containerized applications. Azure Kubernetes Service (AKS) provides a managed Kubernetes environment, simplifying cluster management and maintenance.

Benefits of Using Kubernetes and AKS

1. **Automated scaling:** Kubernetes can automatically scale services based on demand, ensuring efficient resource utiliziation
2. **High Availability:** Kubernetes supports self-healing, automatically restarting failed containers and redistributing workloads.
3. **DevOps Integration:** Kubernetes integrates well with CI/CD pipelines, facilitating continuous integration and delivery
4. **Resource Management:** Kubernetes provides robust resource management capabilities, including efficient handling of CPU, memory, and storage.

Technology Comparison: gRPC vs. Spring Boot

gRPC

gRPC is a high-performance, open-source RPC framework developed by Google. It uses HTTP/2 for transport, Protocol Buffers for serialization, and supports multiple programming languages.

Pros of gRPC

1. **High Performance:** gRPC offers low latency and high throughput, making it suitable for real-time applications.
2. **Efficient Serialization:** Protocol Buffers are compact and efficient, reducing the payload size and improving performance
3. **Strong Typing:** gRPC provides strong typing and contract-first API design, minimizing errors and improving API reliability.
4. **Streaming Support:** gRPC natively supports client, server, and bidirectional streaming.

Cons of gRPC

1. **Complexity:** Implementing gRPC requires learning Protocol Buffers and understanding the intricacies of HTTP/2
2. **Limited Browser Support**: gRPC is not natively supported in browsers, requiring gRPC-Web for web clients.
3. **Operational Overhead:** Managing and debugging gRPC services can be challenging due to its complexity
4. **HTTPS Requirement:** gRPC mandates the use of HTTPS, which can complicate deployment and configuration in Kubernetes

Spring Boot

Spring boot is a popular Java-based framework for building microservices and enterprise applications. It simplifies the deployment process by providing pre-configured templates and a wide range of integrations.

Pros of Spring Boot:

1. **Ease of Use:** Spring Boot offers a convention-over-configuration approach, making it easy to set up and start developing applications.
2. **Rich Ecosystem:** Spring Boot has a vast ecosystem of libraries and integrations, including Spring Cloud for microservices and Spring Data for database access.
3. **Flexibility:** Spring Boot supports both synchronous (REST) and asynchronous (WebFlux) communication, providing flexibility in designing APIs
4. **Integration with Kubernetes**: Spring Boot integrates well with Kubernetes, including support for Spring Cloud Kubernetes for configuration and service discovery.

Cons of Spring Boot

1. **Resource Intensive:** Spring Boot applications can be more resource-intensive compared to lightweight frameworks
2. **Learning Curve:** Although easier than traditional Spring, there is still a learning curve associated with Spring Boot and its ecosystem.

Decision: Choosing Spring Boot over gRPC

After evaluating the pros and cons of gRPC and Spring Boot, the decision to use Spring Boot for the application is based on the following considerations:

1. **Ease of Implementation:** Spring Boot’s convention-over-configuration approach simplifies the development and reduces the learning curve, allowing faster iteration and development
2. **HTTP/2 Complexity:** gRPC’s reliance on HTTP/2 and HTTPS adds complexity to the deployment and configuration process in Kubernetes, whereas Spring Boot’s REST APIs using HTTP/1.1 are simpler to manage
3. **Ecosystem and Integrations:** Spring Boot’s rich ecosystem and seamless integration with Kubernetes, including Spring Cloud Kubernetes, facilitate service discovery, configuration management, and monitoring
4. **Developer Familiarity:** Spring Boot’s widespread adoption and large community support ensure easier access to resources, libraries, and best practices.

Conclusion

In conclusion, for the development of this application using a microservice-based architecture hosted on a Kubernetes cluster in Azure, Spring boot is the preferred choice over gRPC.

I have spent a long time trying attempting to make a PoC of a gRPC microservices in Minikube (local kubernetes) and it would not work properly due to the HTTP/2 and HTTPS reliance. After a few weeks of continuously attempting and trying to fix it, I decided to try out Spring Boot, and it worked way easier and was able to spin up the microservices way quicker.