KAZAKH KBTU BRITISH
TECHNICAL
UNIVERSITY

REPORT

Microservice Architecture for Project

Course: GOLANG APPLICATION DEVELOPMENT

Checked by: Sherkhan Kubaidullov

Done by: Seitbekov Sanzhar

2021, Almaty

CONTENTS

1. Preface & Introduction

First, I'm not systems architect yet. Therefore, this report may contain errors and shortcomings. Thank you for understanding.

So, if my project grew into something more and became a real product, it would need a microservice architecture. If I had unlimited time and knowledge, I would make it possible not only to learn information about anime and manga, but also to watch and read them. Ideally, mobile apps for iOS and Android would be created, as well as a Web-app (or desktop app) for desktop.

Obviously, then I would need somewhere to store all these huge amounts of data. Thus, I would end up with something like a hybrid of a streaming service and an encyclopedia (database).

I think **Netflix** is the best example like my dreams from the real world. Because Netflix is one of the first companies to have successfully migrated from a traditional monolithic to cloud-based microservices architecture. In fact, Netflix implemented this architecture long before the term microservices was even introduced. It took more than two years for Netflix to achieve complete migration to the cloud. That is why for this report I was inspired by this popular service. So, let's start!

2. Architecture

Further, I will call my project - **Weebnet** (the first thing that came to mind, nevermind).
My Anime Project Weebnet could operate based on Amazon Cloud Services (**AWS**) and in-house content delivery work. Both systems will work together seamlessly to deliver high quality video streaming services around the world. From the software architecture point of view, Weebnet comprises 3 main parts: Client, Backend and Content Delivery Network.

Client is any supported browser on a laptop or desktop or a Weebnet app on smartphones etc. We need to develop our own iOS and Android apps to provide the best viewing experience for every client and device. By controlling our apps and other devices through its SDK, Weebnet can adapt streaming services transparently under certain circumstances such as slow networks or overloaded servers.

Backend includes services, databases, storages running entirely on AWS cloud. Backend basically handles everything not involving streaming anime, manga and so on.

Open Connect Delivery Network **CDN** is a network of servers optimized for storing and streaming large videos. These servers are placed inside internet service providers (ISPs) and internet exchange locations (IXPs) networks around the world. They are responsible for streaming videos directly to clients.

2.1 Playback Architecture

1. Delivery Networks constantly send health reports about their workload status, routability and available videos to Cache Control service running in AWS for Playback Apps to update the latest healthy servers to clients.
2. A Play request is sent from the client device to a service running on AWS to get URLs for streaming videos.
3. Playback Apps service must determine that the Play request would be valid to view the anime. Such validations would check subscriber’s plan, licensing of the video etc.
4. Playback Apps service talks to Steering service also running in AWS to get the list of appropriate servers of the requested video. Steering service uses the client’s IP address and ISPs information to identify a set of suitable servers that work best for that client.
5. From the list of servers returned by Playback Apps service, the client tests the quality of network connections to these servers and selects the fastest, most reliable server to request video files for streaming.
6. The selected server accepts requests from the client and starts streaming videos.

2.2 Backend Architecture

1. The **Client** sends a Play request to Backend running on AWS. That request is handled by AWS Load balancer.
2. **AWS** will forward that request to **API Gateway Service** running on AWS instances. The request will be applied to some predefined filters corresponding to business logics, then is forwarded to Application API for further handling.
3. **Application API** component is the core business logic behind operations. There are several types of API corresponding to different user activities such as Signup API, Recommendation API. In this scenario, the forwarded request from API Gateway Service is handled by Play API.
4. **Play API** will call a microservice or a sequence of microservices to fulfill the request.
5. **Microservices** are mostly stateless small programs and can call each other as well. To control its cascading failure and enable resilience, each microservice is isolated from the caller processes. Its result after run can be cached in a memory-based cache to allow faster access for those critical low latency requests.
6. **Microservice**s can save to or get data from a data store during its process.
7. **Microservices** can send events for tracking user activities or other data to the for real-time processing of personalized recommendation or batch processing of business intelligence tasks.
8. The data can be persistent to other data stores such as **MongoDB**, **Cassandra**, etc.

3. Services & Components

1. **Client.** Client components connections to Backend for content discovery and playback. While streaming anime, Client intelligently lowers the video quality or switches to different servers if network connections are overloaded or have errors.
2. **Gateway Service.** API Gateway Service component communicates with AWS Load Balancers to resolve all requests from clients. This component can be deployed to multiple AWS instances across different countries to increase Weebnet service availability.
3. **Application API.** For example, RPC methods and entities can be used, and client libraries/SDKs automatically generated in a variety of languages. This allows Application API to integrate appropriately with auto-generated clients via bi-directional communication and to minimize code reuse across service boundaries.
4. **Microservices.** A microservice can work on its own or call other microservices via **REST** or **gRPC** or **GraphQL**. The implementation of microservice can be like Application API. Each microservice can have its own datastore and in-memory cache stores of recent results. I think that we can use Redis as a primary choice for caching of microservices at Weebnet.
5. **Data Stores.** We can use NoSQL and SQL. **PostgreSQL** databases are used for anime title management and transactional purposes. **ElasticSearch** has powered searching titles for Weebnet apps. **Redis** can be used as caching storage. **MongoDB** can be used to handle large amounts of read requests with no single point of failure.
6. **Stream Processing Pipeline**. The stream processing platform has processed trillions of events and petabytes of data per day. It will also automatically scale as the number of subscribers increases. **Kafka** is responsible for routing messages as well as buffering for downstream systems.

4. Design Ideas & Goals

1. Ensure high availability for streaming services at global scale.
2. Tackle network failures and system outages by resilience.
3. Minimize streaming latency for every supported device under different network conditions.
4. Support scalability upon high request volume.

High availability. The availability of streaming services depends on both the availability of Backend services and servers keeping the streaming video files.
The goal of Backend services is to get the list of most fast servers to a specific client, either from cache or by execution of some microservices.

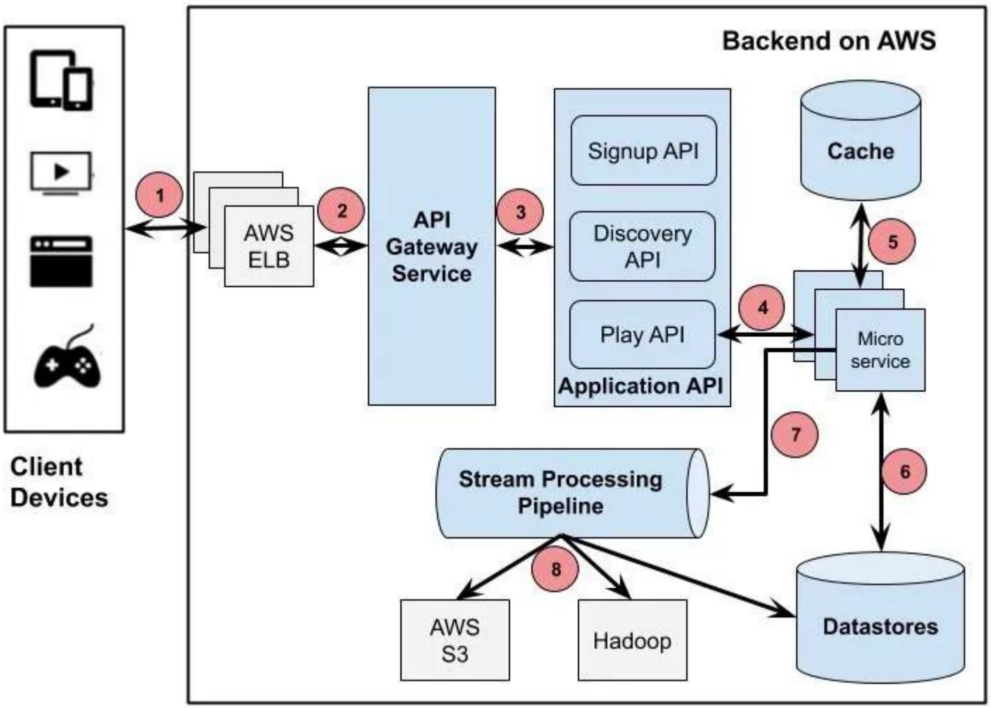
Low latency. The client would immediately switch to other nearby servers with the most reliable network connection if there is a network failure to the current selected server or that server is overloaded. It can also lower the video quality to match with the network quality in case it finds out a degradation in network connection.

Resilience. Designing a cloud system capable of self-recovering from failures or outages is the main reason why we should use AWS cloud.

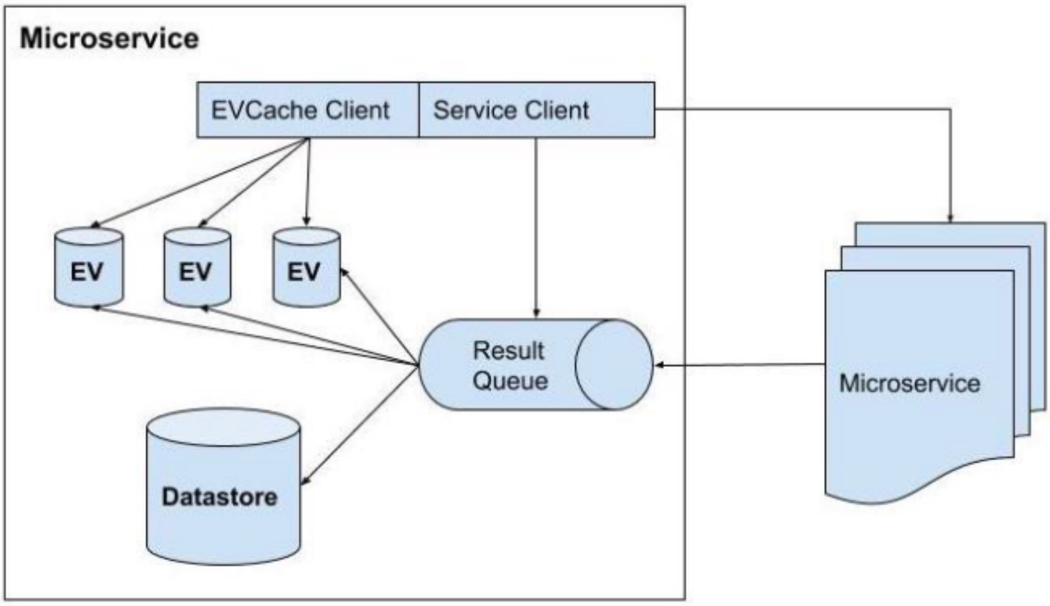
Scalability. We can use the container management system - Docker. Ideally, any component can be deployed inside a Docker container. Also, again we can use AWS Scaling Services. And of course, Redis like key-value object stores and ElasticSearch also offer high availability and high scalability with no single point of failure.

5. A few UML diagrams

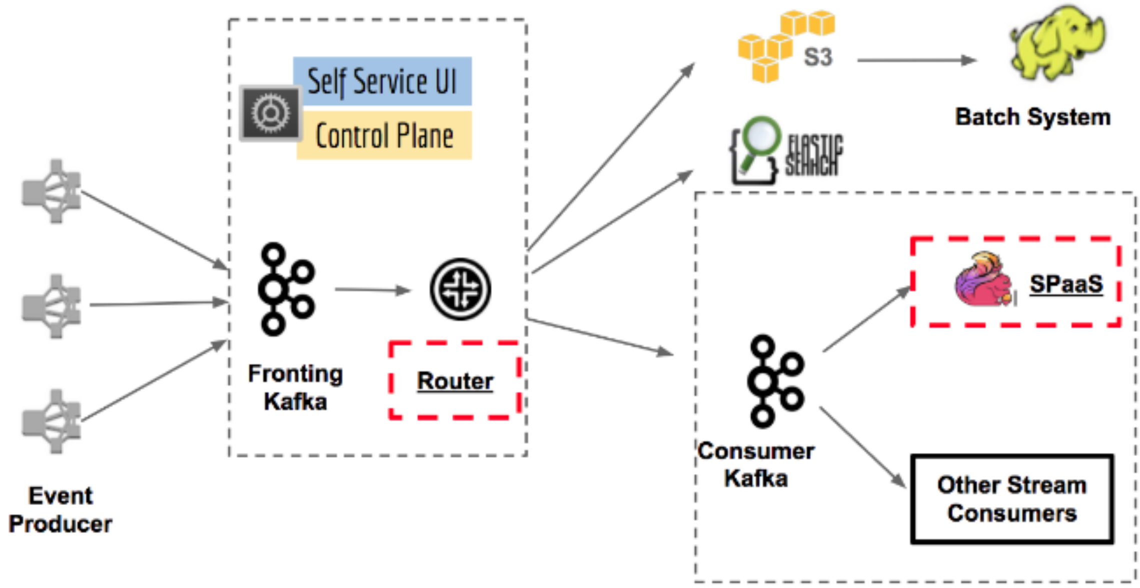
UML Diagram for Backend Service with AWS



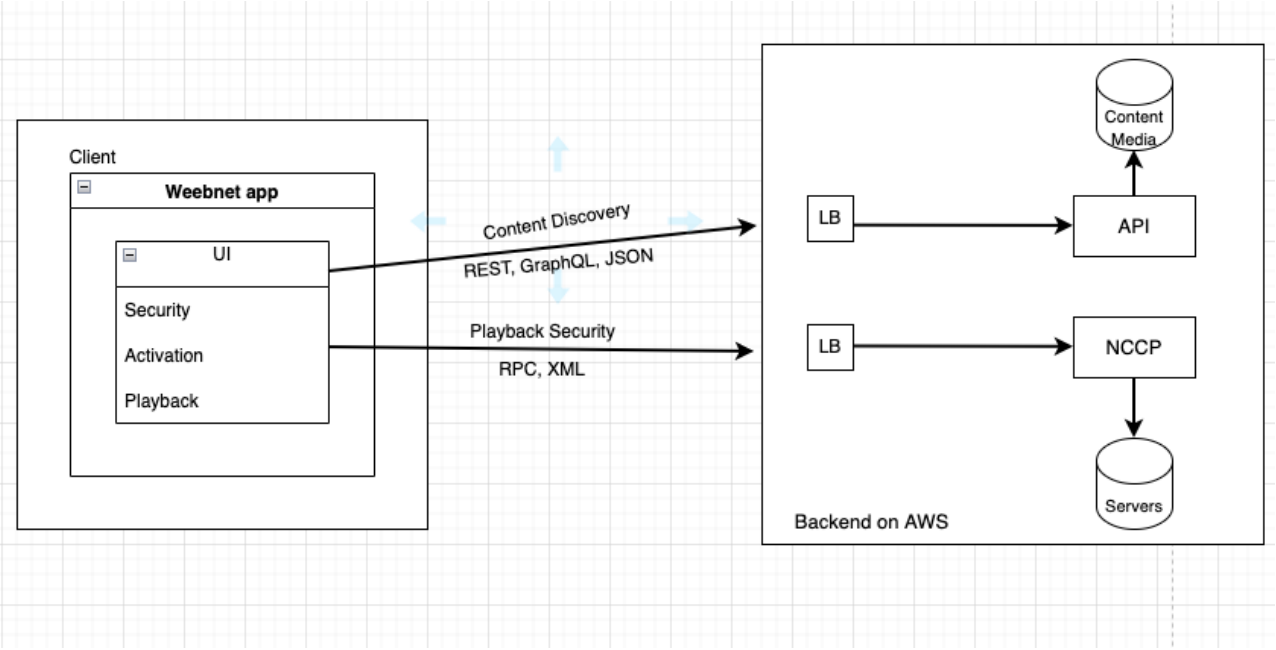
UML Diagram for Microservices



UML Diagram for Stream Processing Pipeline



UML Diagram for Client & Backend Communication



6. Conclusion

Let's summarize. The report has described the microservices architecture of Weebnet. I tried to analyze different design goals in terms of availability, latency, scalability, and resilience. I was inspired by Netflix's cloud architecture, proven by their production system to serve millions of subscribers running on thousands of virtual servers. Such architecture can demonstrate high availability with optimal latency, strong scalability through integration with AWS cloud services and many other microservices. I suppose that their service can serve as a reference implementation of how a microservice production system should be built. And maybe (who knows?) one day such an application as "Weebnet" will really appear in the world and will unite all weeb people around the world! Thank you for your attention.

Best regards,
Seitbekov Sanzhar

7. References

1. A Design Analysis of Cloud-based Microservices Architecture at Netflix. By Cao Duc Nguyen on May 1, 2020.
2. Netflix: What Happens When You Press Play? By Todd Hoff on Dec 11, 2017.
3. Building and Scaling Data Lineage at Netflix to Improve Data Infrastructure Reliability, and Efficiency. By Di Lin, Girish Lingappa, Jitender Aswani on The Netflix Tech Blog. Mar 25, 2019.
4. Netflix Play API — Why we build an Evolutionary Architecture. By Suudhan Rangarajan at QCon 2018. Dec 12, 2018.
5. Kafka Inside Keystone Pipeline. By Real-Time Data Infrastructure Team. April 27, 2016.
6. Performance Vs Scalability. By Beekums. Aug 19, 2017.
7. Netflix Tech Blog: Microservices. Blog since May 8, 2013.