PG3402 - Project Description

A generic streaming service using microservices

Joar Heimonen
contact@joar.me

October 28, 2024

Contents

1 Introduction

This document will describe the implementation of a simple scalable streaming service built using multiple microservices[3]. We will describe how the system handles user access control, content management and load balancing. The most important element of this project will be JSON web tokens[2] that allow data to pass between the different microservices. The service will also feature load balancing using the GoodDNS server[1].

2 Overview

In this project, we aim to develop a robust streaming service that leverages a microservice‘s architecture to offer scalable video streaming. The service is designed to cater to two main types of users: administrators and registered viewers. Administrators have the ability to upload and manage content, while registered users can browse and stream content. The core functionalities include user registration and login, content management, and dynamic load balancing to handle varying loads. The system will utilize JSON web tokens for secure data transmission between microservices utilizing users as intermediates. It incorporates a custom DNS solution, GoodDNS[1], for load management.

3 User Stories

3.1 Story 1: User Registration

As a user,
I want to be able to register an account using a username and a password,
So that I can access the streaming service.

Acceptance Criteria:

1. The registration screen is accessible from the home page.
2. Users can enter a username and password.
3. The system validates that the username is valid.
4. An error message is displayed if the username is not valid.

3.2 Story 2: Content Browser

As a user,
I want to be able to browse content,
So that I can access and stream the content.

Acceptance Criteria:

1. The homepage consists of a list of content.
2. Users can select the content they want to watch.

3. The system checks if the user is logged inn.
4. The user is able to stream content when logged inn.
5. A login prompt is shown if the user is not logged inn.

3.3 Story 3: Add New Content

As an Administrator,
I want to add new content to the service,
So that users can watch the content.

Acceptance Criteria:

1. There is an "Admin" button on the home page.
2. Clicking the button takes the user to an admin page.
3. The content can be submitted on the admin page.
4. An error message is shown if the content submission fails.

3.4 Story 4: Add Tag to Video

As a registered user, **I want to** add tags to videos, **So that** I can help improve video categorization and searchability.

Acceptance Criteria:

1. Users can navigate to the video detail page.
2. There is an option to "Add Tag" next to the tags.
3. Users can type a tag and submit it.
4. The system validates the tag against predefined criteria (e.g., no special characters).
5. A confirmation message is displayed when the tag is successfully added.
6. The tag is visible to other users viewing the video with a count of how many others have added the same tag.

4 Requirements

4.1 Users

The system must be able to handle user registration and login. This includes the ability to securely store user credentials and personal information in a database.

4.2 Content Management

The system must be able to handle content uploading and streaming efficiently. This involves accepting new content from administrators, storing it in the content database and the file system, and serving it to authenticated users upon request. The system should support various media formats and ensure smooth streaming experiences even under high load. Additionally, content metadata management (like titles, descriptions, and tags) should be implemented to enhance content discoverability.

4.3 Load Balancing

The system should efficiently distribute incoming network traffic across multiple servers to ensure reliability and scalability. This involves using a DNS server that can dynamically allocate requests to the least loaded server. The load balancing DNS server must have a second failover DNS server. If necessary an infinite amount of sub-sub DNS servers can be implemented. The time to live is used to control how often users poll the DNS servers for new server allocations.

ttl	**ttl**
0	3600

- `userserver1.dns1.mydomain.com`
- `userserver1.dns2.mydomain.com`
- `contentserver2.dns2.mydomain.com`

5 Architecture

5.1 Discussion

There exists numerous ways to configure the nodes in our architecture. To avoid acquiring technical debt it is important to derive an architecture fitting of our needs. Therefore, we have derived a simple set of architectural guidelines.

- **Load Balancing**: The service must be able to efficiently balance load. No parts of the service can choke due to increased load.
- **Scalability**: All parts of the service must be able to scale as needed to account for increases in load.
- **Security**: The microservices must pass data to each other securely.
- **Simplicity**: There should be no unnecessary complexity.

The simplest way for microservices to pass data to each other is through the user in the form of JSON web tokens. The following scenarios can be solved using both JSON web tokens and direct communication between the micro services.

- **False metrics**: If metrics are not derived from the content management service but the client instead. False metrics might be reported by bad actors.
- **Non-existing content IDs**: If content IDs in the metadata management services manifest is not derived from the content management service. Metadata might be created for content that does not exist.

5.1.1 Load Balancing

There are an infinite amount of strategies for balancing load in services consisting of multiple nodes. And there is much to be gained from utilizing lower level protocols to achieve this. Using GoodDNS solves two problems, it lowers the bandwidth required as the DNS protocol as outlined in RFC 1035 is extremely lightweight. DNS also removes the need for proxying data allowing clients to communicate directly with the relevant services. This makes for a

much more reliable service as we don't risk our proxying solution functioning as a bottleneck.

Using DNS also has a couple of drawbacks. The DNS system is complex and can be quite slow. Recursive DNS servers not compliant with RFC 1035 might not respect a domain's TTL. In cases like this the service might break for users as they are feed out of date DNS data from non-compliant DNS servers. This can be mitigated by making sure scaling of the services happens slowly, so that all recursive DNS servers have time to update their records.

5.2 Nodes

Figure 1 shows a service consisting of the following microservices:

- **DNS Server (Load Balancer)**: Directs incoming network traffic to different servers, balancing the load and serving as a DNS resolver. All servers receive their own subdomains with short times to live.
- **Clients**: End-users who interact with the application through a web browser.
- **User Management Service**: Manages user authentication, authorization, and profiles, interfacing with the **User Database**.
- **Content Management Service**: Handles content storage and retrieval through the **Content Database** and manages file-based content with the **File System**.
- **Metadata Management Service**: Handles the metadata related to content, this service consists of a manifest of all content that has been uploaded to the **Content Management Service**, content descriptions, titles and tags.
- **Metrics Service**: Keeps track of content views, likes and dislikes.
- **Web Server**: Serves a static single page application used by the client to interact with the microservices.
- **Databases and File System**: Separate databases for users and content data ensure efficient management and scalability. The file system stores non-database content.

Communication between clients and services is secured using HTTPS, with REST API requests facilitating interactions between clients and microservices. This architecture ensures security, and scalability under varying load.

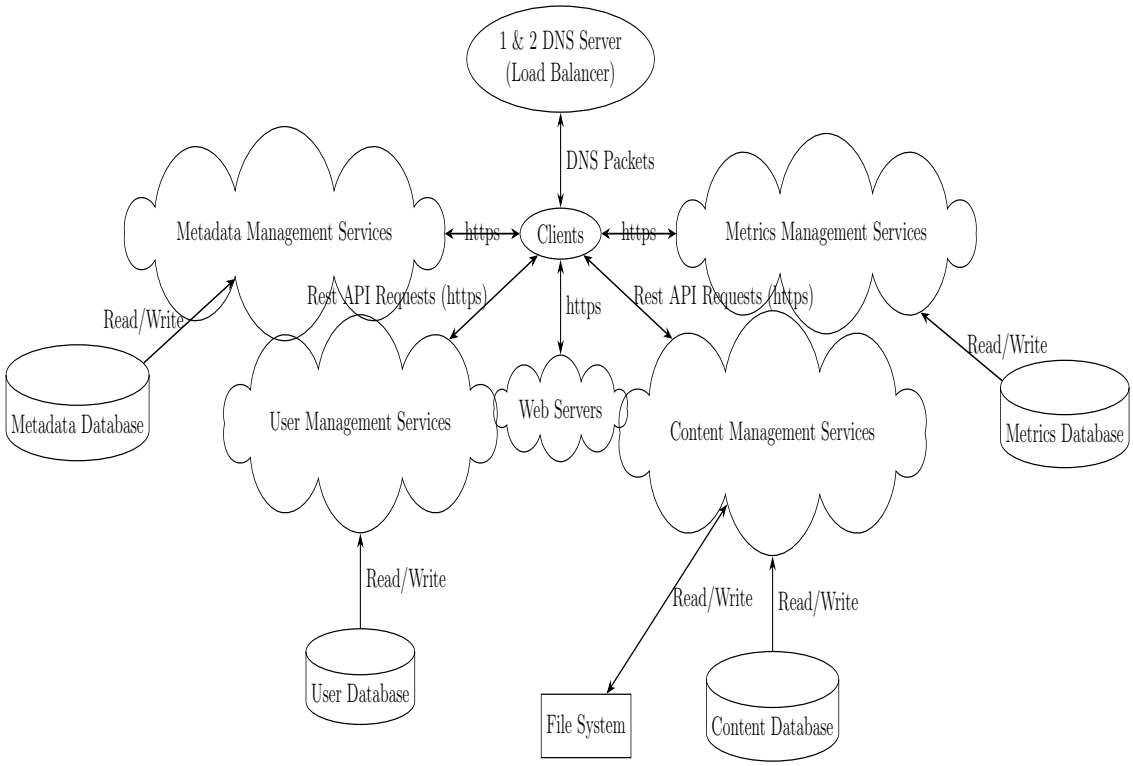

Figure 1: Detailed service architecture showing the interaction between different microservices and databases.

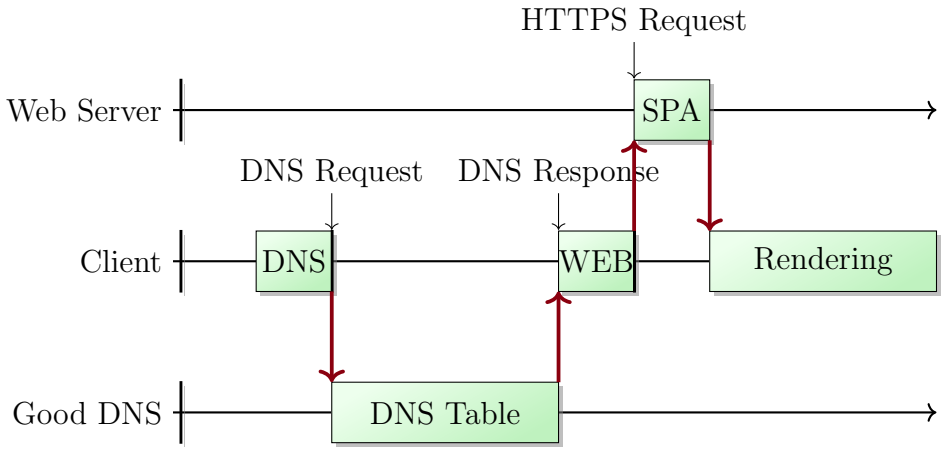

Figure 2: Timeline of a web request with DNS lookup.

6 Notes

6.1 GoodDNS

The architecture does not utilize a central API gateway, Instead it relies on the clients sending requests to the relevant services. This is made possible by the GoodDNS[1] load balancer. This architecture was selected as a central API gateway requires proxying all traffic.

6.2 JSON Web Tokens

Most think of JSON Web Tokens as a measure for authentication. However, JSON Web Tokens are as outlined in RFC 7519[2] a method for transferring data between two parties. By passing as much of the server to server communication trough clients in the form of JSON Web Tokens we will greatly simplify the service architecture yet still have a service that is able to scale. For example, the Metadata Management Service will append and re-sign the users JSON Web Token with a claim that can be used by the other server to validate what videos the user has access to delete.

References

[1] Joar Heimonen. *Slenderman00/GoodDns*. Dec. 2023. (Visited on 10/13/2024).

[2] Michael B. Jones, John Bradley, and Nat Sakimura. *JSON Web Token (JWT)*. Request for Comments RFC 7519. Internet Engineering Task Force, May 2015. DOI: 10.17487/RFC7519. (Visited on 10/14/2024).

[3] "Microservices". In: *Wikipedia* (Oct. 2024). (Visited on 10/14/2024).