Authentication Research

Introduction

This document is going to provide the research done on methods of setting up an efficient authentication and authorization methods within a system consisting of microservices to ensure performance. Role-based authorization is not required in the current architecture. The system as a whole is expected to have around 4,000,000 requests/hour (~1,111/second). Among those requests, most would require authentication and authorization. The aim for this research is to provide a viable solution for the given issue.

Research questions

How can authentication and authorization be implemented in a system consisting of microservices, with minimal hindering to the performance?

- What are good practices, commonly used to solve this problem?
- How do those practices compare in terms of purpose?
- How do those practices compare in terms of performance?
- How do those practices compare in terms of security?

What are good practices, commonly used to solve this problem?

The most common practices in setting up a secure authentication and authorization within your microservices system is to either utilize JSON Web Tokens(JWT) or make use of a third-party authorization services like OAuth2.0, Auth0, Keycloak, Firebase Authentication etc. However, some of the services like Firebase Authentication provide only limited use free of charge which removes such options from consideration. What is more, services like OpenID Connect are built on top of OAuth2.0 providing authentication layer on top of the authorization of OAuth2.0. Generally, most third-party authorization services operate in a similar way with slight differences. Performance could turn out to be a setback when making use of third-party authorization since scalability depends on external factors.

How do those practices compare in terms of purpose?

Few of the most popular choices for implementation of authentication and authorization are compared based on their generally known pros and cons:

- **JWT** – Authentication tokens, generated by a service within the system. Furthermore, JWTs are stateless – each token contains all the required data within itself, which considerably reduces server load. What is more, each JWT can be validated by more than one service easily, if needed.

- **OpenID Connect** – Extends OAuth2.0 by adding an authentication layer on top of it. It provides a standardized authentication flow which can reduce complexity in implementation.

- **Keycloak** – an open-source identity and access management solution providing authentication and authorization services. Offers centralized user management, role-based access control, multifactor authentication, etc.

How do those practices compare in terms of performance?

Few of the most popular choices for implementation of authentication and authorization are compared based on their performance:

- **JWT** – JWTs are usually considered one of the best performing methods for authentication and authorization due to the required server load for performing these tasks. Since these tokens are stateless and all required data is contained within the token itself, the server does not need to query a database which improves performance significantly.

- **OpenID Connect** – Since this service provides an authentication layer on top of OAuth2.0, which introduces additional steps in the authentication process. More steps in the process usually result in increased latency.

- **Keycloak** – has built-in method of caching to increase performance. However, performance also depends on the Keycloak server. Higher latency could be expected due to its nature – being a third-party authentication provider which requires additional steps in the process.

How do those practices compare in terms of security?

Few of the most popular choices for implementation of authentication and authorization are compared based on their security:

- **JWT** – Tokens should be cryptographically signed to ensure data integrity. Furthermore, tokens need to be handled and secured properly to avoid security vulnerabilities. In case these 2 important aspects are handled, JWTs are considered secure.

- **OpenID Connect** - provides standardized authentication, which reduces the risk of misconfigurations. In case there is a security breach within the identity provider, it would also affect this system majorly.

- **Keycloak** – provides built-in features like user federation, encryption and access control. Once properly set up, it applies the standardized authentication and authorization based on given instructions.

Conclusion

All three of the researched solutions could be of value in different systems, depending on the requirements. Keycloak and OpenID Connect provide more advanced features and access control compared to JWTs. However, JWTs perform way better due to the low server load they cause, compared to the other two. Furthermore using a third-party service for authenticating tokens increases latency because additional steps and database queries increase processing time. In terms of security, if implemented properly, JWTs can be as secure as the other 2 options. Since this system would not require role-based authorization or any other more complex security features but focuses more on performance, authentication and authorization will be implemented using JWTs in this project.