2024

Credit card HLD Docunemt

HLD DOCUMENT FOR CREDIT CARD MICROSERVICES PROJECT

YORAM NAGAWKER

Contents

Credit card HLD

Introduction

This High-Level Design (HLD) Document offers a comprehensive overview of the architecture and design principles behind the credit card system. The document outlines the major components, their interactions, data flow, and integrations with external services, aiming to provide a clear and concise understanding of the system's architecture.

Technology Stack

Backend Technologies:

- Java 17
- Spring Boot 3
- REST

DevOps & CI/CD:

- GitHub
- Docker, Docker Compose
- Jenkins
- Google Jib
- Docker Hub

Monitoring & Logging:

- Grafana
- Prometheus
- Grafana Loki
- Grafana Tempo

Messaging:

- RabbitMQ

Database:

- MySQL

Cloud & Orchestration:

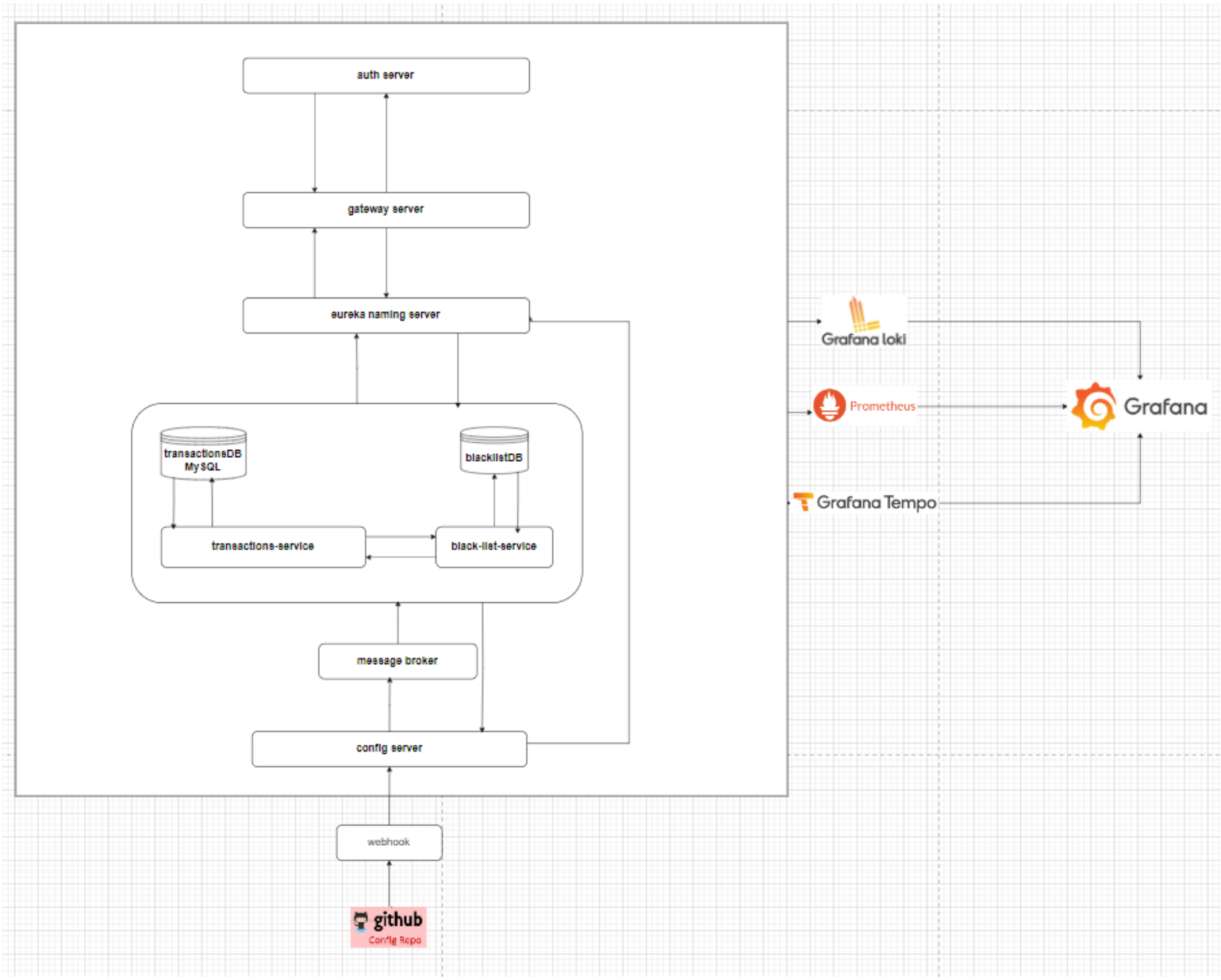
- Kubernetes
- AWS

Security:

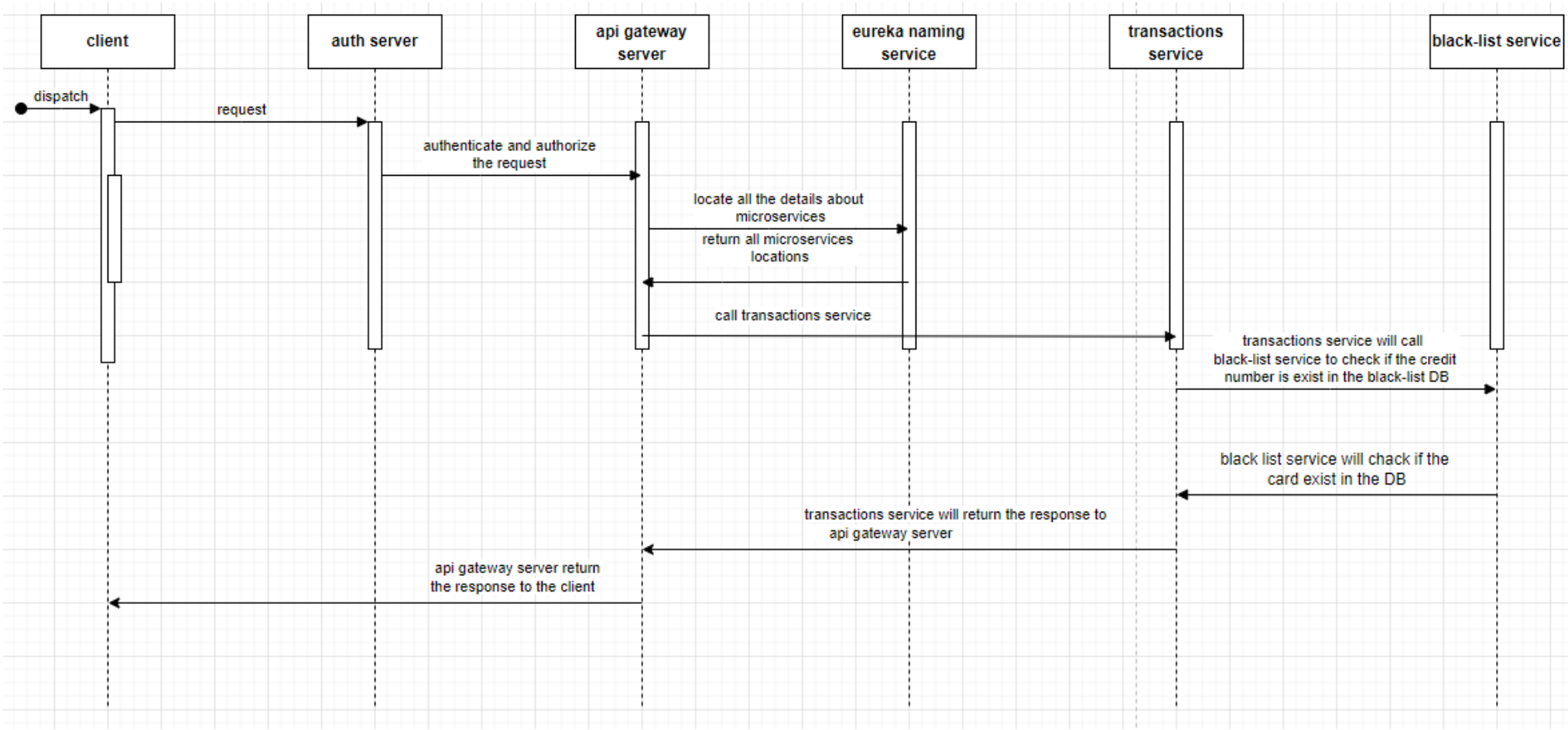
- OAuth2/OpenID Connect
- keycloak

Architectural Overview

The credit card system is designed based on the Microservices architecture pattern. It consists of multiple, interconnected components, each responsible for specific functionality. This design promotes separation of concerns, modularity, and ease of maintenance, while also ensuring scalability and extensibility. The system is built to handle dynamic traffic, high availability, and seamless integration with other systems.



Data Flow Diagrams



Modules Description

Auth server

For authentication and authorization, we are going to use OAuth2/OpenID connect, KeyClock as auth server and spring security.

This section explained in more details in the security paragraph.

Gateway server

The Gateway Server is built using Spring Cloud Gateway, providing an effective API routing mechanism. It also handles cross-cutting concerns such as security, monitoring, metrics, and resiliency, ensuring a robust and efficient communication layer between clients and microservices.

The service gateway sits as the gatekeeper for inbounded traffic to microservice calls within our application, with a service gateway in place our service clients never directly call the URL of an individual service, but instead place all calls to the service gateway.

For fault tolerance we are going to use resilience4j library, it's offering the following patterns for increasing fault tolerance due a network problems or failure of any of the multiple services:

- **Circuit braker** - used to stop making requests when invoked is failing.
- **Fallback** - alternative paths to failing requests.
- **Retry** - used to make reties when retries when a service has temporarily failed.
- **Rete limit** – limits the number of calls that a service receives at a time.
- **Bulkhead** – limits the number of outgoing concurrent requests to a service to ovoid over loading.

Eureka naming service

Service Discovery is a crucial aspect of distributed systems and microservice architectures. The Eureka Naming Service facilitates dynamic discovery of services within the system. It enables microservices to register and deregister themselves as they scale or fail, allowing seamless communication between services. Here are the key components and concepts associated with service discovery:

1 .Service Registry

- A central database or registry that keeps track of information about available services in the system.
- Each microservice, when it starts or shuts down, registers or deregisters itself with the service registry. This includes information such as its network location, IP address, and the ports it is using.

2 .Service Provider

- The microservice that offers a specific functionality or resource is considered a service provider.
- Service providers register themselves with the service registry so that other services can discover and connect to them

3 .Service Consumer

- The microservice that needs to use the functionality provided by another service is considered a service consumer.
- Service consumers query the service registry to discover the location and details of the service providers they want to interact with.

4 .Dynamic Updates

- Service discovery systems handle dynamic updates seamlessly. As new microservices are added or existing ones are removed or scaled, the service registry is updated accordingly.
- This dynamic nature allows the system to adapt to changes without manual intervention.

5 .Load Balancing

- Service discovery often involves load balancing mechanisms. When a service has multiple instances, load balancing ensures that requests are distributed optimally across these instances, improving performance and resource utilization.

6 .Decentralized Communication

- Unlike traditional monolithic architectures where communication may be centralized, service discovery enables decentralized communication among microservices. Services can find and communicate with each other without relying on a central authority.

7 .Health Checking

- Service discovery systems often include health checking mechanisms to monitor the status of service instances. Unhealthy or unreachable instances can be automatically excluded from the registry to ensure that only healthy services are used.

Config server

Spring Cloud Config enables centralized configuration management in a distributed microservices environment. It allows for externalizing the configuration of all microservices in a central repository, simplifying updates and deployments across various environments. With the config server we have central place to manage external properties for application across all environments.

with spring config server, we can separate the configuration/properties from the microservices so that same Docker image can be deploy in multiple envs and inject configuration/properties that microservices needed during startup of the server.

each component is going to load his properties file by connecting to the config server.

RabbitMQ

RabbitMQ will be used in conjunction with Spring Cloud Bus to link nodes in the distributed system. It facilitates real-time message broadcasting, enabling state changes (e.g., configuration updates) or management instructions to be propagated to all relevant services efficiently.

When a properties file is been update in the GitHub the config server is going update the RabbitMQ for the update and the others nodes are going to get the message to take the latest properties file from the config server.

Webhooks

Webhooks will notify an external web server when new configuration data is pushed to the GitHub repository. The webhook will invoke the /monitor endpoint on the Config Server, triggering a configuration change event and prompting a refresh on all subscribed nodes.

GitHub config repo

GitHub config repo will hold all properties files on the web, the files are encrypted and will be load when the config server will start up, this way we can properly secure our GitHub repo so no one can access it and versioning.

transactions-service and black-list-service

The system involves two RESTful services transactions-service and black-list-service that communicate with each other and have multiple components for implementation, testing, and monitoring.

Components Overview

The architecture consists of several key components that are organized into layers:

DTO (Data Transfer Objects)

DTOs are used for the transfer of data between services, encapsulating the request and response data structures. They make it easier to define the structure of the data that is sent over the network.

- **Request DTO:** Defines the data expected from the client.
- **Response DTO:** Defines the data structure that the client will receive as a response.
- **Error DTO:** Used for sending error information in a structured format (e.g., error codes, messages, etc.).

Entity

Entities define the structure of data within the database.

Exception Handling

A global exception handler would be used to manage exceptions across the application, making sure that common errors (like validation errors or service unavailability) are handled uniformly. We will use @ControllerAdvice to globally catch exceptions in Spring applications, making sure to send consistent error responses.

- **Custom Exception Classes:** These would represent specific business exceptions like ServiceUnavailableException
- **Global Exception Handler:** Handles different types of exceptions and returns appropriate responses with proper HTTP status codes.

Repository

Repositories handle database interaction via Spring Data JPA. They abstract away the complexities of database queries and allow you to perform basic CRUD operations (Create, Read, Update, Delete). You will use the JpaRepository interface for basic operations and add custom methods when needed.

REST Controllers

Controllers define the REST API endpoints that accept HTTP requests and return HTTP responses. They map incoming HTTP requests to the service layer methods.

Service Layer

The service layer contains business logic and coordinates between the controllers and repositories. This layer is responsible for the actual processing of data and implementing the core functionality.

Communication between transactions-service and black-list-service

OpenFeign

OpenFeign is a declarative web service client for Java. It allows you to define HTTP clients (for calling other RESTful services) in a much simpler way by just defining an interface and annotating methods

Eureka Naming Service

Eureka is a service discovery tool that allows microservices to discover each other dynamically. Instead of hardcoding the service URL, the services can register with Eureka, and other services can discover them through the service registry.

Logging, Monitoring, and Fault Tolerance

SLF4J for Logging

SLF4J (Simple Logging Facade for Java) will be used for logging across the application.

Swagger 3 for API Documentation

Swagger 3 is used to generate API documentation and provide interactive endpoints. This allows the developers and users to explore the API via a user-friendly UI.

Prometheus for Health Checks and Metrics

Prometheus can monitor the application's health and expose metrics (e.g., service uptime, response times, error rates). Spring Boot applications can integrate Prometheus through spring-boot-starter-actuator to expose metrics.

Resilience4j for Fault Tolerance

Resilience4j provides features like retries, circuit breakers, and rate limiters to make the services more resilient to failures. It will ensure that calls between services don't fail the whole system if one service is down or experiencing issues.

Testing with JUnit5

JUnit5 will be used for writing unit and integration tests across the application layers. Tests will ensure that each component works as expected and help prevent regressions.

The tests will be part of the continuous integration (CI) pipeline to ensure that changes do not break existing functionality.

Database Design

We use **MySQL** for persistent data storage with the following primary databases:

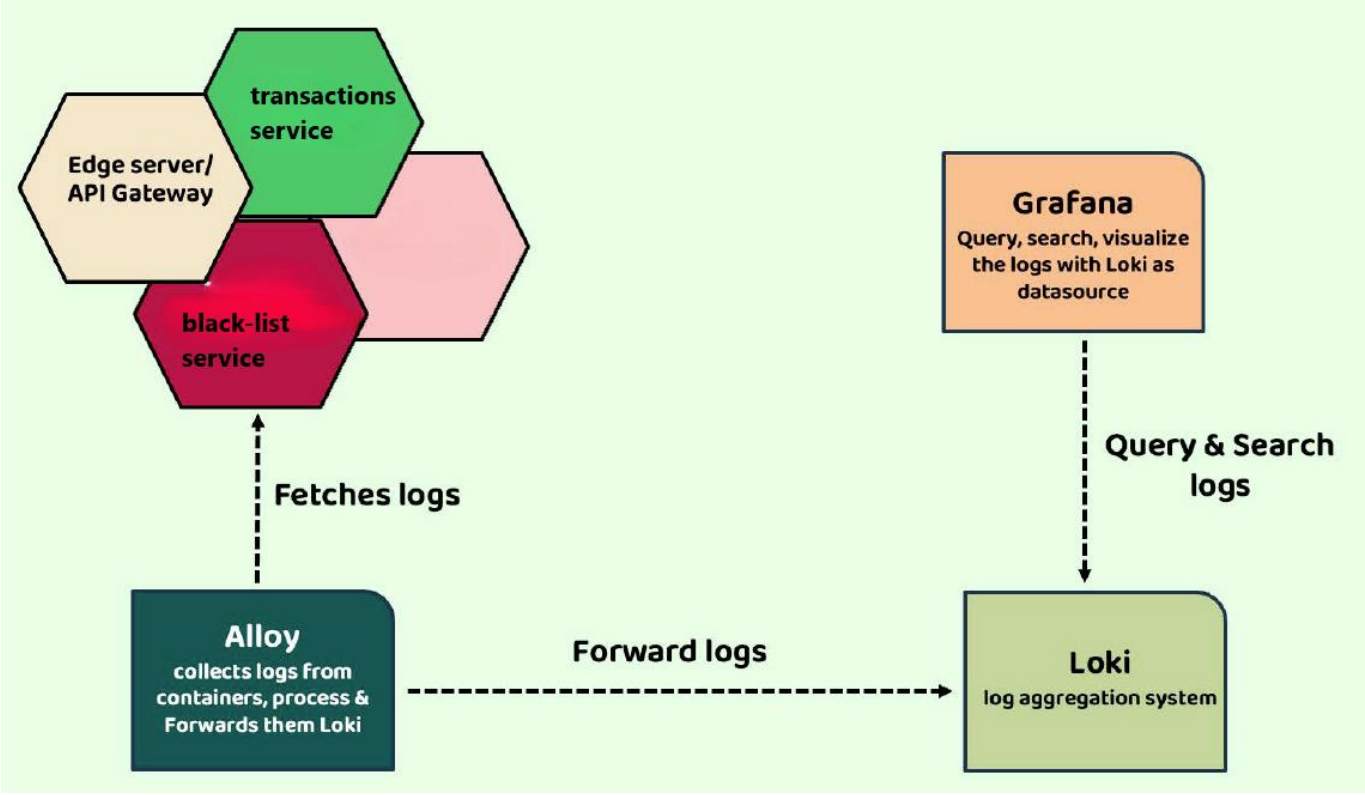
- **TransactionsDB**: Stores transaction details.
- **BlacklistDB**: Stores credit card numbers marked as invalid.
- **Security**: Credit card numbers are encrypted using **SHA256** and validated with the **LUHN algorithm**.

Grafana Loki

Grafana Loki is a horizontally scalable, highly available, and cost-effective log aggregation system.

Grafana Alloy is a lightweight log agent that ships log from containers to Loki.

Together, Grafana Loki and Alloy provide a powerful logging solution that help to understand and troubleshoot our system.

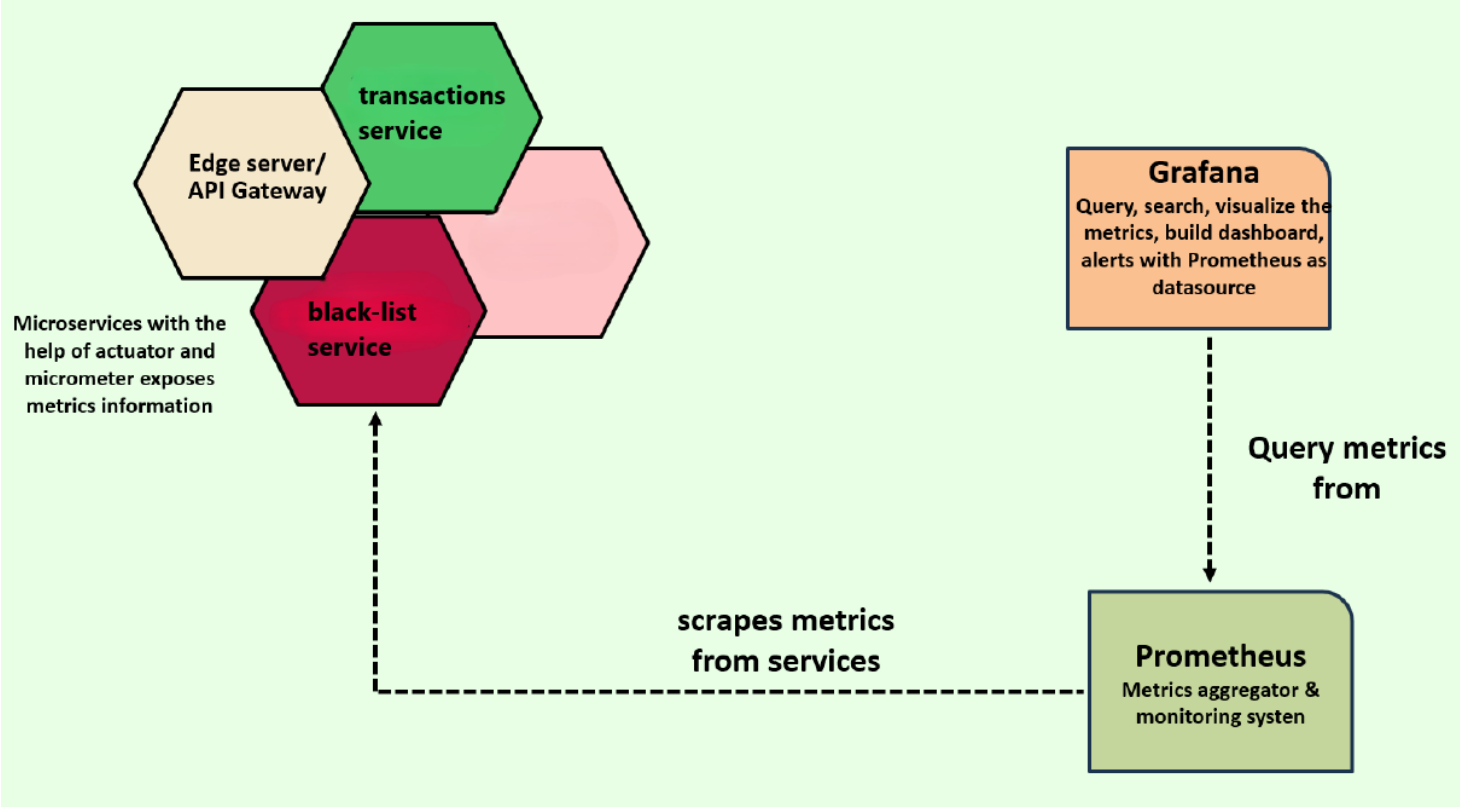


Prometheus

Metrics are numerical measurement of the application performance, collected and aggregate at regular intervals. Thay can use to monitor the application health and performance and set alerts or notifications when thresholds are exceeded.

To implements Prometheus with Grafana we are going to use:

- **Actuator** – actuator is mainly use to expose operational information about the running application health, metrics, dump, env etc. it uses HTTP endpoints to interact with it.
- **Micrometer** – micrometer automatically exposes /actuator/metrics data into something our monitoring system can understand.
- **Prometheus** – the most common format for exporting metrics.
- **Grafana** – Grafana is visualization tool that can be use to create dashboards and charts form Prometheus data

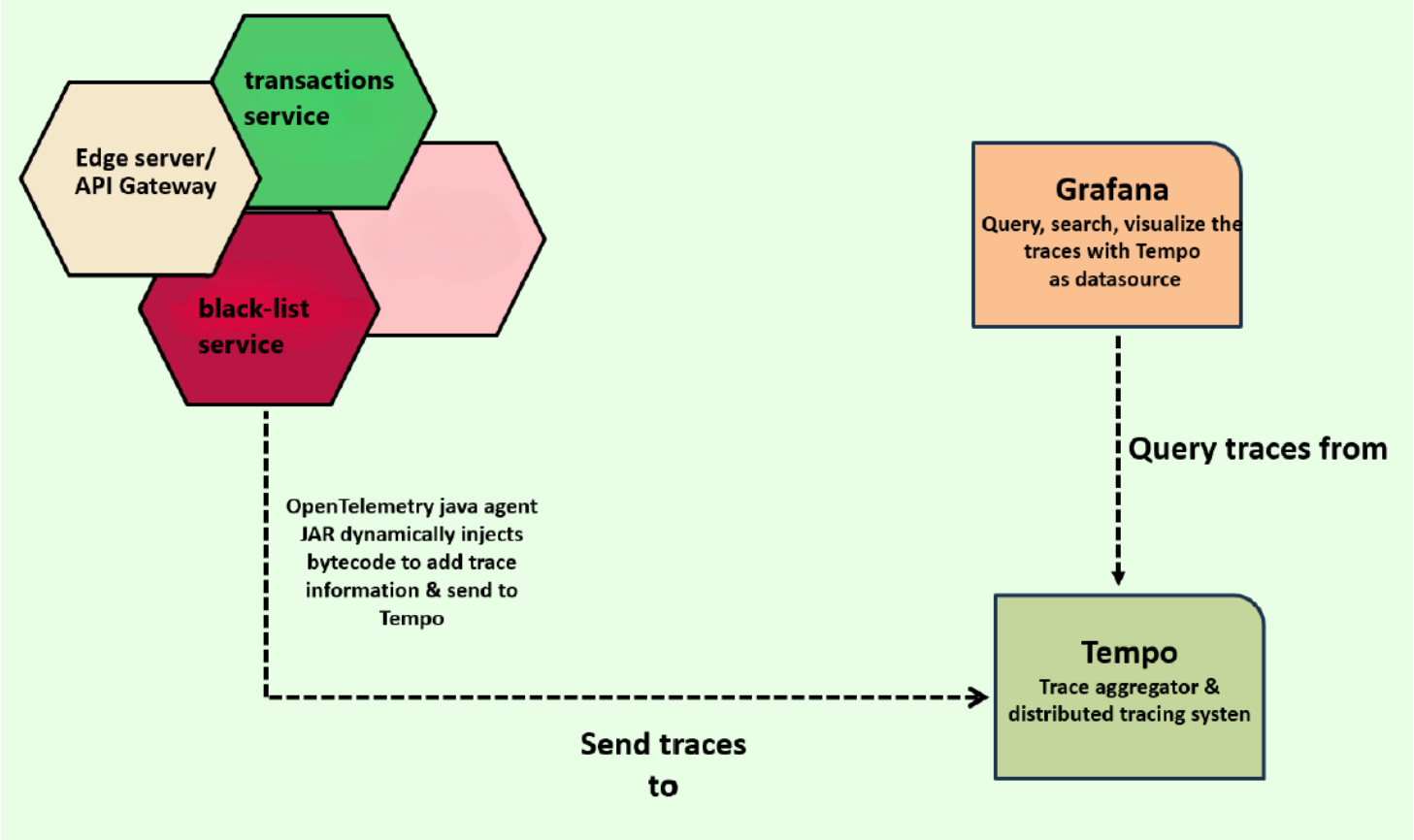


Grafana Tempo

Distributed tracing is a technique used in microservices or cloud-native applications to understand and analyze the flow of requests as they propagate across multiple services and components. It helps in gaining insights into how are processed, identifying performance bottleneck and diagnosing issues in complex distributed systems.

To implements Grafana Tempo with Grafana we are going to use:

- **OpenTelemetry** – using OpenTelemetry generate traces and spans automatically, OpenTelemetry is a vendor-neutral open-source Observability framework for instrumenting, generating, collecting, and exporting telemetry such as traces, metrics and logs.
- **Grafana Tempo** – index the tracing information using Grafana Tempo.
  Tempo is an open source, highly scalable and cost-effective distributed tracing backend designed for observability in cloud native enticements environments. It is a part of the Grafana obseralility stack and provides a dedicated solution for efficient storage, retrieval and analysis of trace data.
- **Grafana** – using Grafana we can connect to Tempo as a data source and see the distributed tracing in action, we can integrate Loki and Tempo as well so that we can jump to tracing details directly from logs inside Loki.



Grafana

Grafana is a multi-platform open-source analytics and interactive visualization web application. It can produce charts, graphs, and alerts for the web when connected to supported data sources.

Grafana is a centralize dashboard that will show all the logs, metrics and tracing.

DevOps & CI/CD

GitHub

The code for all Microservices will be store in the GitHub.

Docker, Docker Compose and Docker Hub

- Docker is used to containerize each microservice, making them portable and consistent across different environments (DEV, TEST, PROD).
- Docker Compose is used for managing multi-container applications. Separate Compose files will be used for different environments.
- Docker Hub is utilized for storing and sharing Docker images. Once the Docker images are created, they are pushed to Docker Hub, from where they are pulled into various environments for deployment.

Sequence of Starting Docker Containers:

The system defines the startup sequence of Docker containers, with each service depending on the health of the previous one before continuing:

1. transactionsDB
2. blacklistDB
3. RabbitMQ
4. Config Server
5. Eureka Naming Server
6. transactions-service
7. black-list-service
8. Gateway Server
9. Auth server

Jenkins

Jenkins is an open-source automation server that facilitates continuous integration (CI) and continuous delivery (CD). It automates various tasks such as building, testing, and deploying code.

Jenkins integrates with Docker to:

- Build Docker images as part of the CI/CD pipeline.
- Deploy applications in containers.
- Run tests within Docker containers to ensure that applications behave consistently across environments.

Security

OAuth2/OpenID Connect

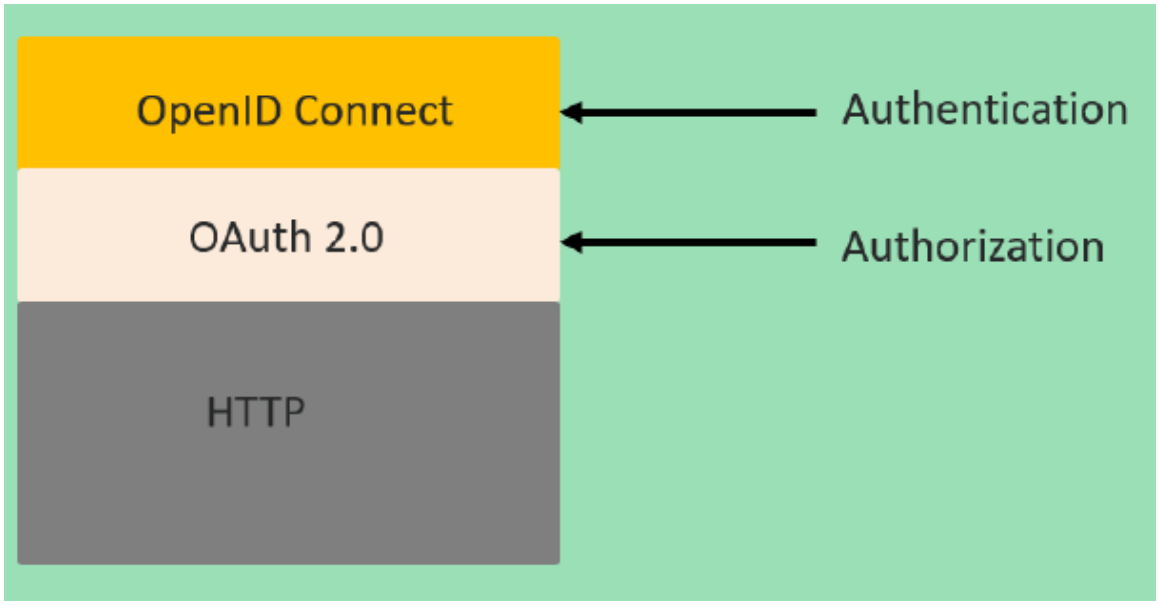
Oauth2.1 is a security standard where you give one application permission to access your data in another application.

below are the few advantages of OAuth2:

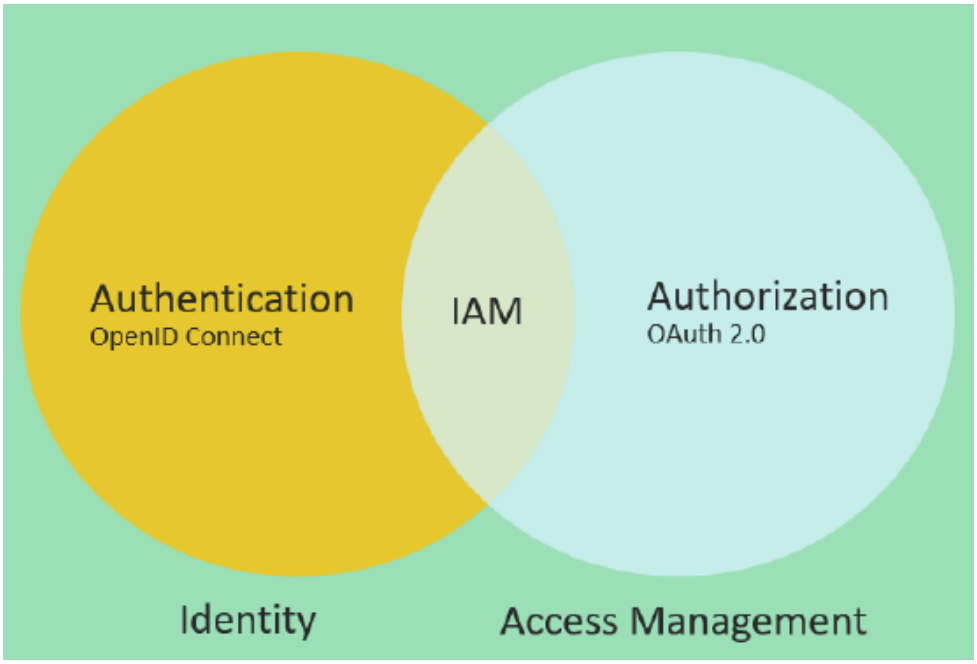
- **support all kinds of apps** - OAuth2 support multiple cases addressing different device capabilities. It's support server-to-server apps, browser-based apps, mobile/native apps, IoT devices and consoles/TVs.
- **Separation of Auth logic** – inside OAuth2 we have authorization server which receives requests from the client for access tokens and issue then upon successful authentication. This enables us to maintain all the security logic in a single place. regardless of how many applications an organization has, they all can connect to Auth server to preform login operations.
  All user credentials and client application credentials will be maintained in a single location which it's inside Auth server.
- **No need to share credentials** – if you plan to allow third-party application and services to access your resource, then there is no need to share your credentials.

OpenID

OpenId connect is a protocol that sits on the top of the OAuth20 farmwork, while OAuth2 provides authorization via an access token containing scops, OpenID connect provides authentication by introducing a new ID token which contains a new set of information and claims specifically for identity.



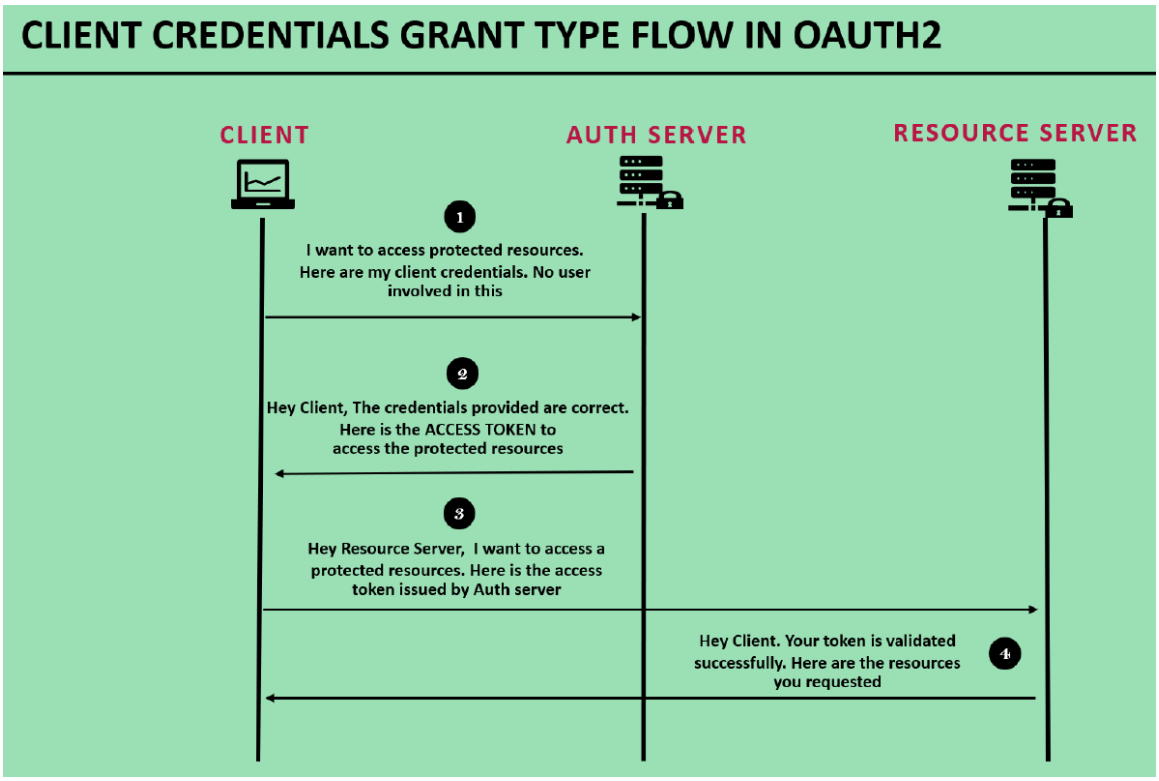
Identity is the key to any application, at the core of modern atheization is OAuth2.0, but OAuth2 lacks an authentication component. Implementing OpenID connect on the top of OAuth2 completes an IAM (Identity & Access Management) strategy.

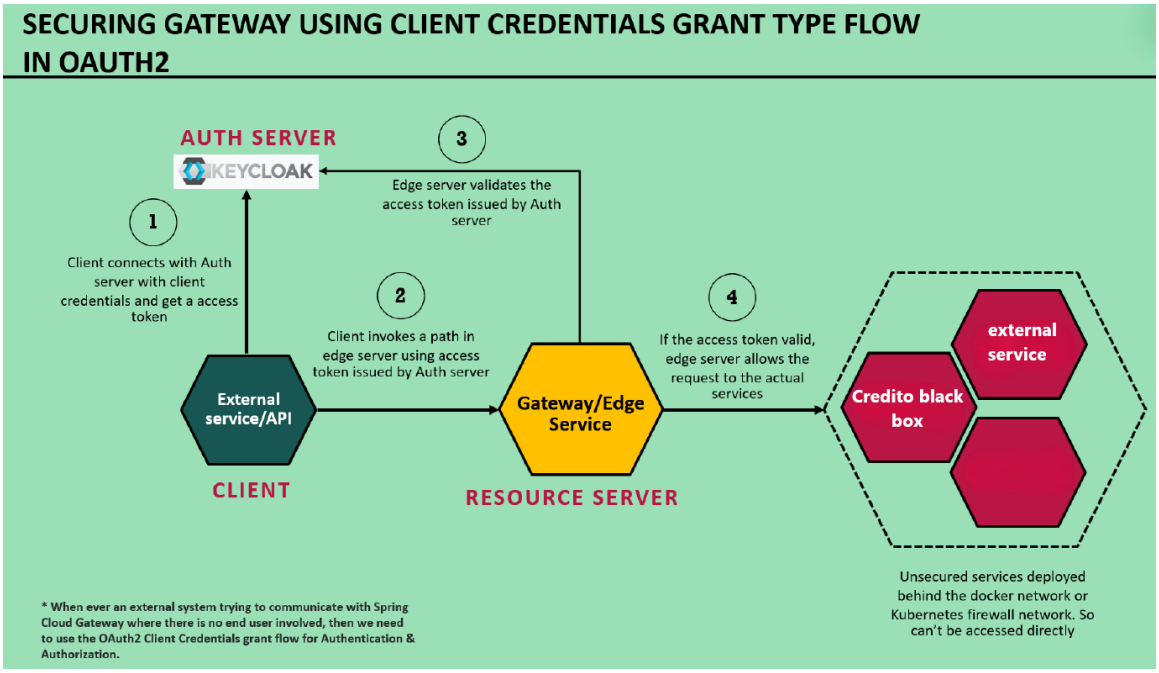


OAuth Grant Types

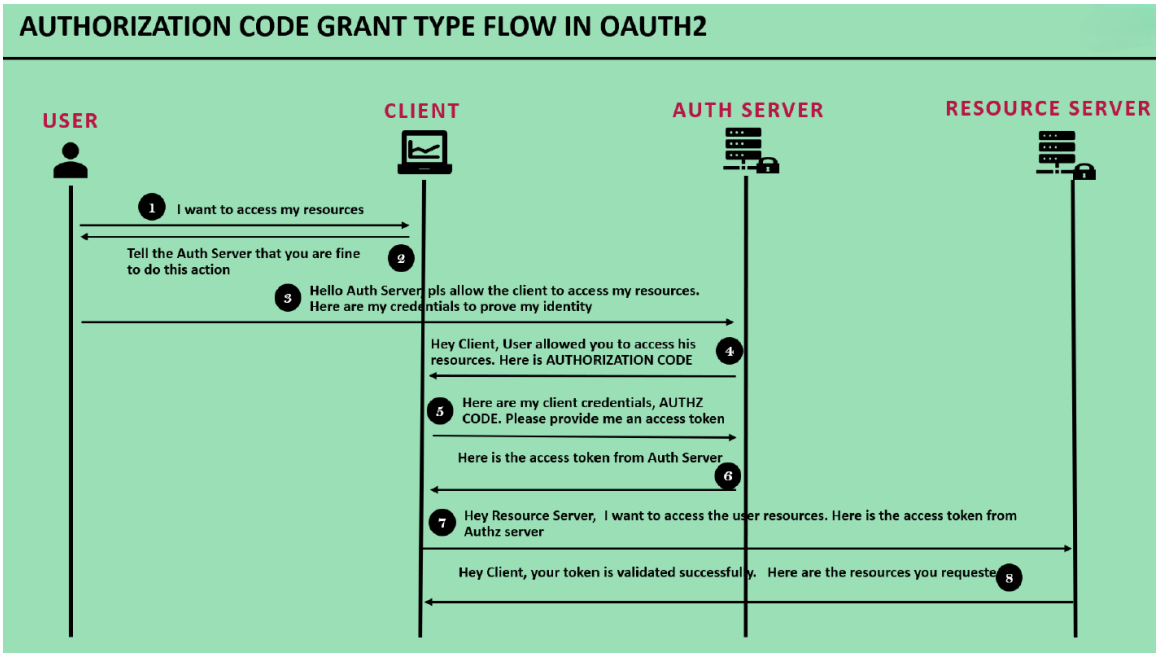
grant type basically refers to the way your app gets the access token. OAuth 2.0 offers different types of grant types

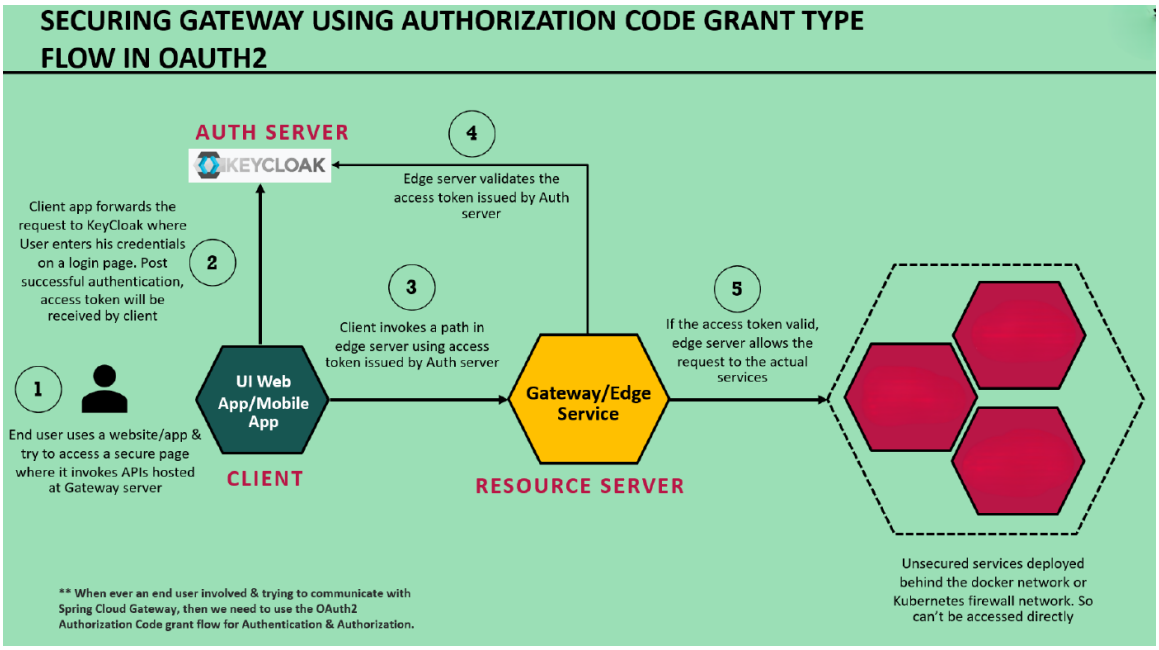
Client Credentials Grant





Authorization Code Grant





mTLS

for security between the microservices we are going to use mTLS, mTLS enhances the security offered by TLS by introducing mutual authentication between the client and the server, whiten the framework of mTLS, both the client and the server exchange their respective certificates and mutually confine each other's identities prior to establishing a secure connection.

Managing many certificates in mTLS can become challenging, which is where automatic mTLS through a service mash help ease certificate complexity.

credit card will act as its own certificate authority, this contrasts with standard TLS, which the certificate authority is an external organization that checks if the certificate owner legitimately owns the associated domain.

Deployment Architecture

Kubernetes

For Service Orchestration we are going to use Kubernetes, Kubernetes is an open-source system for automating deployment, scaling and managing containerized applications.

Kubernetes provides you with a framework to run distributed systems resiliently. Is takes care of scaling and failover for your application, provides deployment patterns it's provides you with:

- Service discovery and load balancing.
- Container & storage orchestration.
- Automated rollouts and rollback.
- Sefe-healing.
- Secret and configurations management.

Kubernetes is so modular, flexible and extensible that it can be deployed on-prem, in a third-party data center, in any of the popular cloud providers.

AWS

AWS – EKS (Elastic Kubernetes Service) is leveraged to manage Kubernetes clusters in the cloud, simplifying deployment and scaling of services.