coditas

Microservices architecture using Netflix oss and Apache kafka

Aim is to implement microservices arch using netflix OSS components with the latest dependency or tools available in the community.

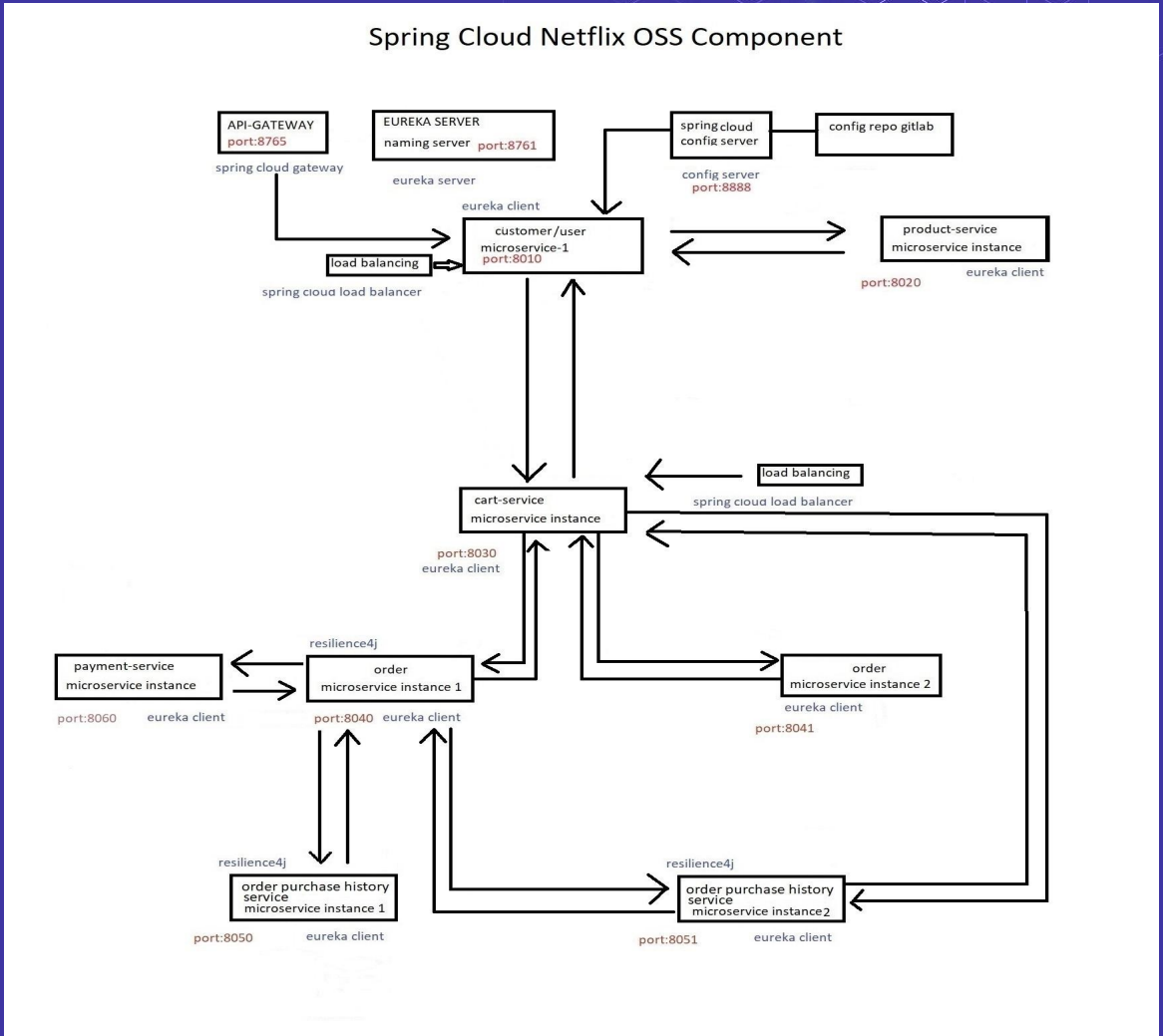
Presented by -
Rajat
Sejal
Rakshit

Components

Requirements	Component
Service registration and discovery	Eureka server and Eureka client
Api gateway	Spring cloud gateway
Configuration management(Config server and config client)	Spring cloud config server.
Refreshable configuration	Spring cloud bus and spring cloud monitor
Declarative rest client	Feign client
load balancing	Spring cloud load balancer
Fault tolerance	Resilience4j(Circuit breaker,slow calls)
Distributed Tracing	Sleuth-zipkin(Error tracing)
Centralized Logging	ELK Stack
Monitoring	Prometheus and Grafana

Application flow

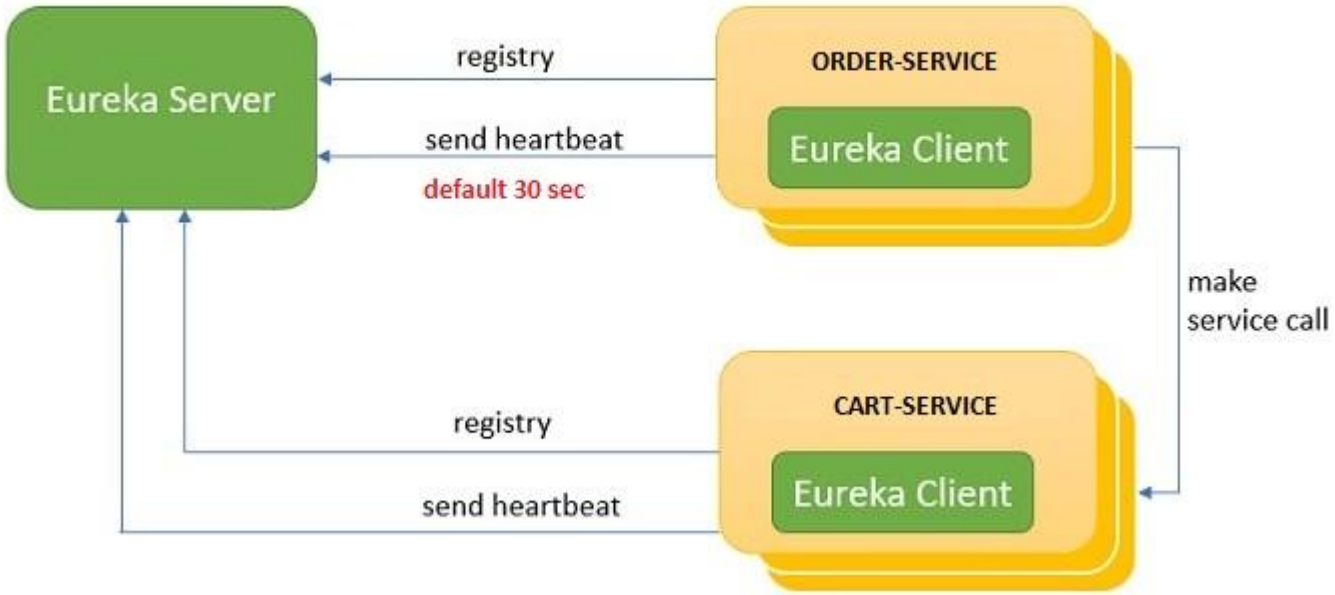
API gateway pattern



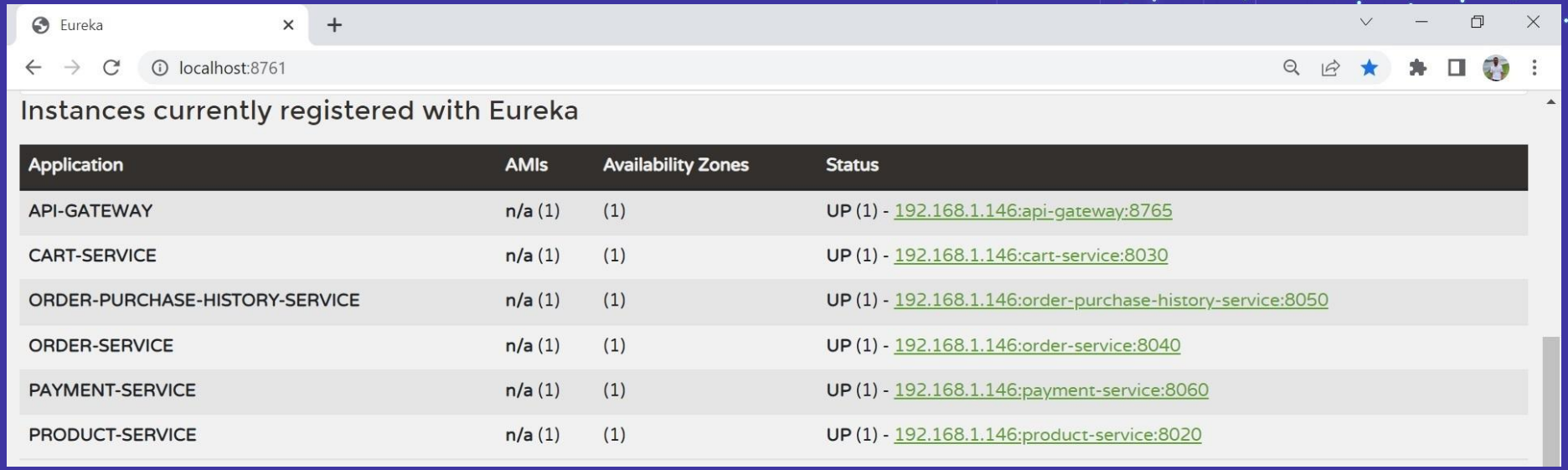
coditas

Service registration and discovery - Eureka server

- Eureka Server is an application that holds the information about all client-service applications.
- Every Microservice will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address.
- Eureka uses the client heartbeat to determine if a client is up.



Service registration and discovery - **Eureka server**

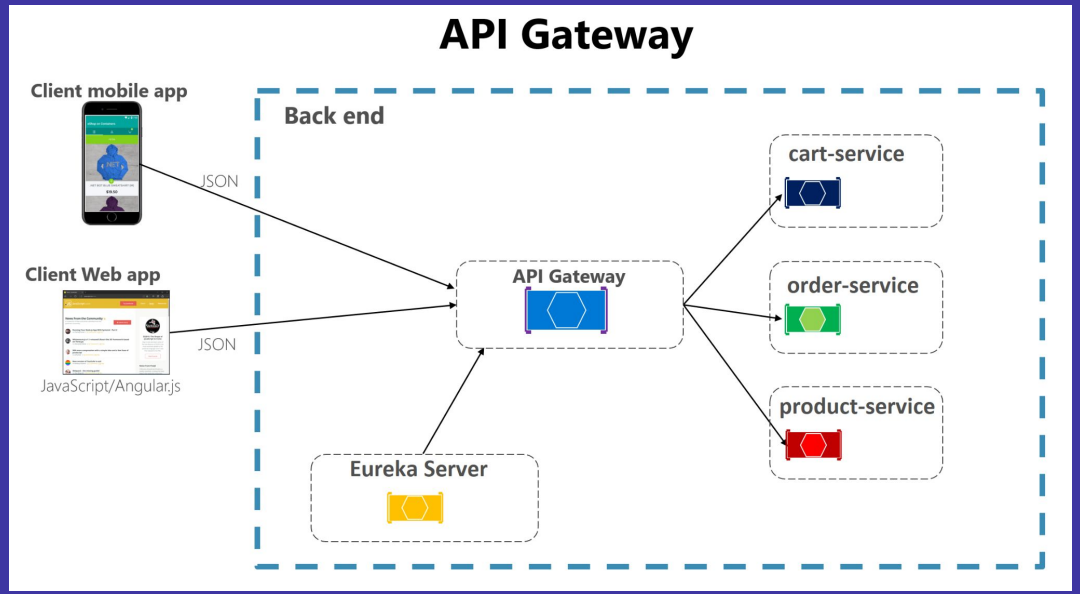


- It can be a microservice that is ready to work so it registers itself to the Eureka server.
- Configuration for Eureka client.

API Gateway - Spring Cloud Gateway

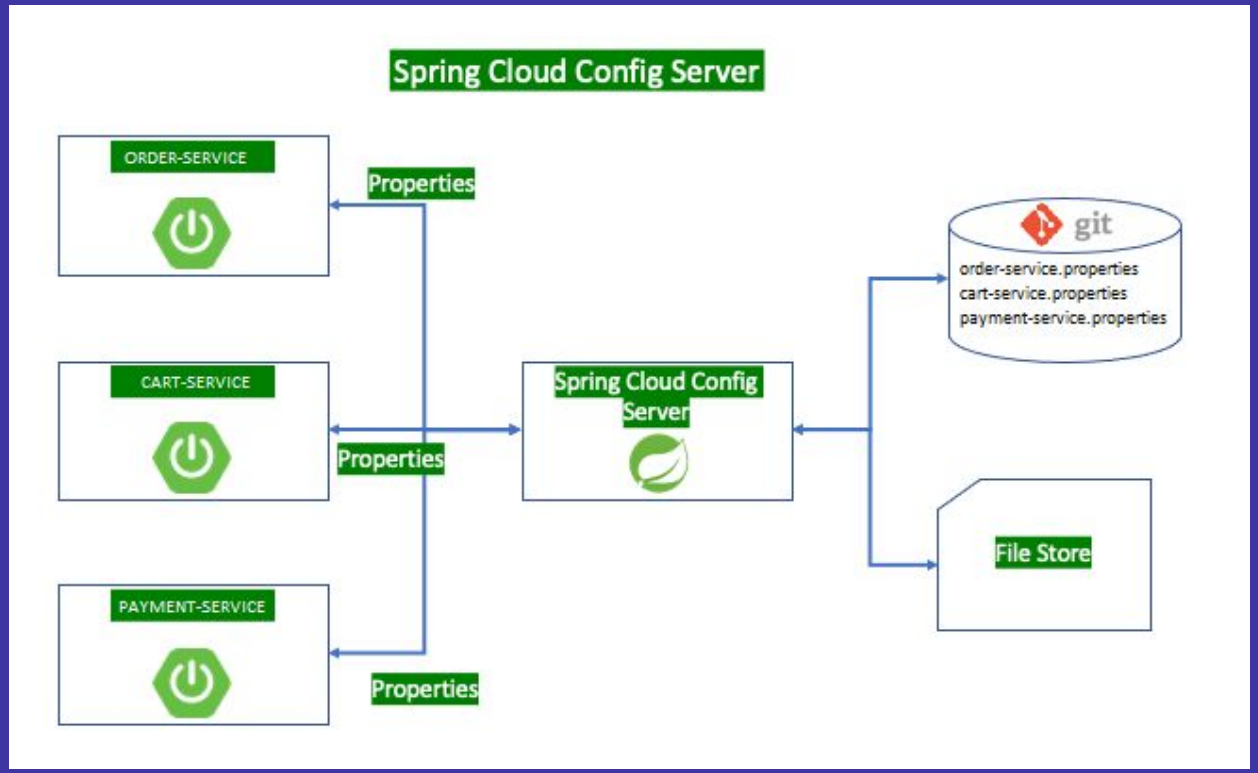
Spring Cloud Gateway aims to provide a simple, yet effective way to **route** to APIs and provide cross cutting concerns to them such as: **filter**, **security** etc.
We can do configuration in two ways.

1) In application.properties file.
2) By creating configuration class.



Configuration management - **Spring cloud config server/client**

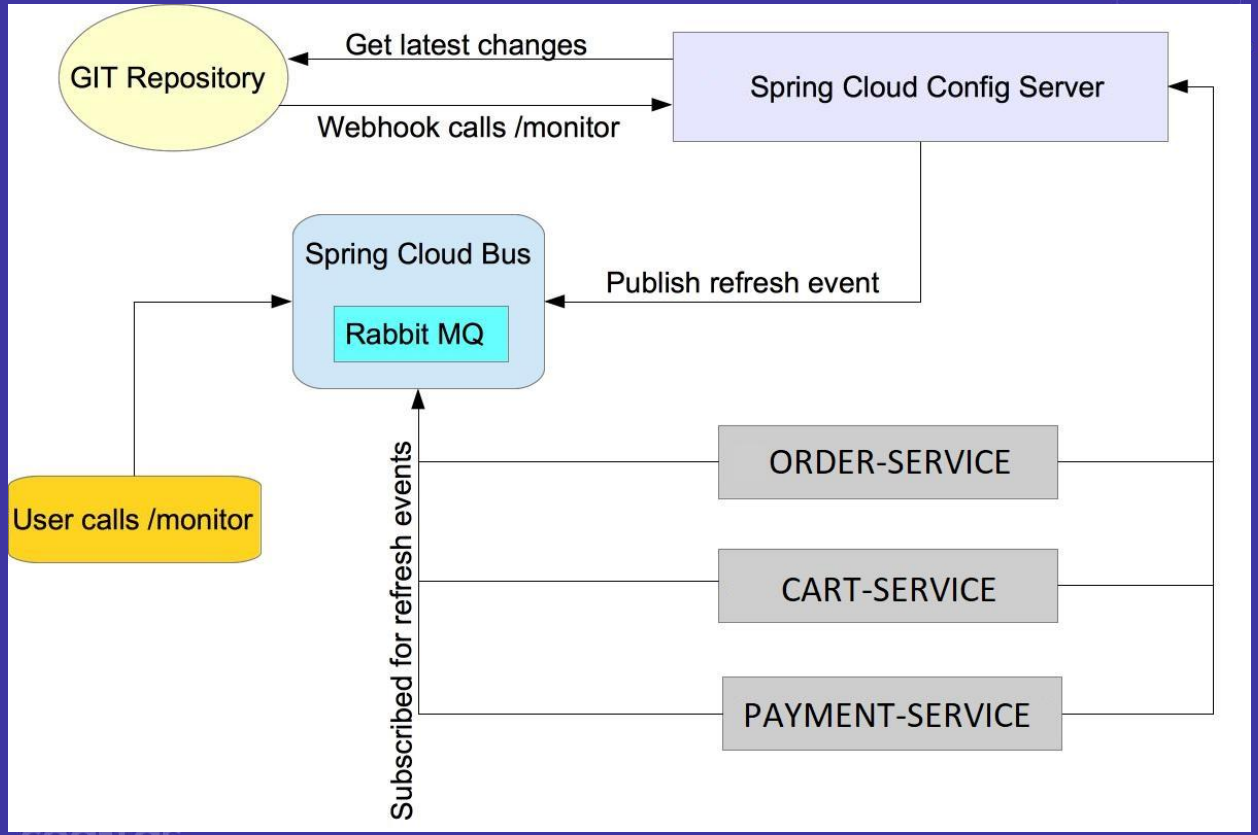
- Spring Cloud Config is **Spring's client/server** approach for storing and serving configurations across multiple applications and environments.
- This configuration store is ideally versioned under Git version control and can be modified at application runtime.



Configuration

- Spring cloud config server.
- Spring cloud config client.

Refreshable configuration - Spring cloud bus and spring cloud monitor.



Configuration

- Config Repo.
- Webhook configuration.

Declarative Rest Client - Feign

- Feign is a declarative web service client.
- Reduce boilerplate code.
- Annotate the interface with @Feign client.

Features

- logging(Debug)
- Exception handling(Error Decoder)
- Timeout(Connection and Read timeout)
- RequestInterceptor

Configuration

- Feign Declarative Rest Client.

load balancing

- using spring cloud load balancer

- Spring Cloud Netflix Ribbon has been deprecated and is not included in the 2020.0.0 release train.
- Ribbon requires Spring Boot Version >= 2.0.0.RELEASE and < 2.4.0-M1.
- Feign client uses spring cloud load balancer.
- It supports RoundRobinLoadBalancer and RandomLoadBalancer.
- We are planning to implement Weighted-Time-Response-Load-Balancer-algorithm.

Configuration

- Custom configuration class.
- Custom load balancer feign client configuration.
- View cart.

Fault tolerance - Resilience4j

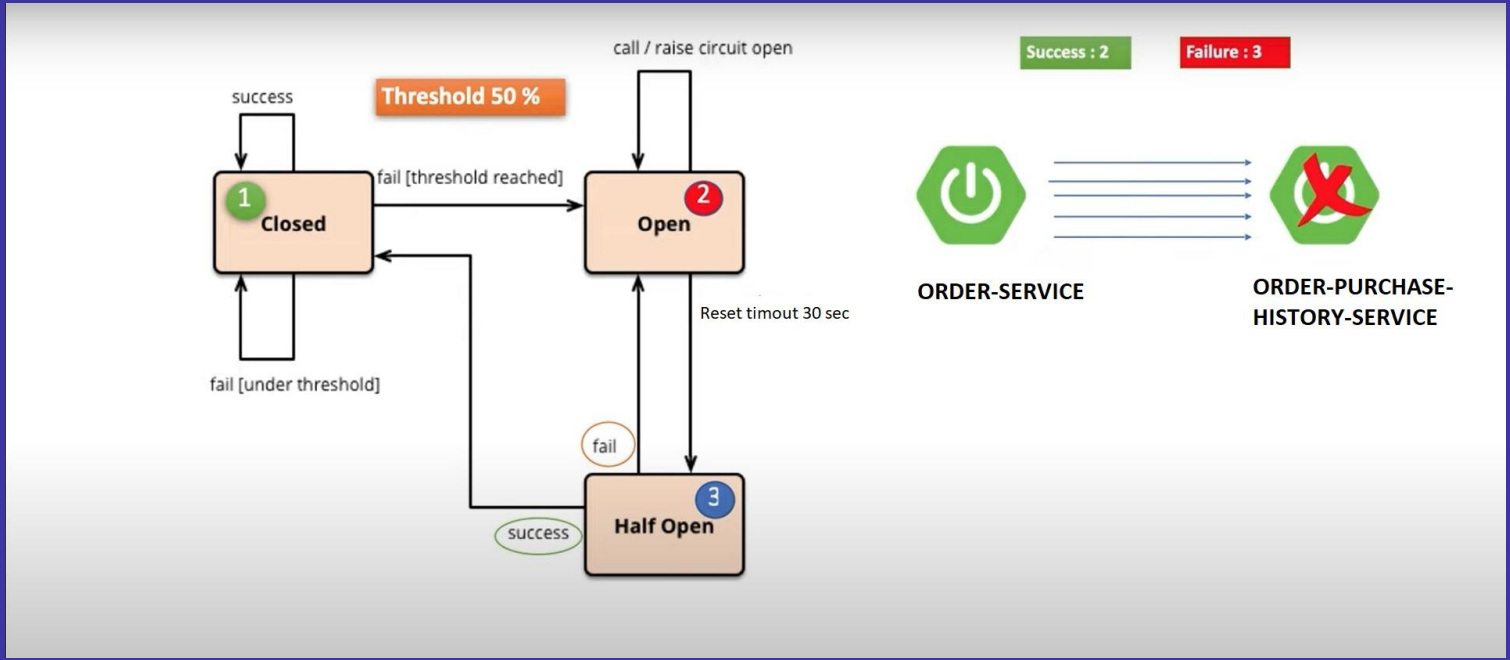
- Hystrix has been deprecated.

Features

- Circuit breaker
- Slow calls
- Retry
- Rate Limiter
- Bulkheading

Circuit breaker

The CircuitBreaker is implemented via a finite state machine with three states: **CLOSED**, **OPEN** and **HALF_OPEN**.



Distributed Tracing

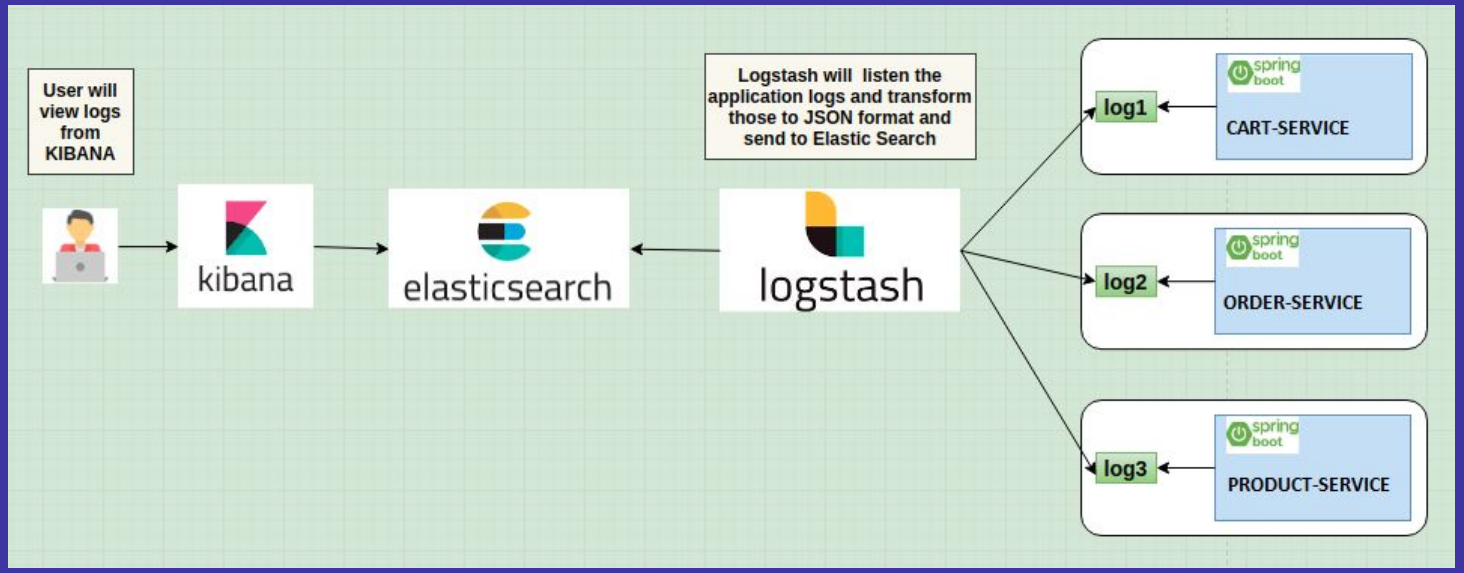
- **Sleuth** and **zipkin** are provided by spring cloud used for distributed tracing.
- **Sleuth** creates **traceid** and **spanid** to find execution path details and stores in temporary memory.
- **Zipkin client** collects data from the **sleuth** and sends it to the **Zipkin server**.

API calls.

- Success call.
- Failure call.
- Exception.

Centralized Logging - ELK Stack

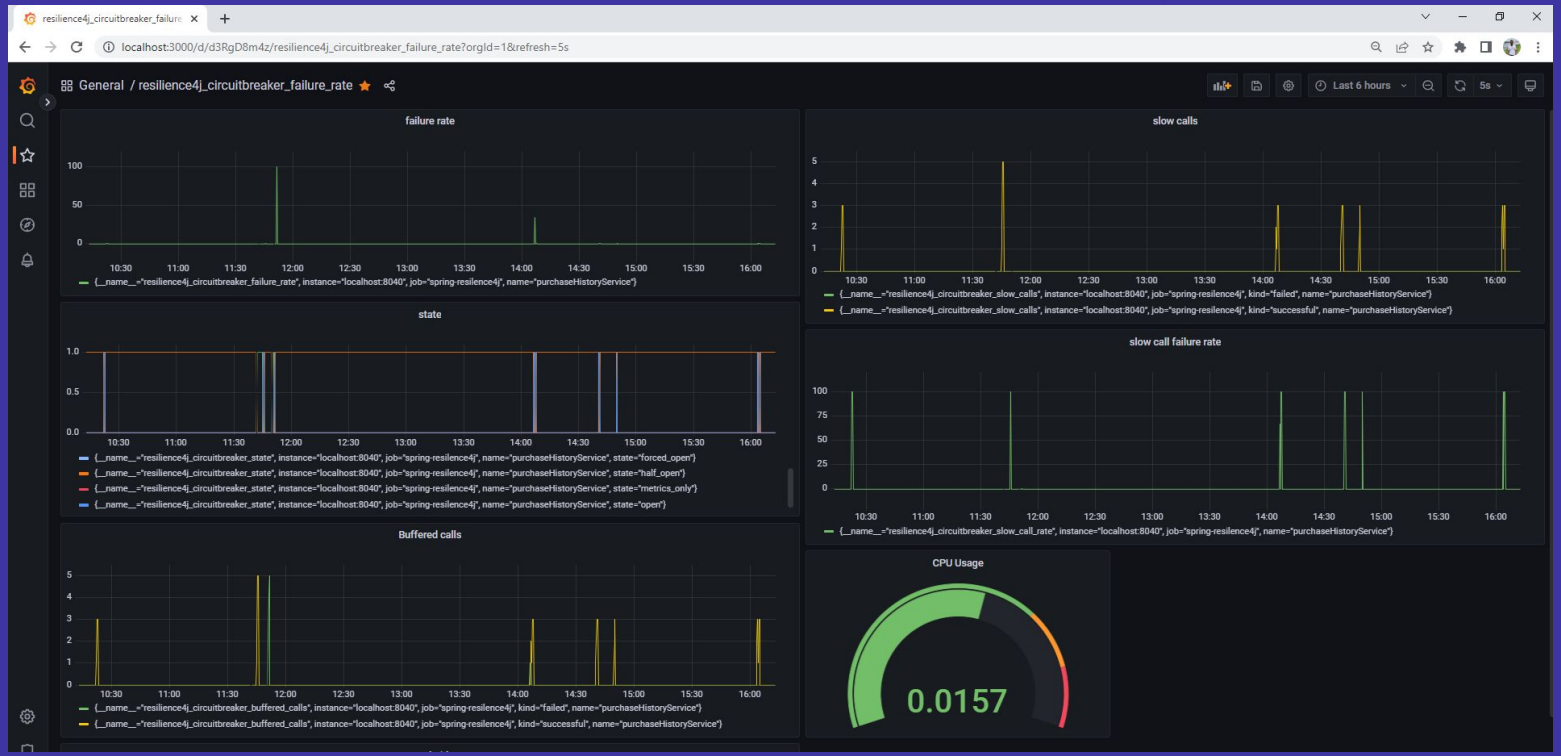
- The ELK Stack consists of three open-source products - Elasticsearch, Logstash, and Kibana from Elastic.
- Elasticsearch is a NoSQL database that is based on the Lucene search engine.
- Logstash is a log pipeline tool that accepts inputs from various sources, executes different transformations, and exports the data to various targets.
- Kibana is a visualization UI layer that works on top of Elasticsearch.



- Kibana dashboard.

Monitoring - Prometheus and Grafana

- Hystrix dashboard has been deprecated.
- It also provides support for alerting(Email,slack).
- Grafana dashboard.



coditas

Questions

coditas

What is Kafka?

Kafka is an event streaming platform. It provides 3 capabilities-

- publish/subscribe to events
- Store events durably
- Process streams of events

Advantages of Kafka over other Messaging Queues-

- Durability of messages
- Data Replication
- Multiple Subscribers
- Automatic Rebalancing of Consumers
- High Throughput
- Easy Scaling without any downtime.
- Pull based

Terminologies

Event	It is a record of something that happened.
Topic	A place where the events are stored. It is multi-producer & multi-consumer.
Partition	These are the sections within a topic. Events are actually stored in one or more partitions.
Producer	Client applications that can publish events to Kafka.
Consumer	Client applications that subscribe to the events for further processing.
Consumer Group	A set of consumers that cooperate to consume data from a topic.
Replica	Topics can be replicated to make data fault-tolerant and highly available.
Broker	It is the server that holds the actual partitions.

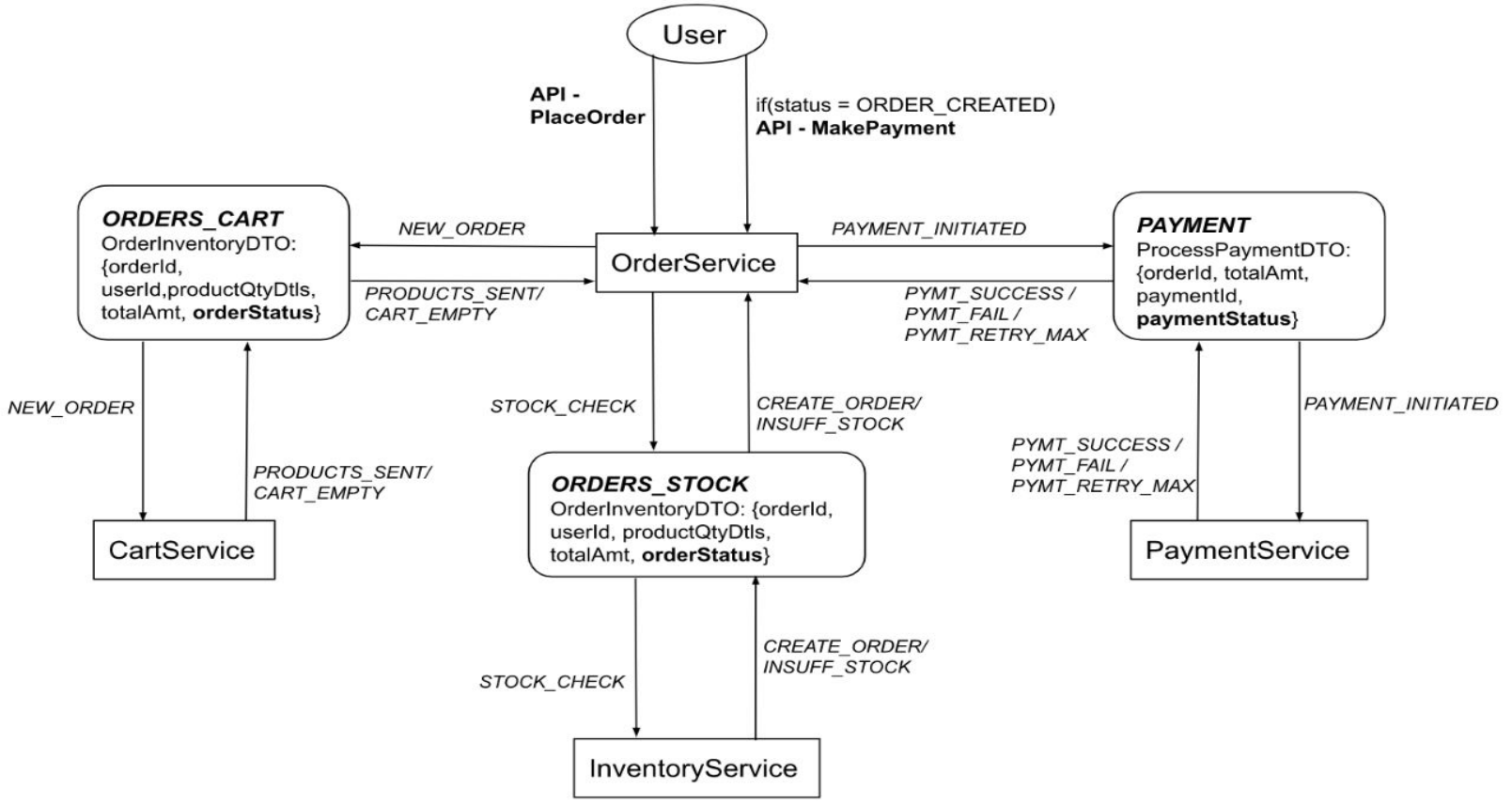
What is Zookeeper?

Zookeeper is used for coordination between distributed brokers of a Kafka cluster. It performs functions like -

- Controller Election
- Cluster Membership & Naming Service
- Cluster Control
- Topic Configuration
- Access Control Lists for Topics

Note: As of now Kafka cannot run without zookeeper, starting from v2.8.0 it will be possible but it is not available yet.

Application Flow



Spring Kafka

Spring Kafka Configuration

```
spring.kafka.bootstrap-servers= localhost:9092
spring.kafka.producer.key-serializer= org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer= org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.consumer.key-deserializer= org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.properties.spring.json.trusted.packages = com.microservices.*
```

Topic Configuration

```
@Bean
public NewTopic createOrderRequestTopic(){
    return TopicBuilder.name(TopicNames.ORDERS_CART)
            .build();
}
```

Producing Messages

```java
@Service
public class PaymentProducer {

    1 usage
    private Logger logger = LoggerFactory.getLogger(PaymentProducer.class);

    1 usage
    @Autowired
    private KafkaTemplate<String, ProcessPaymentDTO> orderKafkaTemplate;

    1 usage   Sejal *
    public void publishPaymentStatus(ProcessPaymentDTO processPaymentDTO){
        logger.info("Published to "+ TopicNames.PAYMENT+" : "+processPaymentDTO);

        orderKafkaTemplate.send(TopicNames.PAYMENT,processPaymentDTO);
    }
}
```

Consuming Messages

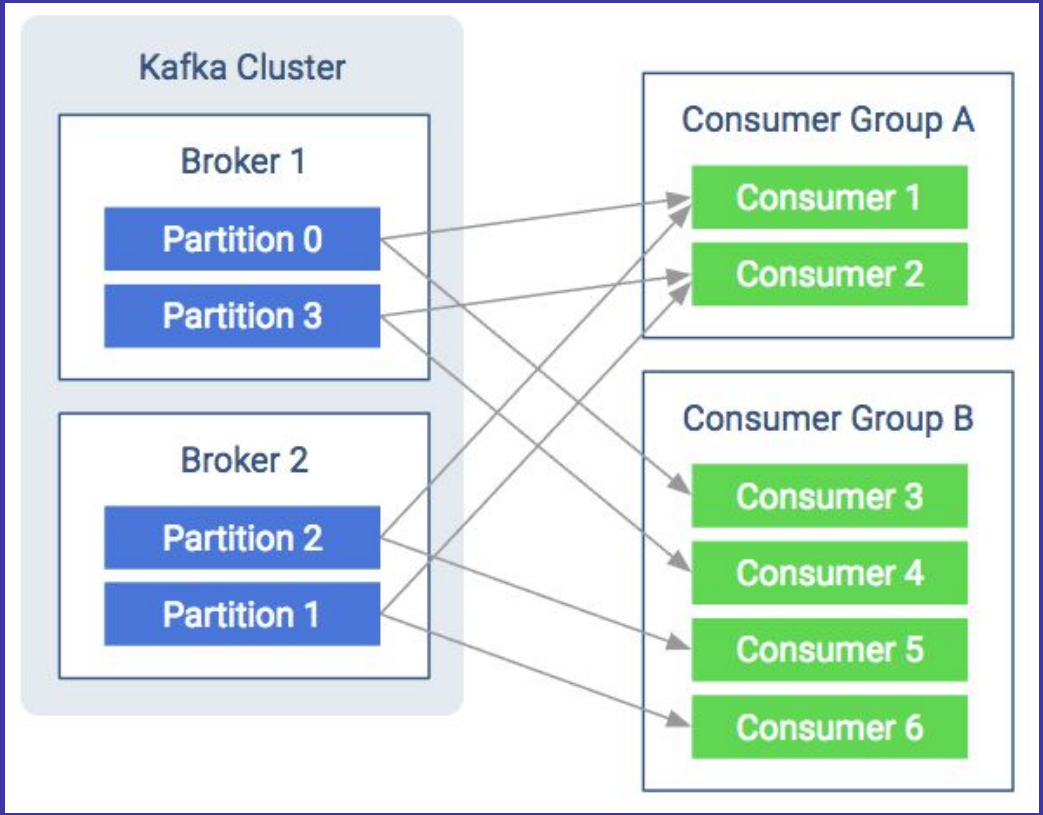
```java
@Service
public class PaymentConsumer {
    2 usages
    @Autowired
    private PaymentService paymentService;

    1 usage
    private Logger logger = LoggerFactory.getLogger(PaymentConsumer.class);

    Sejal *
    @KafkaListener(topics = TopicNames.PAYMENT, groupId = "orderConsumer")
    public void processPayment(ProcessPaymentDTO processPaymentDTO){
        if(processPaymentDTO.getPaymentStatus().equals(PaymentStatus.PAYMENT_INITIATED)) {
            logger.info("PaymentService subscribed from "+TopicNames.PAYMENT+" : "+processPaymentDTO);

            paymentService.processPayment(processPaymentDTO);
            paymentService.publishPaymentStatus(processPaymentDTO);
        }
    }
}
```

Kafka Consumers & Partitions



coditas

How Kafka manages consumers

Group Coordinator

- One of the brokers is coordinator.
- Consumer sends joinGroup request to coordinator.
- Maintains list of all consumers in a group.
- Sends this list to leader on change.

Group Leader

- First consumer to join the group.
- Handles partition assignment using PartitionAssigner.
- Has list of all members and their partitions.

Group Members

- All members of group send heartbeat to coordinator.
- Properties - session.timeout.ms (def 10s), heartbeat.interval.ms(⅓ of session timeout)

Rebalancing

- Eager Rebalancing (old versions of Kafka)
- Incremental

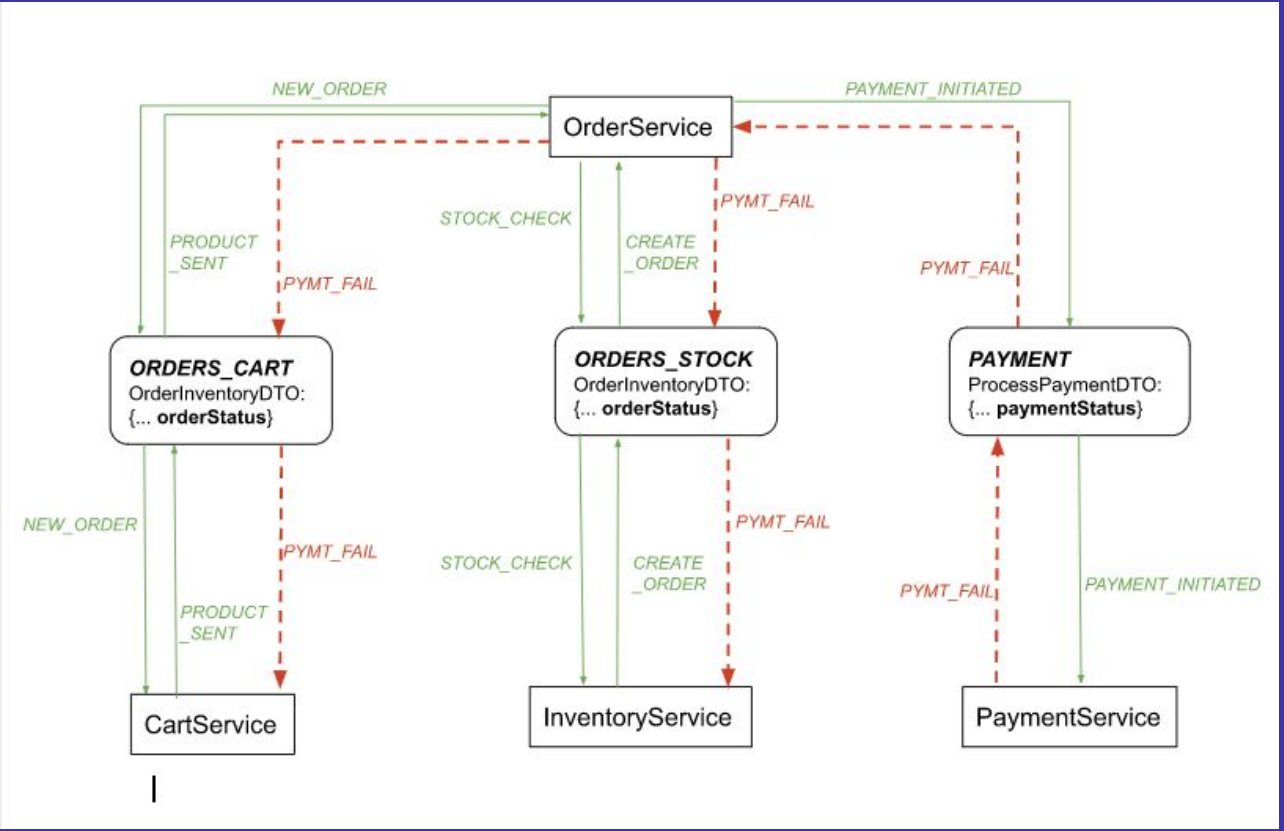
Saga Design Patterns

Saga pattern is a way to maintain data consistency among distributed applications, especially in case of failure.
Two saga patterns -

- Orchestration
- Choreography

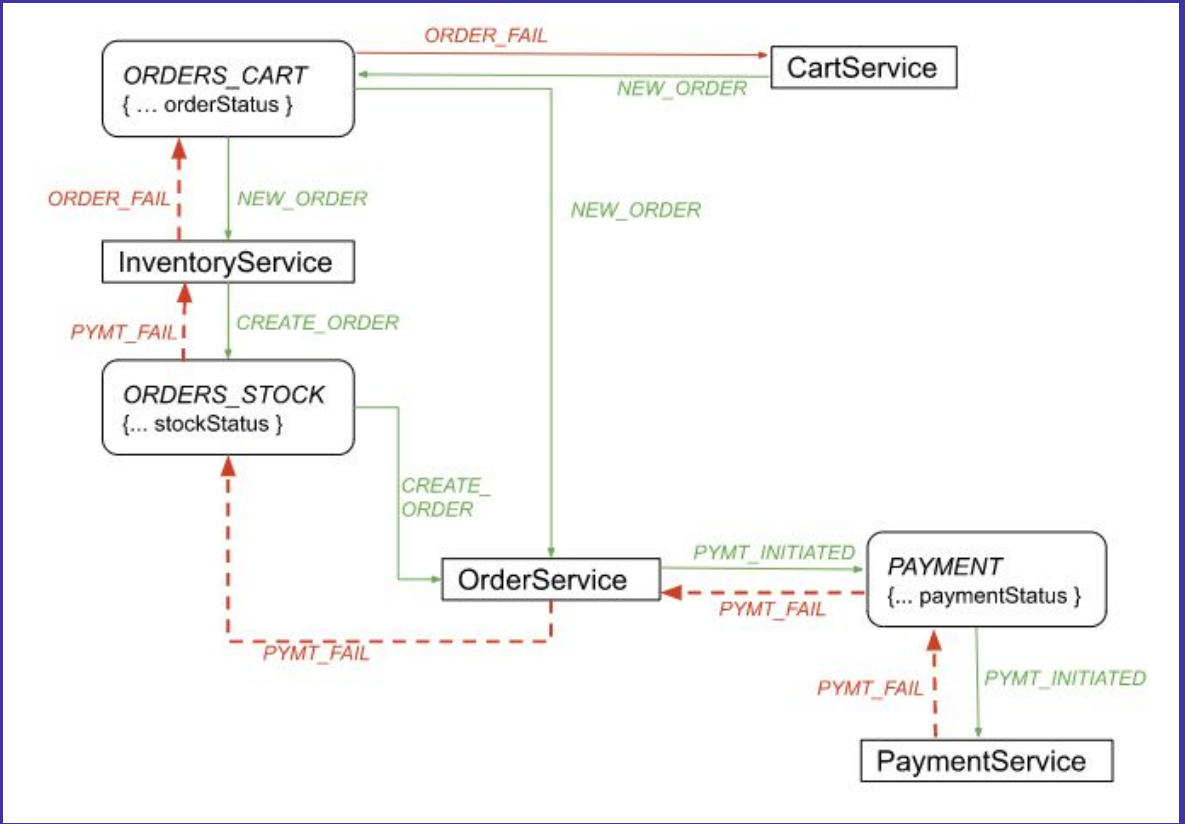
Alternative Pattern - 2 Phase Commit
Disadvantages of 2PC - Blocking and synchronous

Saga Orchestration Example



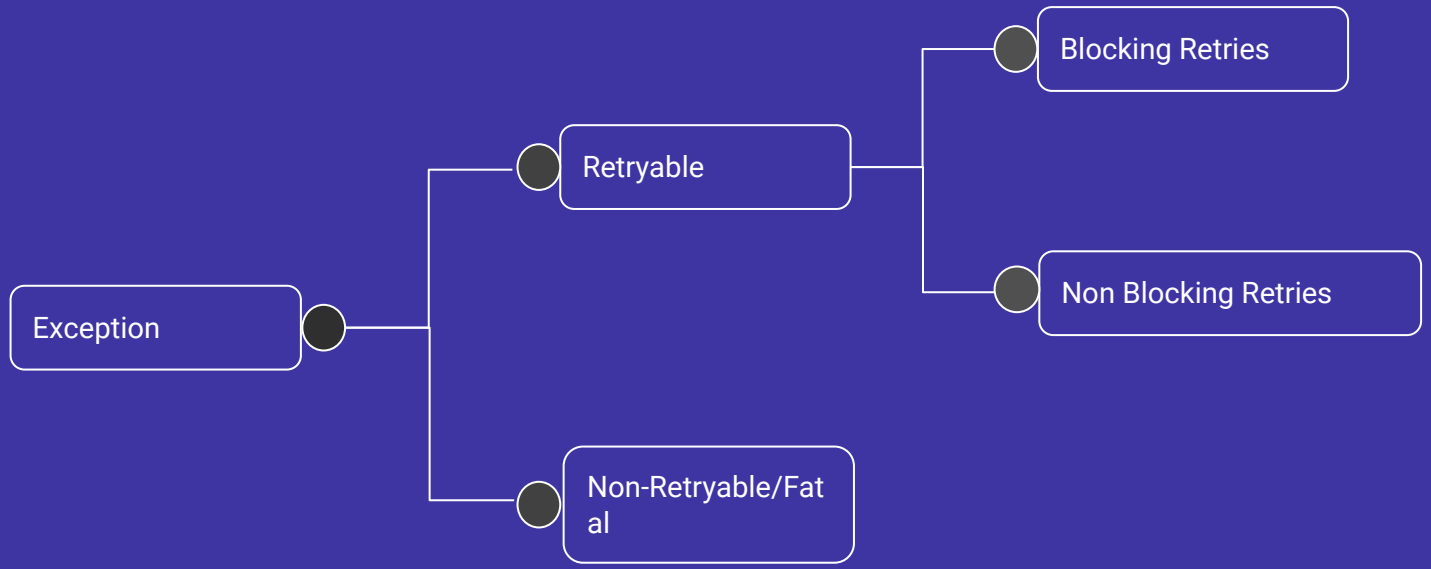
coditas

Saga Choreography Example



coditas

Exception & Error Handling



Retryable Exceptions

These are failures that might be successful on retrying after some time. It can be achieved in 2 ways-

- Blocking - *RetryTopicConfigurationSupport* should be extended and *configureBlockingRetries()* should be overridden
- Non-Blocking - *@RetryableTopic* annotation along with Backoff policy can be used.

```
@RetryableTopic(attempts = "3", backoff = @Backoff(delay = 3000, multiplier = 6))
@KafkaListener(topics = TopicNames.CART_STOCK, groupId = "inventoryConsumer")
public void addToCart(StockStatusDTO stockStatusDTO){
    if (stockStatusDTO.getStatus().equals(StockStatus.STOCK_AVAILABLE)){
        logger.info("Subscribed from "+TopicNames.CART_STOCK+" : "+stockStatusDTO);
```

```
@DltHandler
public void DLTCart(StockStatusDTO stockStatusDTO) {
    logger.info("Event moved to DLT: " + stockStatusDTO);
}
```

Non-Retryable Exceptions

These exceptions need to be handled using Error Handlers available in the spring-kafka project.

- Listener level error handler - ConsumerAwareListenerErrorHandler
- Container level error handler - CommonErrorHandler

```
@Service(value = "orderAmountErrorHandler") @Slf4j
public class OrderErrorHandler implements ConsumerAwareListenerErrorHandler {
    Sejal *
    @Override
    public Object handleError(Message<?> message,
                              ListenerExecutionFailedException exception,
                              Consumer<?, ?> consumer) {
        log.warn("Error in placing order : {}, because: {}, with headers:{}",
            message.getPayload(), exception.getMessage(),message.getHeaders());
        throw exception;
    }
```

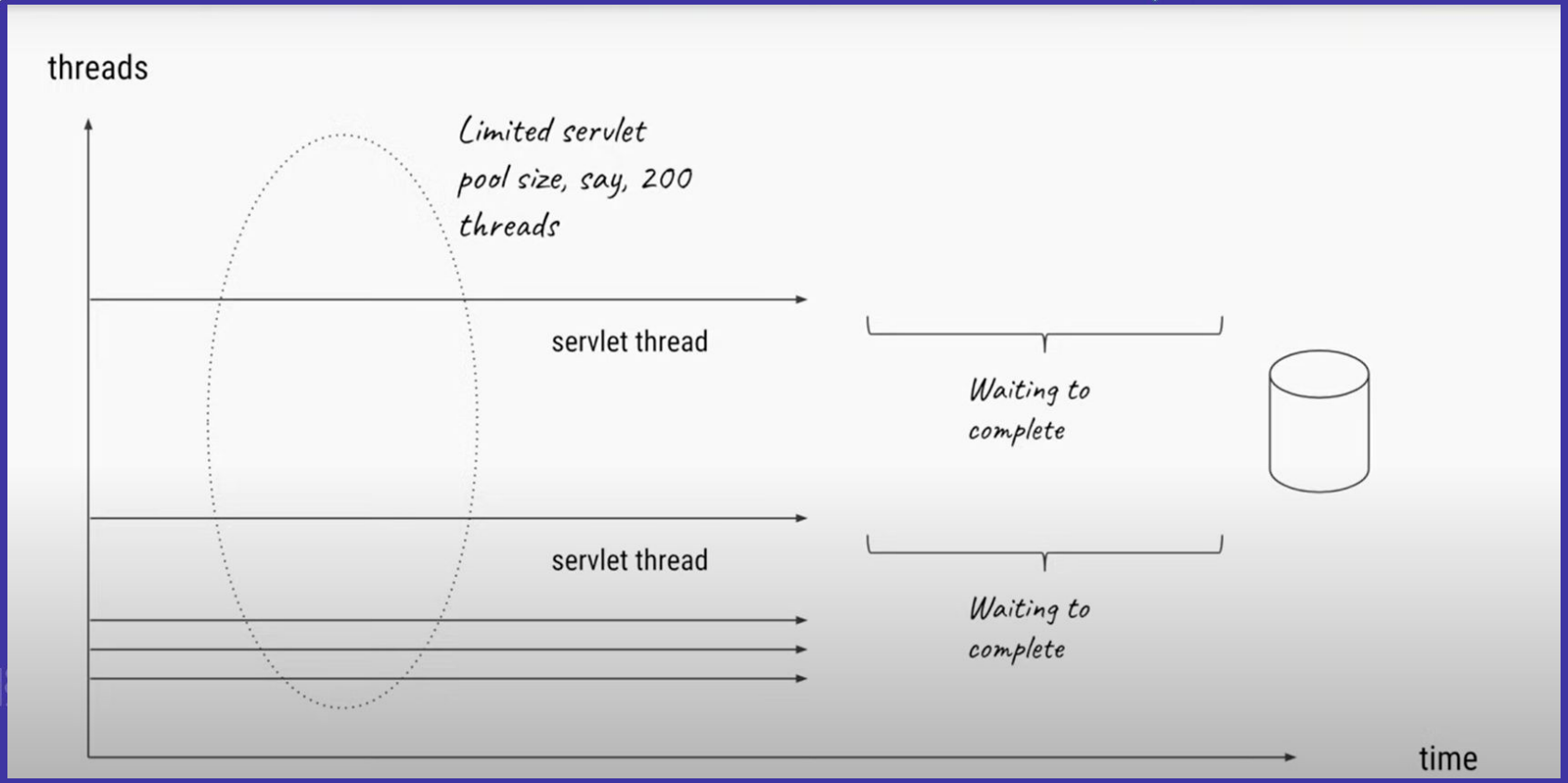
Reactive Feign Client

- The API that we defined via Feign is synchronous — it makes blocking calls to the server.
- Reactive Feign Client is Implementation of Feign which uses Spring WebClient.
- WebClient supports both synchronous and asynchronous calls.
- using WebClient everything will be as usual, but every method must return Mono or Flux.

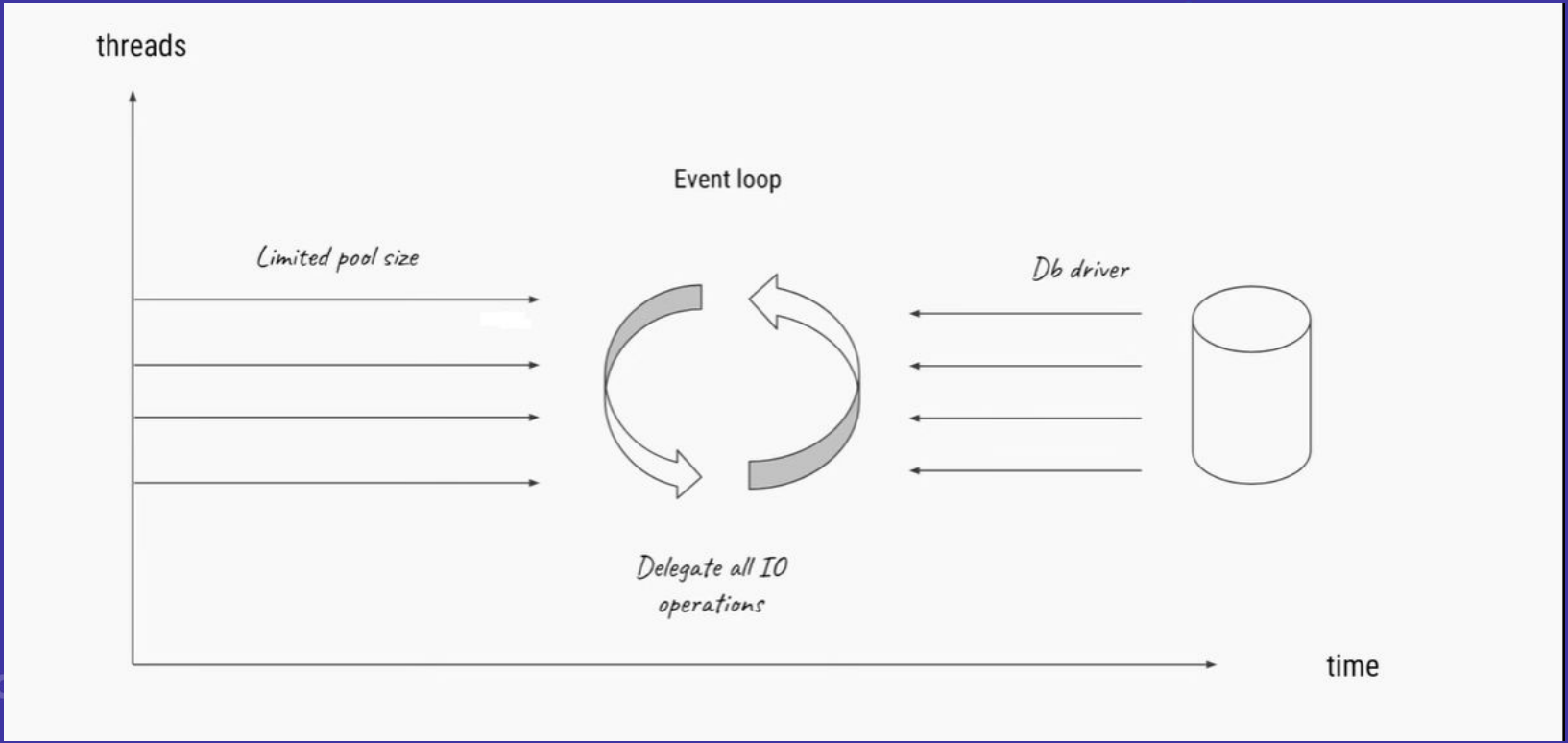
Required Dependency.

Changes in Feign Client

Blocking / synchronous call



Non-Blocking / Asynchronous call



Future Scope

- API Gateway
  - Protocol Adapter
  - Security
  - Authentication
  - Authorization
  - Adapter
  - Static Content
  - Response Cache
  - Billing
  - Filtering
- Kafka
  - Kafka Streams
  - KConnect
  - KSql DB
- Reactive Repository
  - R2DBC
- Cloud Services
  - API Gateway
    - AWS
    - Azure
    - Google Cloud Endpoints
    - Apigee
  - Services
    - Iaas
    - Saas
    - Paas
  - Cloud
    - Private Cloud
    - Public Cloud
    - Hybrid Cloud
    - Serverless
- Service Mesh
  - Istio & Envoy (Sidecar Pattern)

Thank you

www.coditas.com

coditas