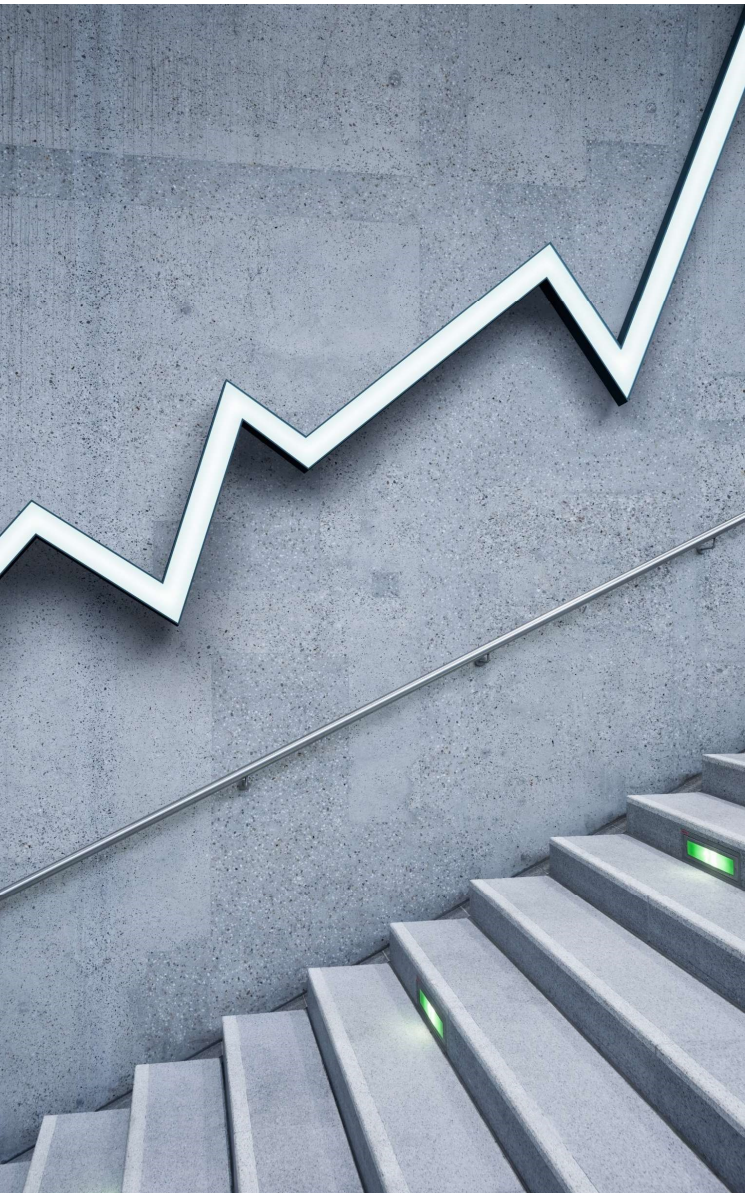


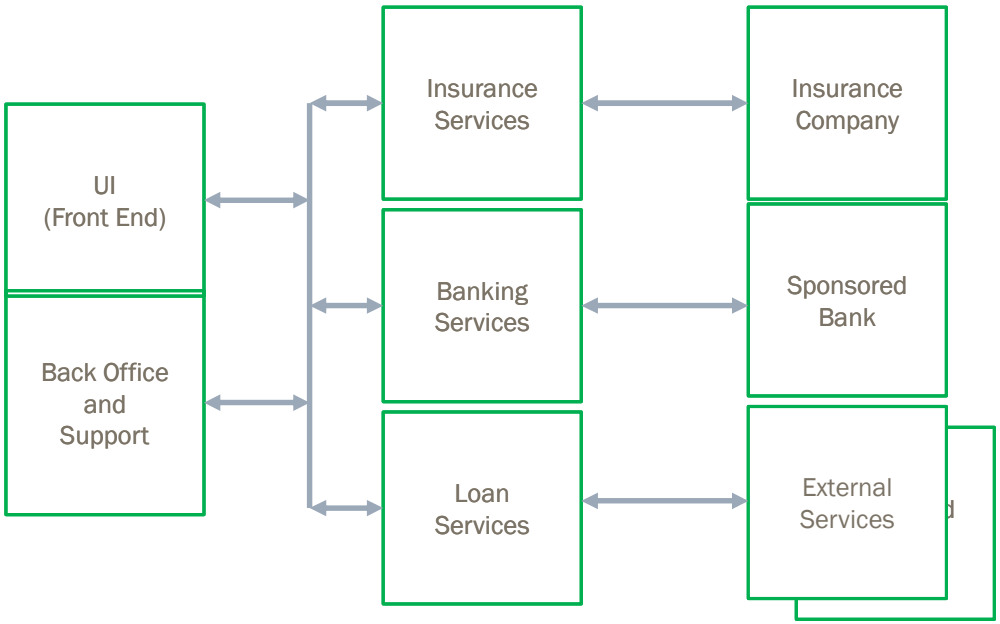
Message Driven Architecture

with KAFKA

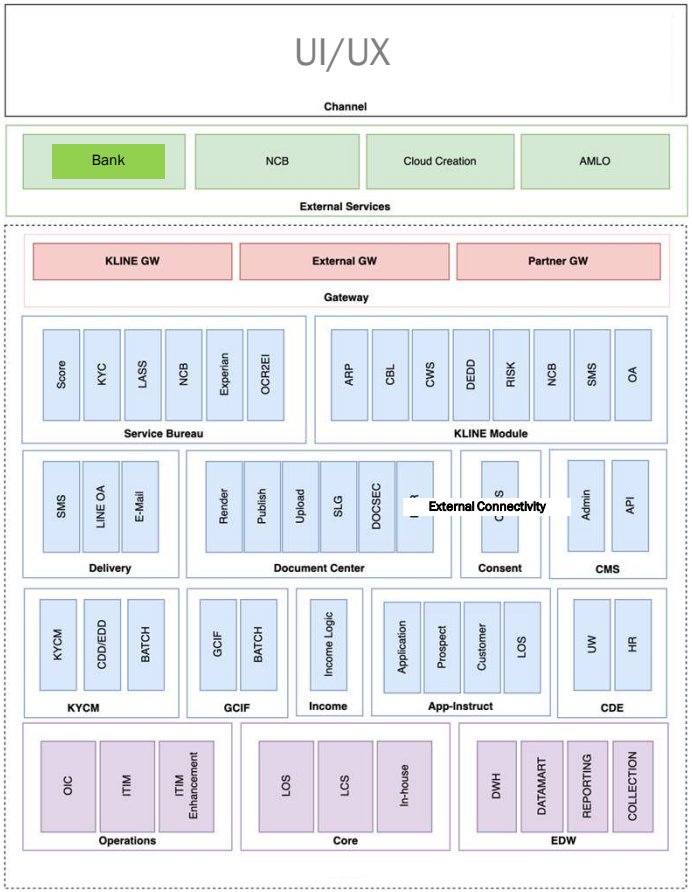
For KMITL 13016385, Distributed Computing

BY CHATCHAI DUSADENOAD

Solution Structure



High Level Structure

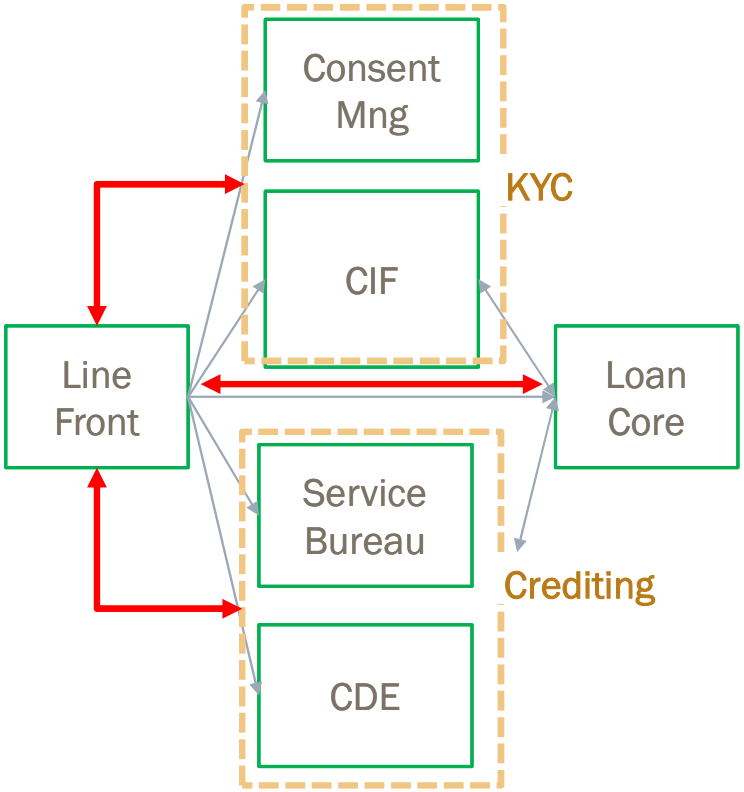


Detail Components

Non-Functional Requirements

- High Availability (near 100%, 24x7)
- Scalability
- Change Resiliency
- Low RTO (Return to Operation)
- Low RPO (Recovery Point Objective)
- Quick UI/UX
- Low cost solution
- Support automated operation

Solution Overview – Lending Services



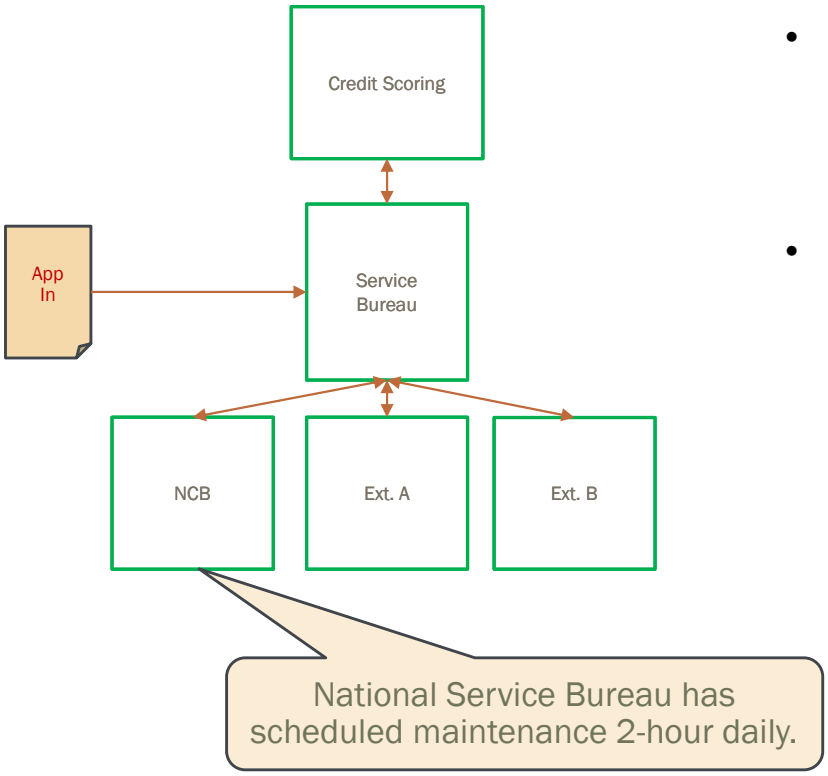
3 Main Services

1. Customer Data
2. Loan Application
3. Loan disburse and repay

3 Main Challenges

1. High Traffic in with some slow processing/temporary outages nodes
2. Low RPO or Zero Loss
3. Fluctuated workloads

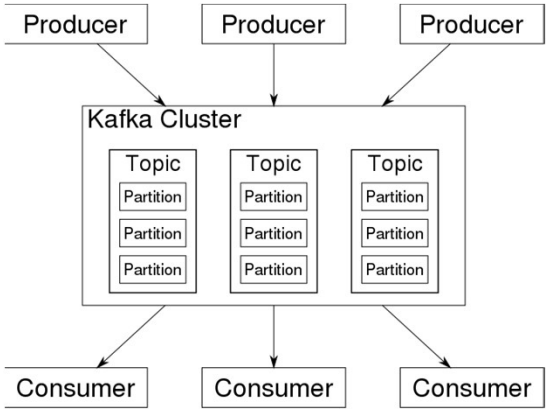
Solution Decision – Is it simple Message Queue system?

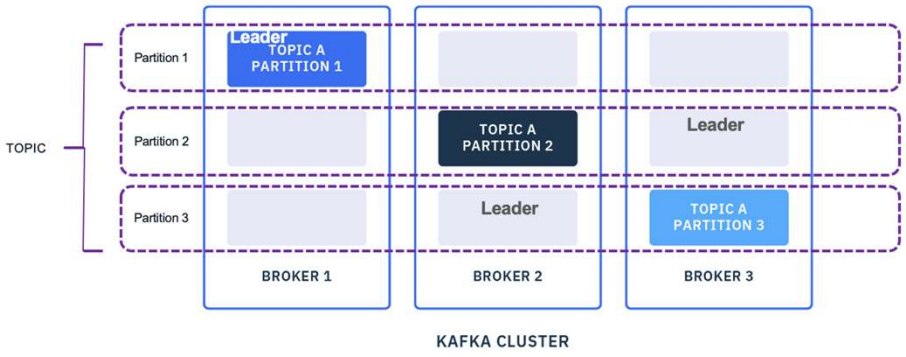


- The loan application is handled in multiple steps with several drop off points. -> multiple queues based on each stage of credit evaluations.
- The external services will parallelly process and respond on their own capacity; response time various from milli-seconds to days. -> very complex to manage queue length and response time.

Message with state driven should be better fit.

Let's Kafka





Brokers and Clusters
Kafka runs on clusters. Each cluster consists of multiple servers, generllay called brokers (and sometimes called nodes). That's what makes a Kafka a distributed system: data in the Kafka cluster is distributed amongst multiple brokers. And multiple copies (replicas) of the same data exists in a Kafka cluster.

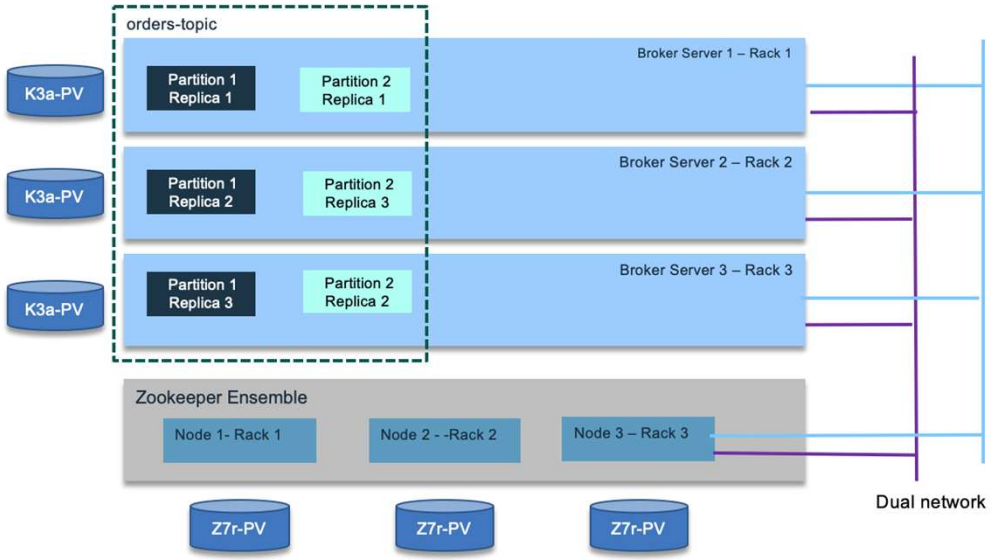
Topics
A Kafka topic is an immutable log of events (sequence). Producers publish events to Kafka topics; consumer subscribe to topic to access their desired data. Each topic can serve data to may consumers.

Partitions
A partition is the smallest storage unit in Kafka. Partitions serve to split data across brokers to accelerate performance. Each Kafka topic is divided into partitions and each partition can be placed on a separate broker.

Producers
A producer is anything that creates data. Producers constantly write events to Kafka. Examples of producers include web servers, other discrete applications (or application components), IoT devices, monitoring agents, and so on.

Consumers
Consumers are entities that use data written by producers. Sometimes an entity can be both a producer and a consumer; it depends on system architecture.

Parallel and Fault Tolerant in Kafka

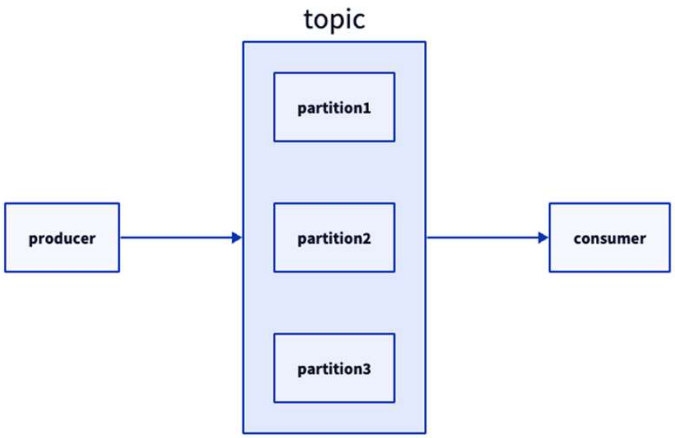


The record key in each record sent by producer, determines the partition allocation in **Kafka**: the records with the same key (hashcode of the key to be exact) will be in the same partition (It is possible to use more complex behavior via configuration and even code).

Kafka is keeping its cluster states in Apache Zookeeper. From version 2.8, it is possible to do not use Zookeeper, but not yet for production deployment: the brokers are using a Raft quorum algorithm, and an internal topic to keep state and metadata (See the process.roles property from the broker configuration).

Writes to Zookeeper are only be performed on changes to the membership of consumer groups or on changes to the Kafka cluster topology itself. It is possible to have a unique zookeeper cluster for multiple kafka clusters. But the latency between Kafka and Zookeeper needs to be under few milliseconds (< 15ms) anyway.

On the programming Example



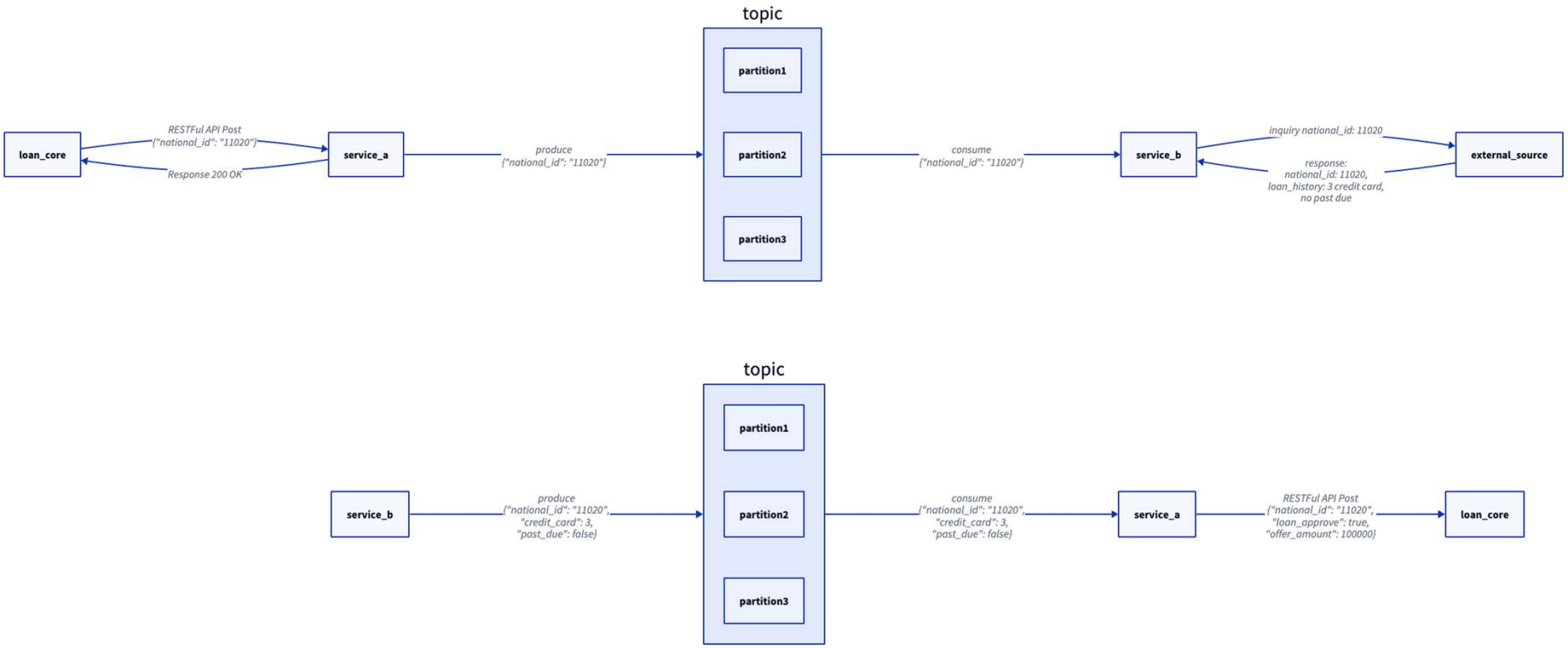
We use Kafka as an event-driven microservice.

When we need to check the eligibility for an application, the loan core system will send the API request to our service.

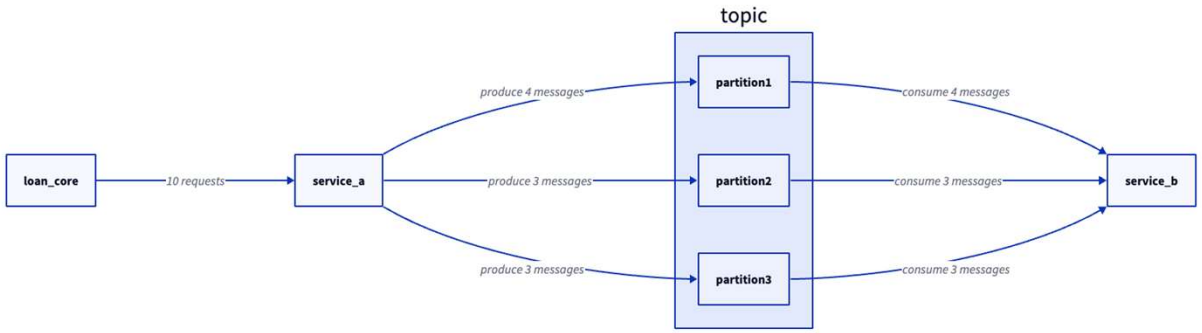
Our service will verify that the request contains all necessary information, like the ID card number, name, and last name, and then it will return to the loan core system, where we will add this to the queue (Response 200 OK).

Then the service will produce the message on the Kafka topics, and the service that needs to check that message will consume that message to process until it is finished. It will then produce the result back.

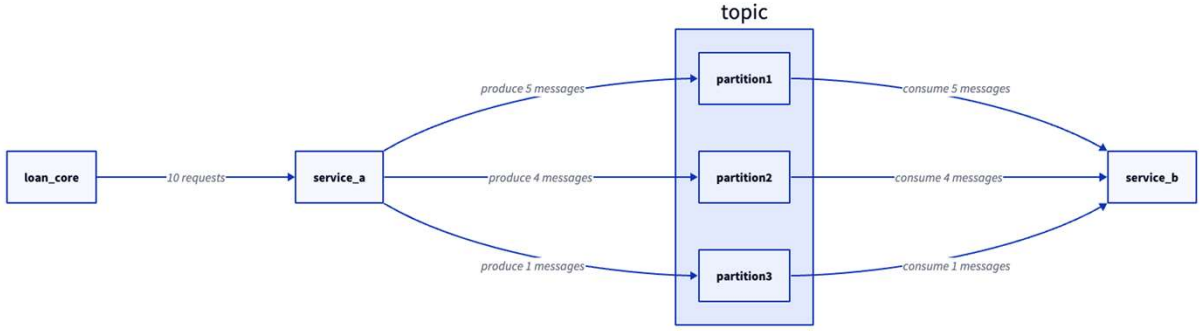
Message Exchange Example



Message Exchange Example

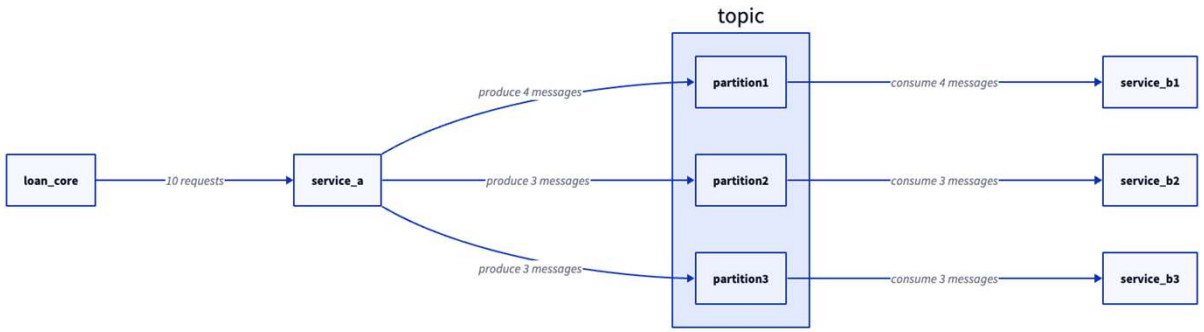


By default, Kafka Topic will generate 3 partitions for each topic; you can add more partitions if needed.



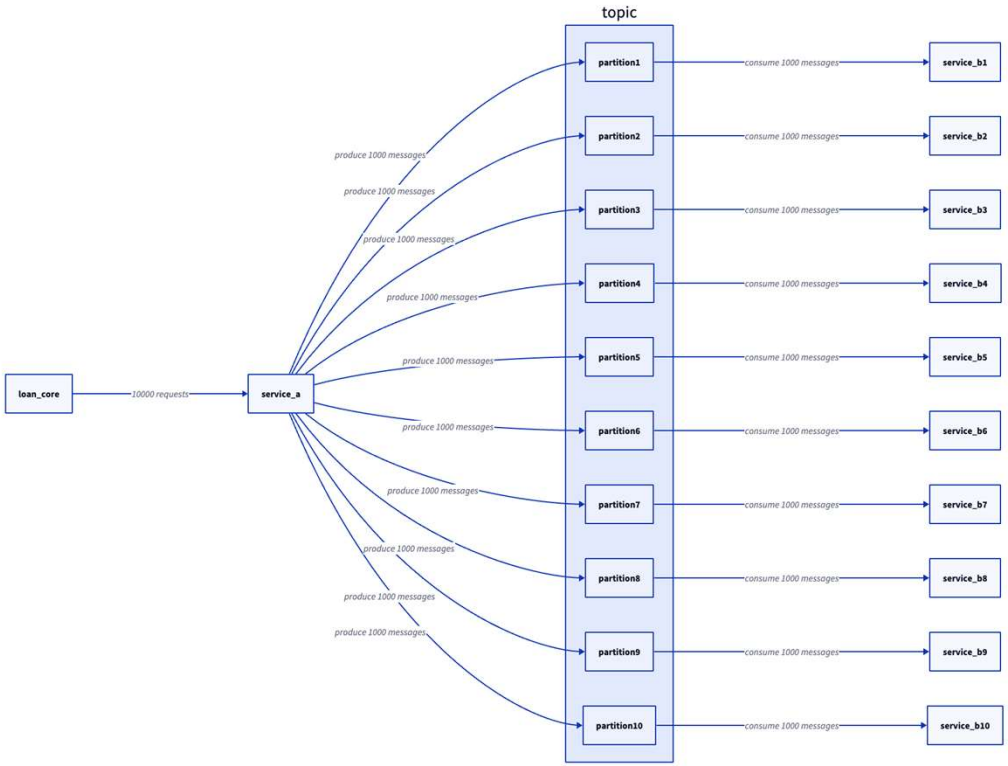
You can specify the message that should be produced to which partitions; if you don't specify, Kakfa will use the round robin method.

Message Exchange Example



We can add more consumers to reduce the workload from 1 consumer, and it will process in parallel for each consumer connected to each partition.

Message Exchange Example



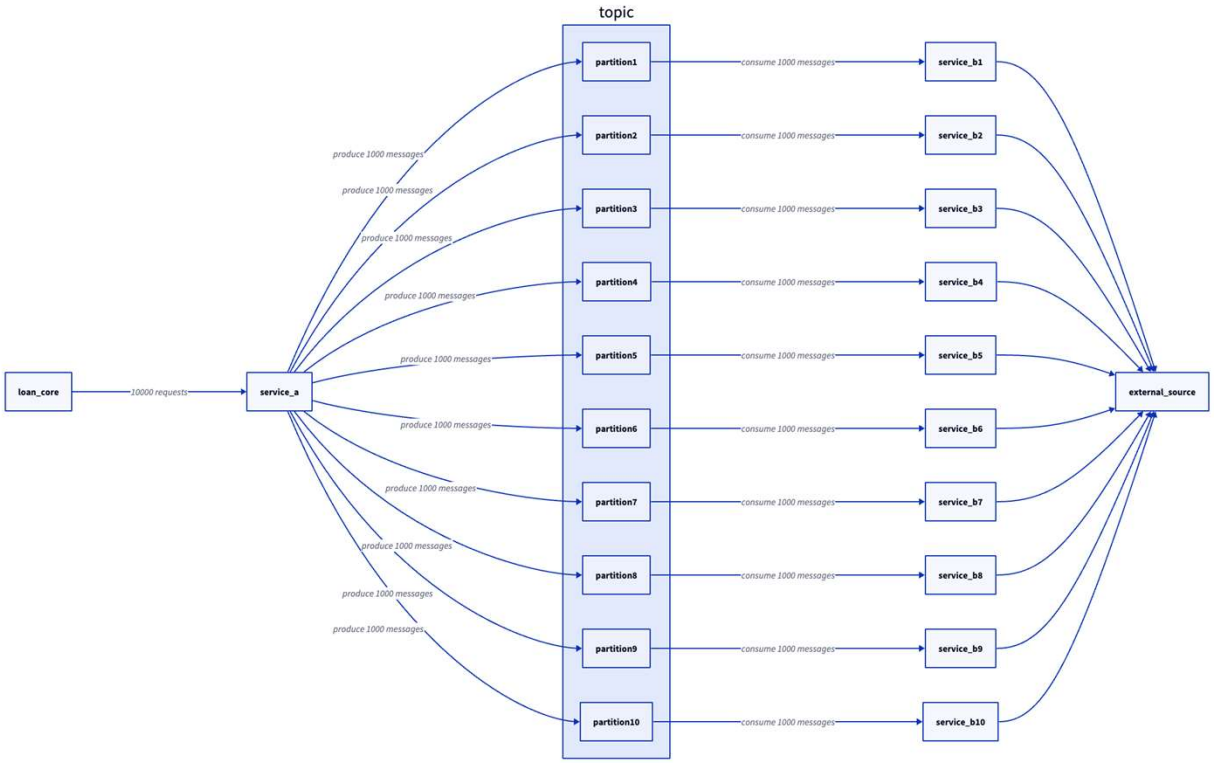
Example: We have 10,000 applications that require checking. Each app required 2–3 seconds to process.

So, it means we require 8 hours to complete 10,000 applications by 3 partitions and 1 consumer.

But if you want to complete it within 1 hour, we just scale the partitions from 3 to 10 and add consumers up to 10.

3 x 1000 seconds = 50 minutes.

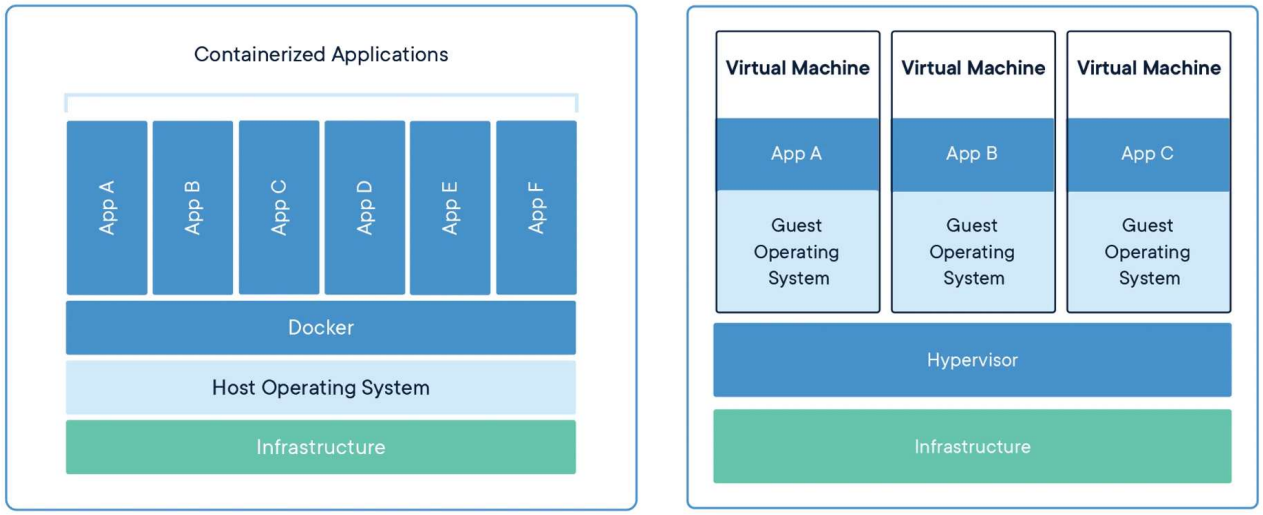
Message Exchange Example



We also use Kafka to reduce workloads on other systems.

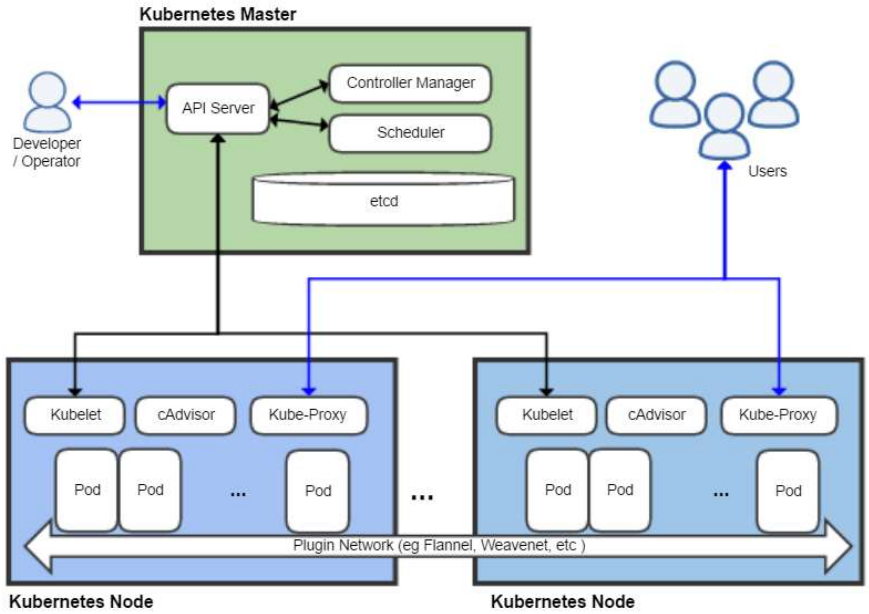
Example: If we have 10 partitions and 10 consumers, it will create 10 requests in parallel to the destination source.

Docker Container



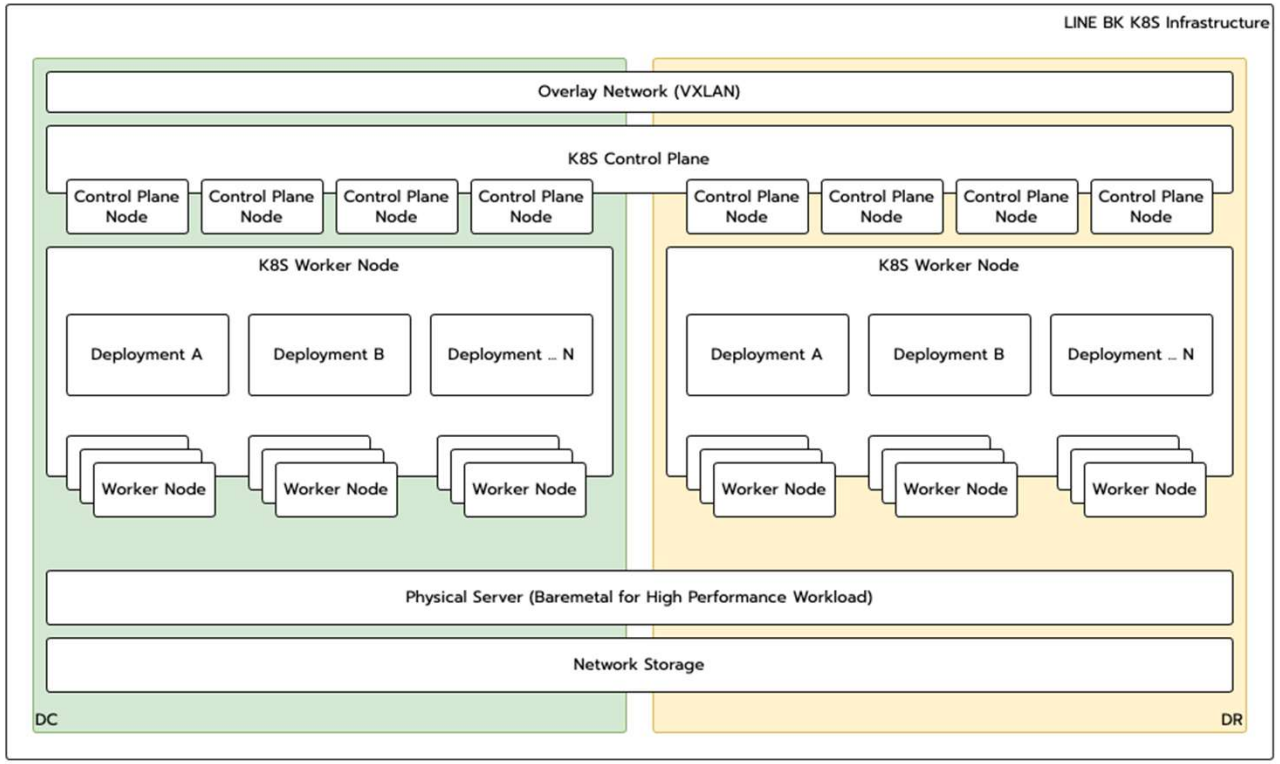
A container is an isolated environment for your code. This means that a container has no knowledge of your operating system, or your files. It runs on the environment provided to you by Docker Desktop. Containers have everything that your code needs in order to run, down to a base operating system.

Kubernetes



Kubernetes (also known as k8s or “kube”) is an open source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications.

Docker + K8S = Scalability

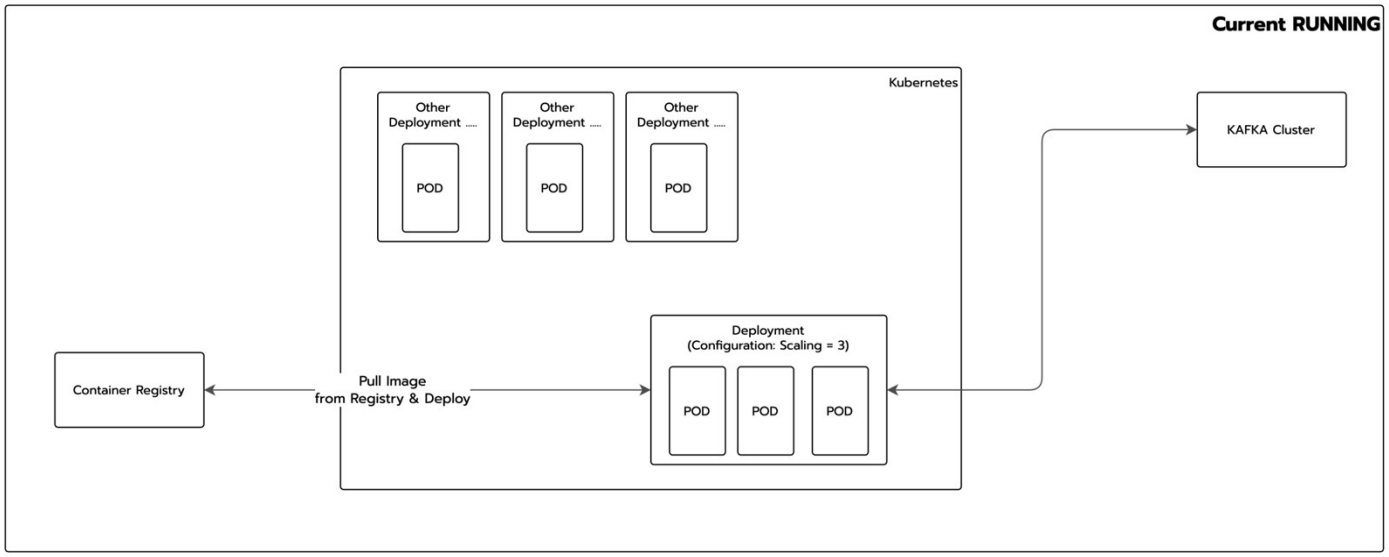


Docker + K8S In Action

Using 3 containers to process the message from KAFKA. When there are burst in transactions from customer and the Kafka processing starting a lag events (message inside Kafka is high), there are 2 steps in response;

1. Increase the consumer node (Scaling deployment from current 3 to 10) (Remark: numbers of pod is a calculated from developers/infra team)

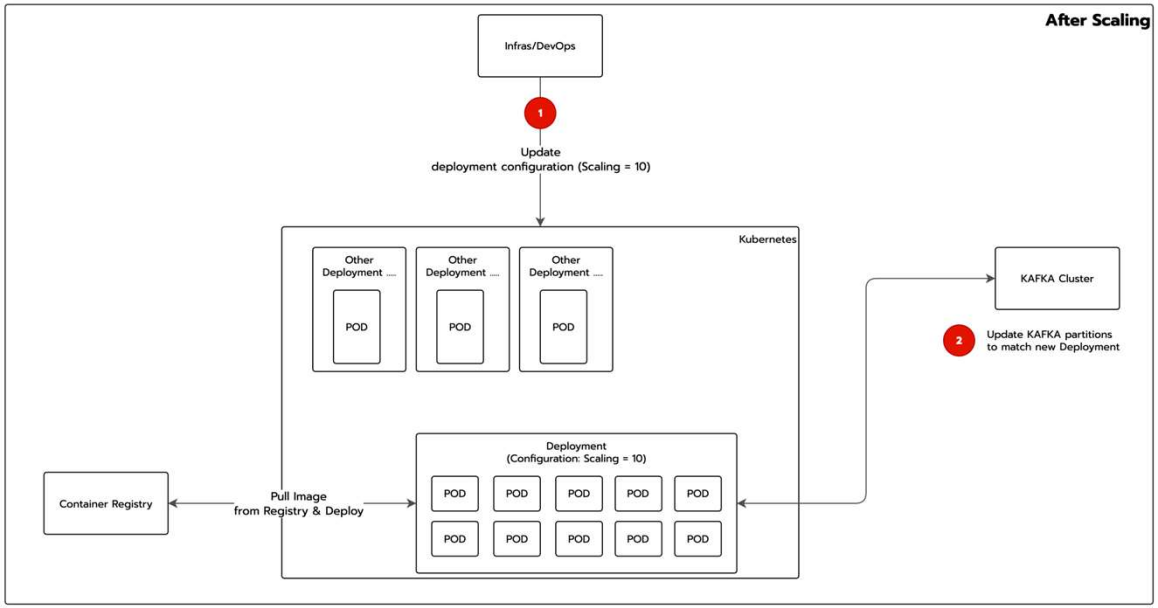
2. Reconfiguration of number of partitions in Kafka to match consumer



Docker + K8S In Action

After Step1 and Step2 in above the result of cluster will be below.

Then we monitor the Kafka message is still lag or not if still lag, we will do the estimation and re apply with the new numbers



Stacking for High Availability