14-848 Cloud Infrastructure

APACHE KAFKA

Agenda

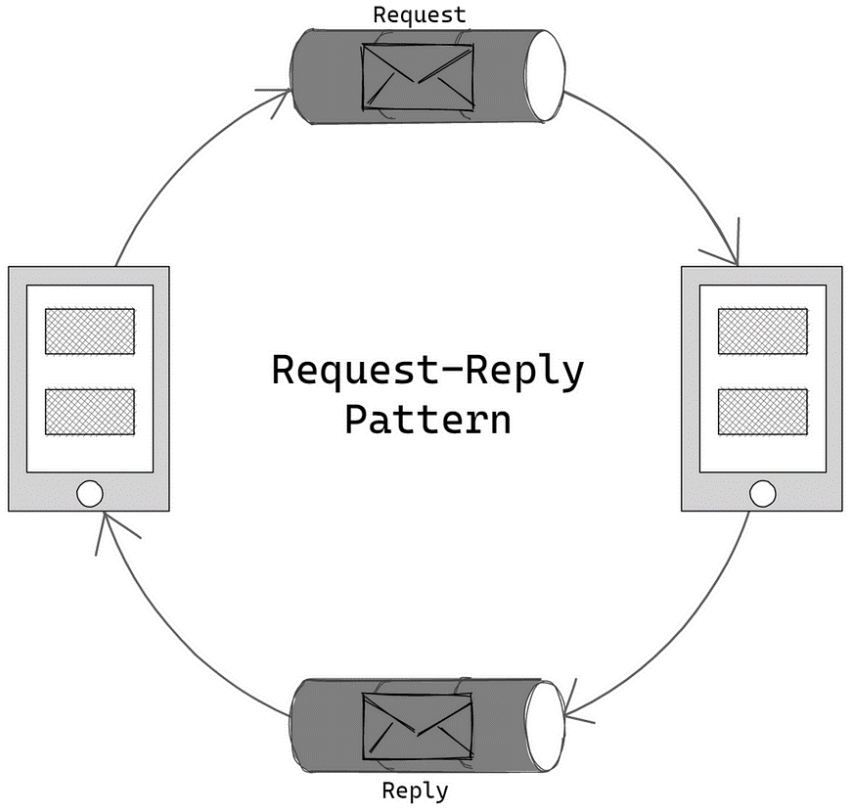
- Announcements
- User-response Architectures
- Apache Kafka
- Kafka & Microservices
- Confluent-Kafka

Announcements

- Course Project is Due by this Thursday.
  - Plan to submit ahead of time to avoid last-minute issues
  - Read the submission guidelines carefully
- HW-7 is released on Wednesday.
  - HW-7 is due by November 21st

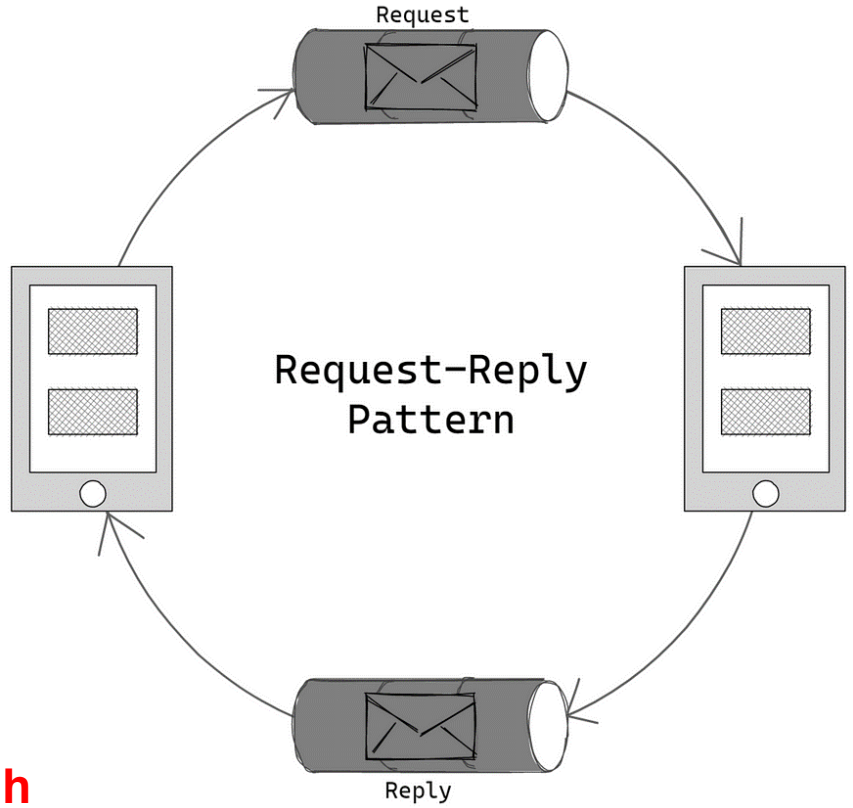
Request-Response Architecture

- Your Machine Learning Infrastructure offers the ability to train models based on incoming training requests.
- What happens if the number of training requests increases?
    - Your Machine Learning infrastructure can leverage distributed training environment to process multiple requests simultaneously.
- Well, what happens if the number of user requests continue to increase beyond your ML infrastructure resources?



Request-Response Architecture

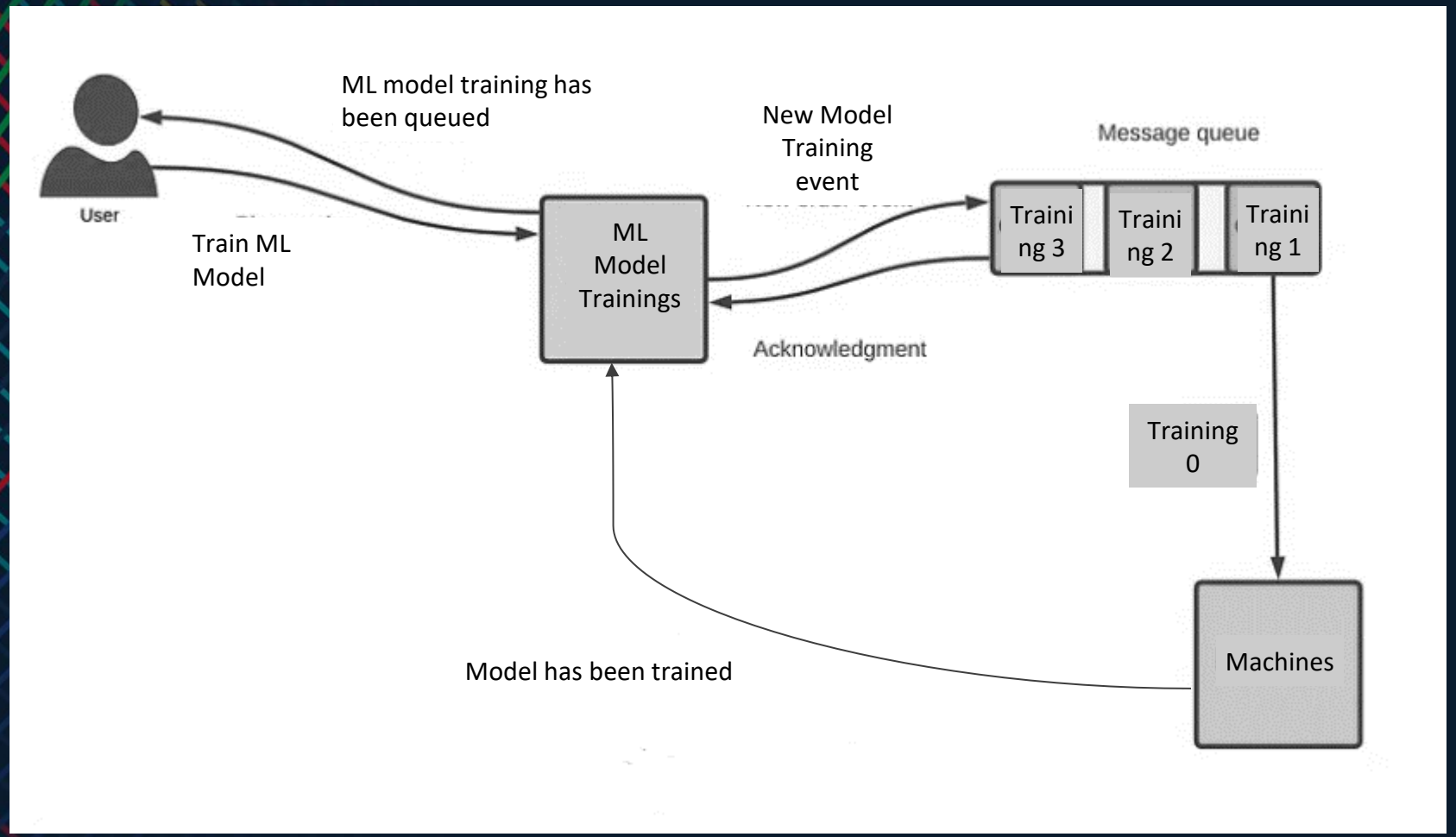
- Well, what happens if the number of user requests continue to increase beyond your Kubernetes cluster resource capability?



If the number of requests is beyond the available hardware resources, user requests will be dropped because the ML infrastructure can't handle all the incoming requests

In other words, your ML infrastructure won't have enough memory or processing power to conduct distribute training

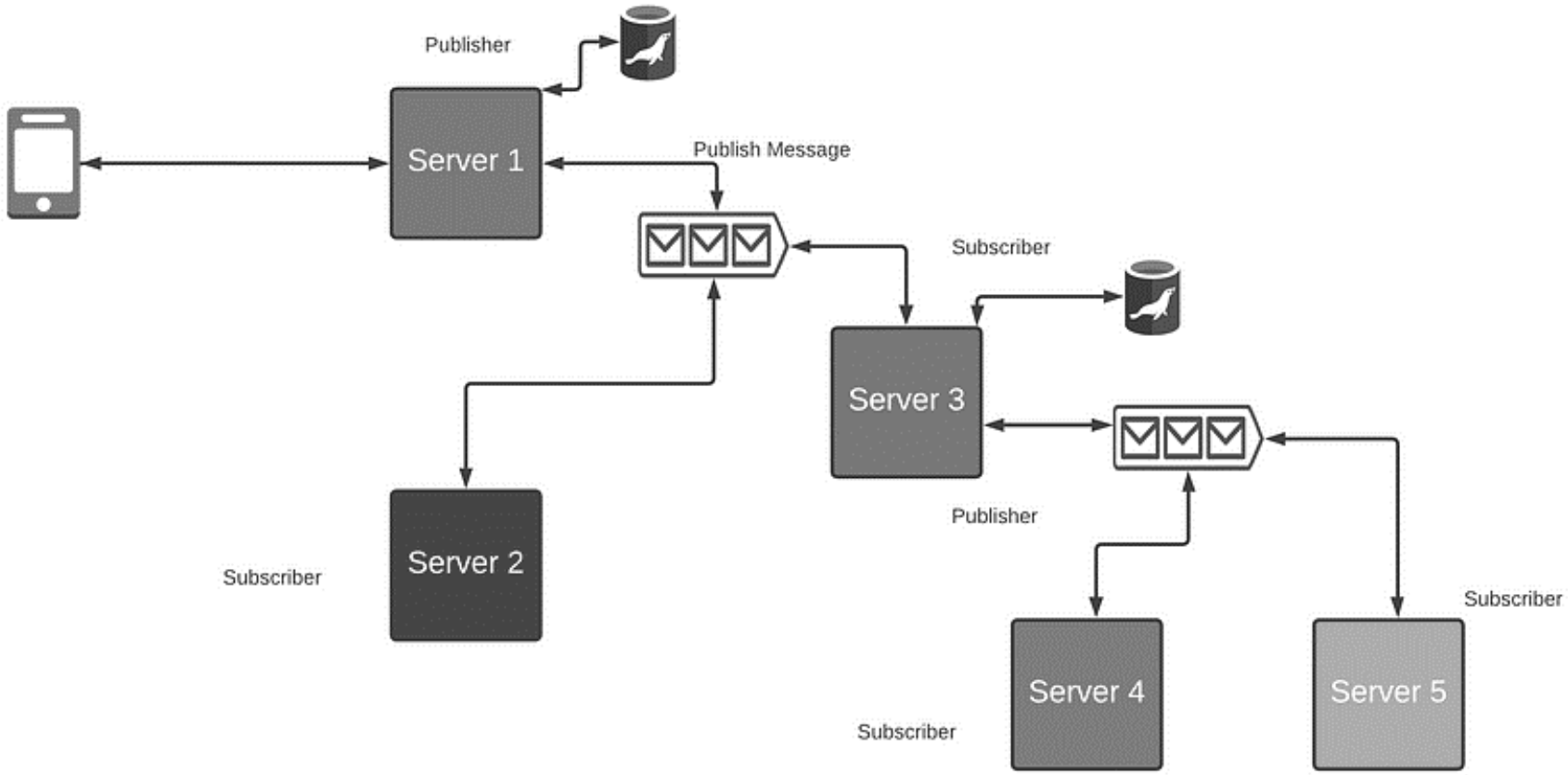
Solution: Use Message Queues



Apache Kafka

Apache Kafka is a scalable message queue with publish-subscribe software support that offers distributed streaming processing.

Digression 1: What is a Publish-Subscribe Model?



Kafka History

- Apache Kafka was originally developed by LinkedIn in 2010.
- Later, it was donated to the Apache Software Foundation and was made public in 2011.
- Currently, it is maintained by Confluent under Apache Software Foundation.

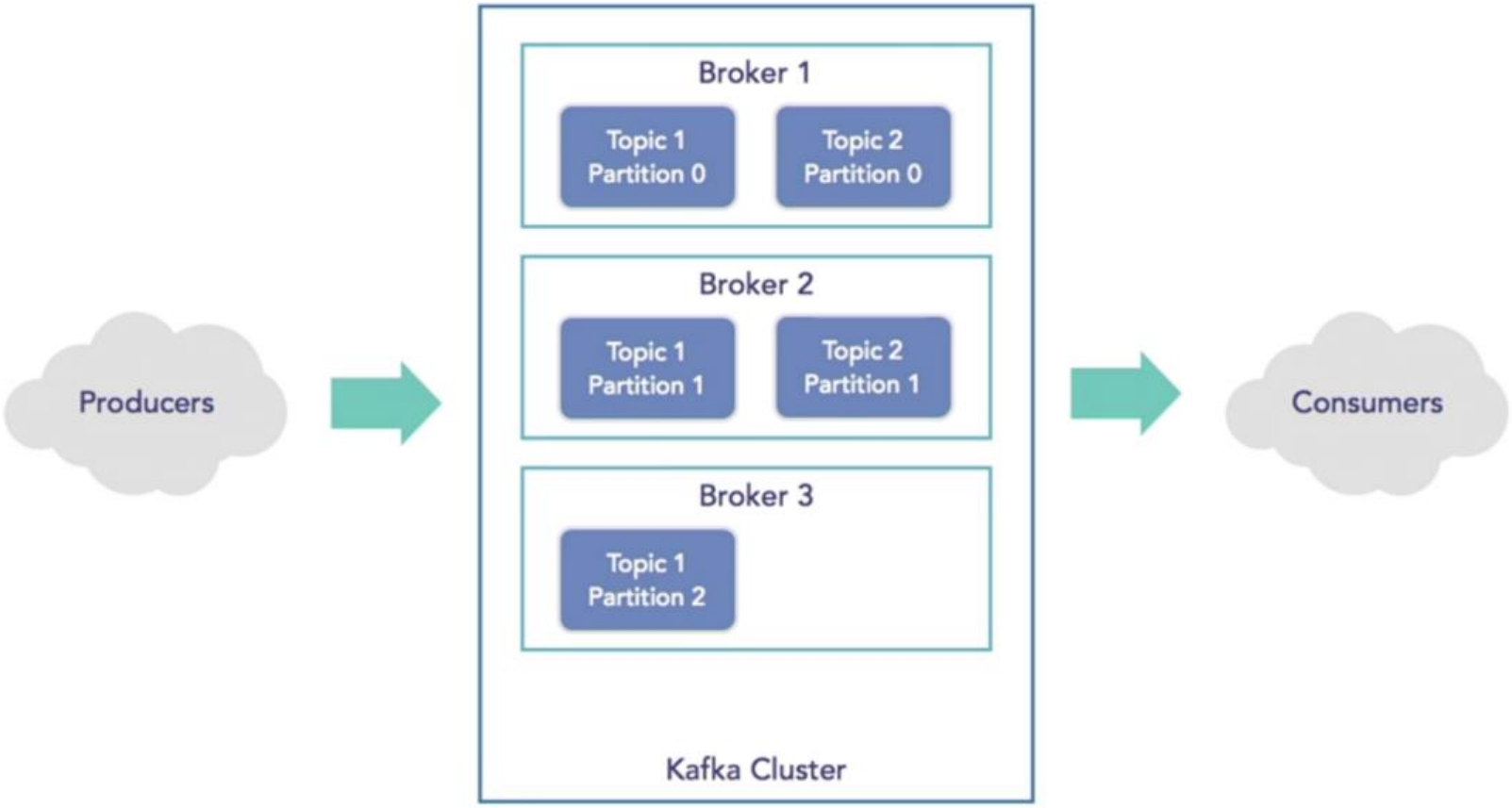
Kafka Advantages

- Very high performance: Kafka uses in-memory writes and reads
- High-throughput: LinkedIn leverages Kafka for up-to 3.2M messages/second
- Elastically scalable
- Low operational overhead
- Durable, highly available
- Fault-tolerant: Kafka is highly available and resilient to node failures and supports automatic recovery

Kafka Use Cases

- **LinkedIn** uses Kafka to prevent spam, collect user interactions to make better connection recommendations in real-time. Kafka is used to process 220B msg/day with a peak of **3.2M msg/second**.
- **Netflix** uses Kafka to apply recommendations in real-time while you are watching the TV shows.
- **X** uses Kafka as part of their Storm real-time data pipelines.
- **Spotify** leverages Kafka for log delivery.
- **Loggly** retains Kafka for log collection and processing
- **Other companies**: Airbnb, Cisco, Gnip, InfoChimps, Ooyala, Square, and Uber

Kafka Architecture



Topics

- Topic is a particular stream of data
- You may have as many topics as you want
- A topic is identified by its name
- A topic is divided into partitions

Partitions

- Each partition is ordered
- Each message within a partition gets an incremental id, called offset

Brokers

- A Kafka cluster is composed of multiple brokers (servers)
- Each broker is identified with its ID (integer)
- Each broker contains certain topic partitions
- After connecting to any broker, you will be connected to the entire cluster

Producers

- Producers write data to topics (which are made of partitions)
- Producers automatically know to which broker and partition they should write to
- In case of Broker failure, producers will automatically recover the failed broker.
- Producers choose whether to receive **acknowledgement** of data writes (ack)
- Producers choose whether to send a **key** with the message

Consumers

- Consumers read data from a topic (identified by a topic name)
- Consumers know which broker to read from.
- In case of Broker failures, consumers know how to recover the failed broker.
- Data are read in order within each partition.

Zookeeper

- Zookeeper manages brokers (keeps a list of them)
- Zookeeper helps in performing election for partitions
- Zookeeper sends notifications to Kafka in case of changes (e.g., new topic, broker failure, etc.).
- Zookeeper is an important component for Kafka.

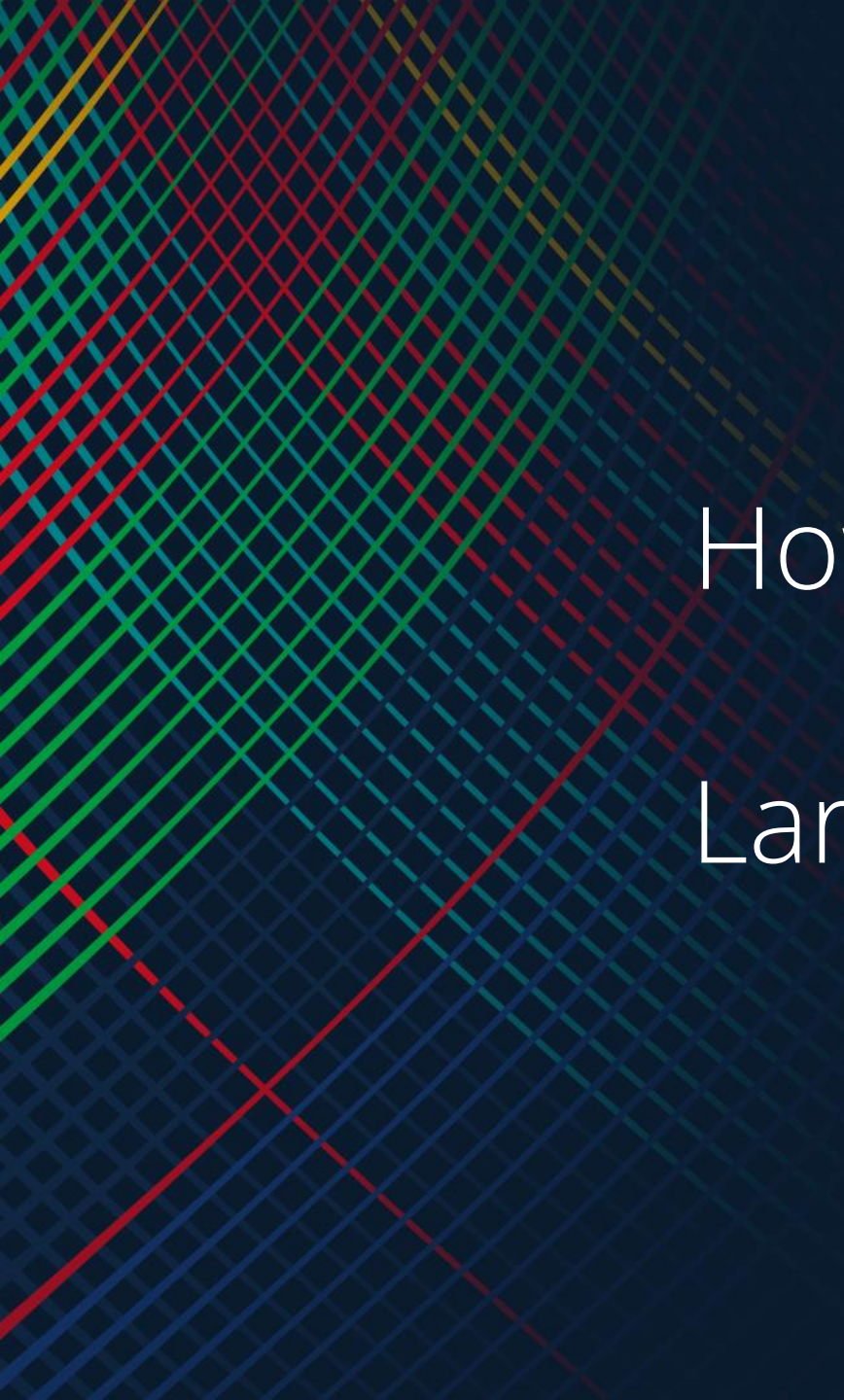
Topic Partitions

Topic Partitions

- A topic partition is the **unit of parallelism** in Apache Kafka.
- **Producers and Brokers:**
  - Writes to different partitions can be done in parallel.
  - Parallelism frees up hardware resources for operations like compression.
- **Consumers:**
  - Consumers can be grouped together into **several consumer groups**.
  - You can have up to one consumer instance per partition within a consumer group.
  - Any additional consumers beyond the number of partitions will remain idle.
- **Kafka Cluster**: More partitions in a Kafka cluster lead to higher system throughput.

More on Topic Partitions

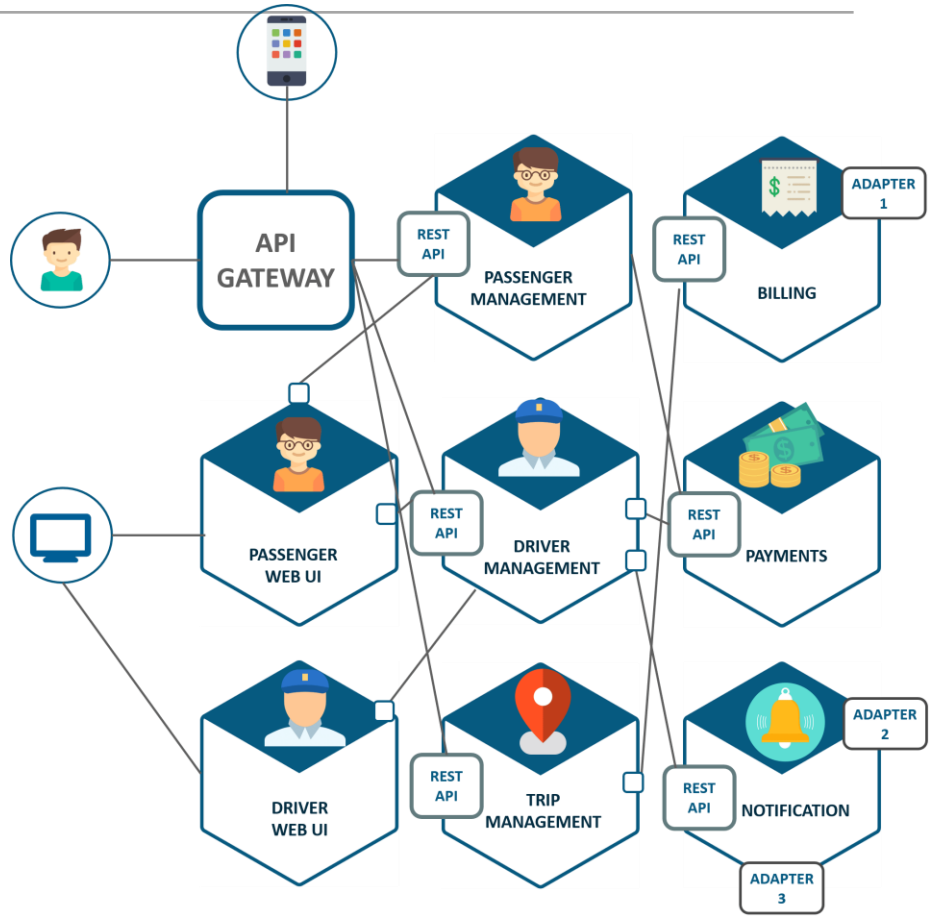
- **Scalable Partitioning**: You can expand the number of partitions as your data needs grow.
- **Keyed Messages Caution**: Exercise caution when messages are produced with keys, as they are deterministically assigned to partitions.
- **Consistent Routing**: Kafka ensures that messages with identical keys are consistently routed to the same partition, preserving the integrity of order-dependent applications.



But ..
How does Kafka fit in the development of Large-scale applications?

Digression 2: Microservices

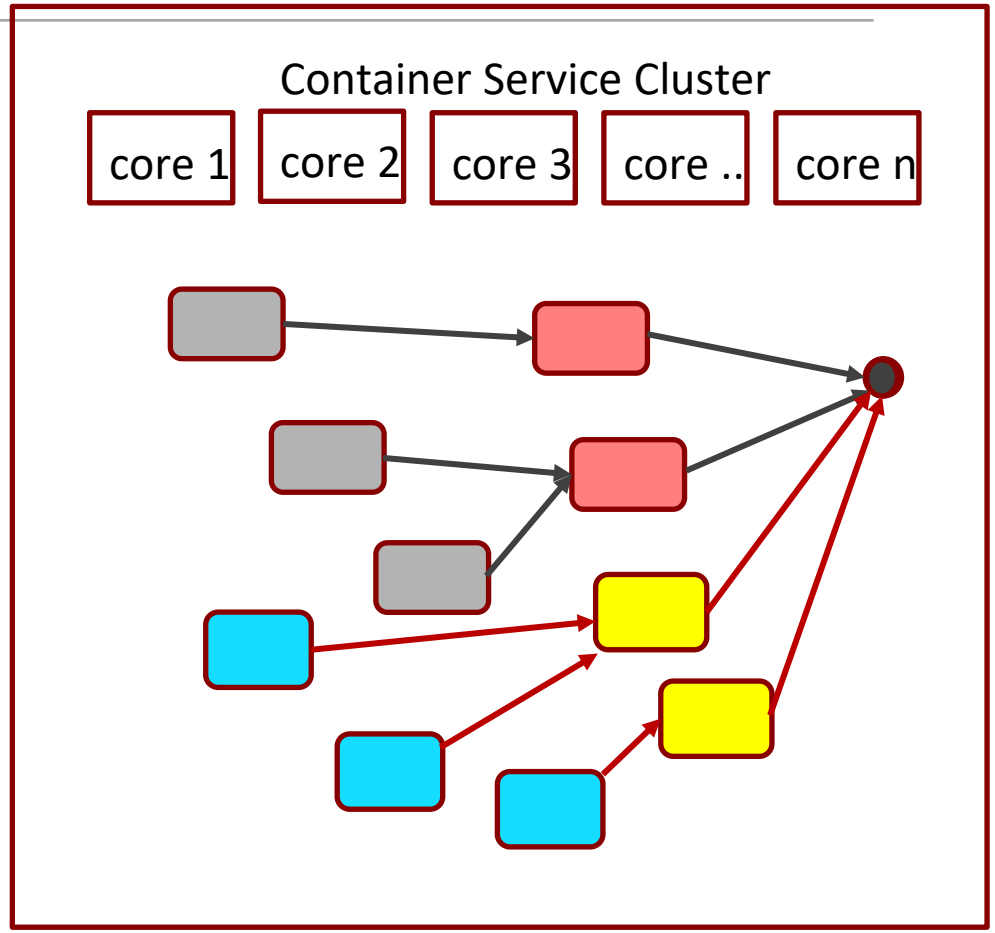
- Microservices are common software design pattern for applications running at scale.
- Divide a computation into small, mostly stateless components that can be:
  - Easily replicated for scale
  - Communicate with simple protocols
  - Computation is as a swarm of communicating workers.



Microservices in the Cloud

Typically run as containers using a service deployment and management service on systems like:

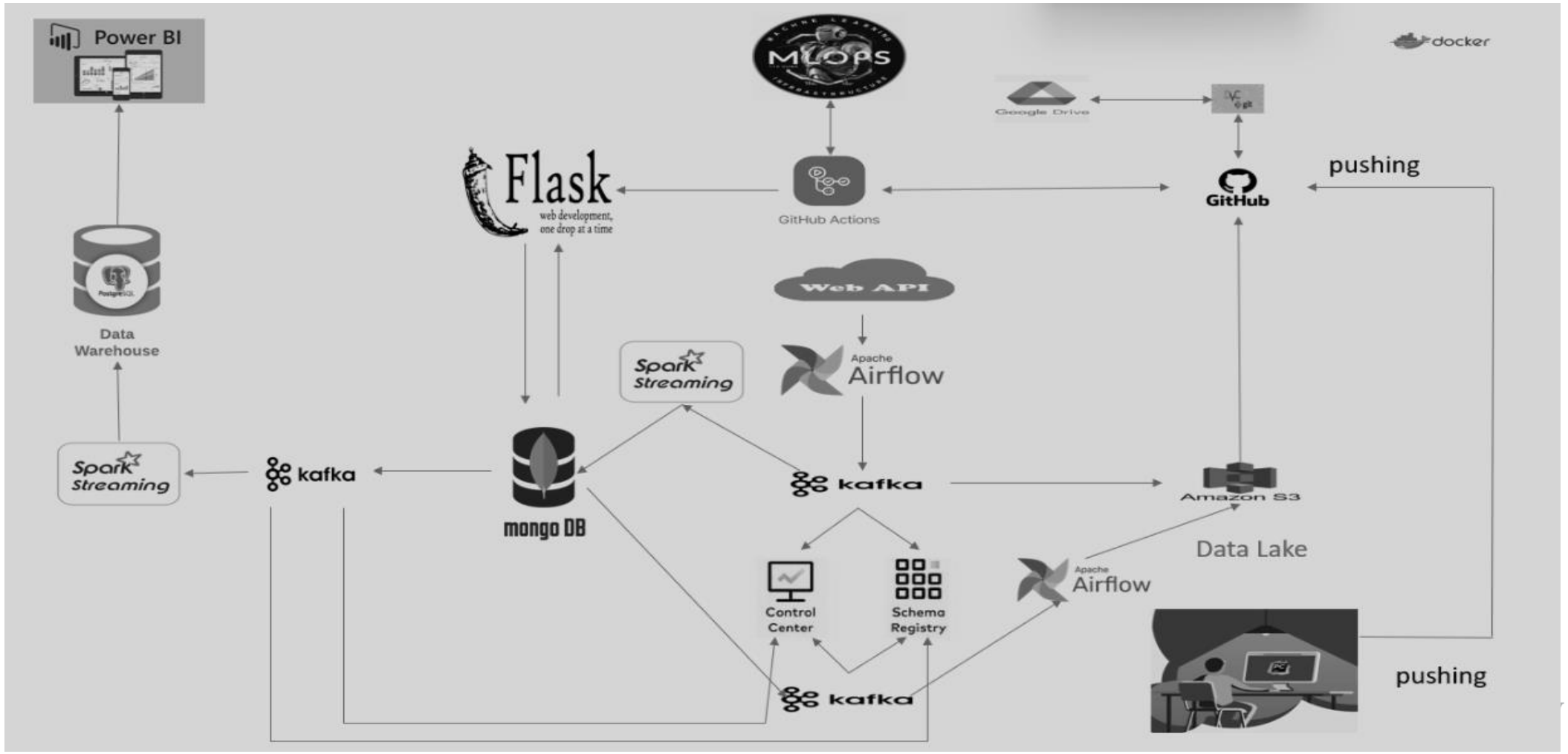
- Amazon Elastic Container Service
- Google Kubernetes
- DCOS from Berkeley/Mesosphere
- Docker Swarm



Digression 3 : REST APIs

- REST APIs provide an interface for distributed hypermedia systems, created by Roy Fielding in 2000
- The key abstraction of information in REST is a resource
  - Identified by a resource identifier, ie URI
- Resources can be retrieved or transformed to another state by a set of methods
  - GET/PUT/POST/DELETE/PATCH
- The clients and servers exchange representations of resources by using a standardized interface and protocol typically HTTP

Kafka and Microservices Combined Example



Apache Kafka
&
Confluent Kafka

Apache Kafka and Confluent Kafka

- Handling Kafka requires managing low-level, complex data infrastructure.
- Confluent Kafka is built on top of Apache Kafka to offer complete, fully managed, cloud-native data streaming that's available wherever your data and applications reside.

Apache Kafka vs Confluent Kafka

	Apache Kafka	Confluent
Serverless Automated, fully managed Kafka clusters with zero ops	–	✓
Elastic Scaling Scale up and down from 0 to GBps without over-provisioning infra	–	✓
Infinite Storage / Tiered Storage Cost-effectively retain data at any scale without growing compute	–	✓
High Availability Guaranteed 99.99% uptime SLA with built-in failover and multi-AZ replication	–	✓
No ZooKeeper management Metadata management completely abstracted away	–	✓
No-touch patching and upgrades Fully optimized infra with zero-downtime patching and upgrades	–	✓

Take-Home Activity (Complete before next lecture)

Create a “Self-managed” Confluent-Kafka on Your Machine

- Follow the following tutorials:
  - https://docs.confluent.io/platform/current/get-started/platform-quickstart.html

- Optional: Deploy Confluent Kafka to GKE:
  - https://docs.confluent.io/operator/current/co-quickstart.html