Lund&Bendsen
developing developers

Kafka overview

Exercise

Lund&Bendsen
developing developers

2.1

- Start kafka cluster
  - Make sure DockerDesktop is running
  - Open PowerShell and navigate to <project-folder>/docker
  - Run command **docker run hello-world** to see if docker I up and running
  - Type **docker-compose.exe up –d**
  - This will start a cluster with one zookeeper server and two Kafka brokers in detached mode.
  - Expect 5 min install time!
  - Make sure to keep this window running
  - All services are running in docker containers. The two brokers are exposed on port 29092 and 39092 on the host network
  - Open a new PowerShell window and navigate to <project-folder>/docker
  - Type **docker ps**
  - You should see something like this, showing all running containers:

```
PS C:\LB\LB2628-Kafka> docker ps
CONTAINER ID   IMAGE                              COMMAND                  CREATED       STATUS                   PORTS                                                  NAMES
db3ddc5a68e6   docker-connect-standalone          "start-kafka.sh"         3 hours ago   Up 53 minutes            0.0.0.0:8083->8083/tcp                                 connect-standalone
2b91310a8923   debezium/connect:1.6               "/docker-entrypoint.…"   3 hours ago   Up 53 minutes            8778/tcp, 9092/tcp, 9779/tcp, 0.0.0.0:9090->8083/tcp   postgres-debezium
57b3b8d9658c   tchiotludo/akhq                    "docker-entrypoint.s…"   3 hours ago   Up 3 hours (healthy)     0.0.0.0:8080->8080/tcp                                 akhq
b0beeb3926a1   confluentinc/cp-kafka:latest       "/etc/confluent/dock…"   3 hours ago   Up 53 minutes            9092/tcp, 0.0.0.0:39092->39092/tcp                     kafka2
01e549a6af17   confluentinc/cp-kafka:latest       "/etc/confluent/dock…"   3 hours ago   Up 53 minutes            9092/tcp, 0.0.0.0:29092->29092/tcp                     kafka1
22a21c8052df   confluentinc/cp-zookeeper:latest   "/etc/confluent/dock…"   3 hours ago   Up 3 hours               2888/tcp, 0.0.0.0:2181->2181/tcp, 3888/tcp             zookeeper
79a251ed09e2   postgres                           "docker-entrypoint.s…"   3 hours ago   Up 3 hours               0.0.0.0:5433->5432/tcp                                 postgres
PS C:\LB\LB2628-Kafka>
```

Kafka overview

Lund&Bendsen
developing developers

Kafka was originally built by LinkedIn to handle the growing amount of data streams in the company. Later it was open sourced. It's a distributed log optimized for high-throughput. Today it is used by many companies because of its flexibility and additional features build around it like **Kafka Streams**, a declarative DSL API to simplify stream-processing, **Kafka Connect**, an integration product that provides pluggable connectors to consume and produce to/from external data stores. The name Kafka was chosen by one of the founders of Kafka, Jay Kreps, because he liked the author Franz Kafka, and he thought the product was optimized for writing.

Producer: A producer is a component or application that publishes (produces) events or messages to Kafka topics.

Consumer: A consumer is a component or application that subscribes (consumes) events or messages from Kafka topics.

Broker: Kafka clusters consist of one or more Kafka brokers, which are responsible for storing and serving events. Each broker is a Kafka server.

ZooKeeper: Used for distributed coordination and management of Kafka brokers (control plane).

Kafka Streams: client-side DSL API to build event stream pipelines

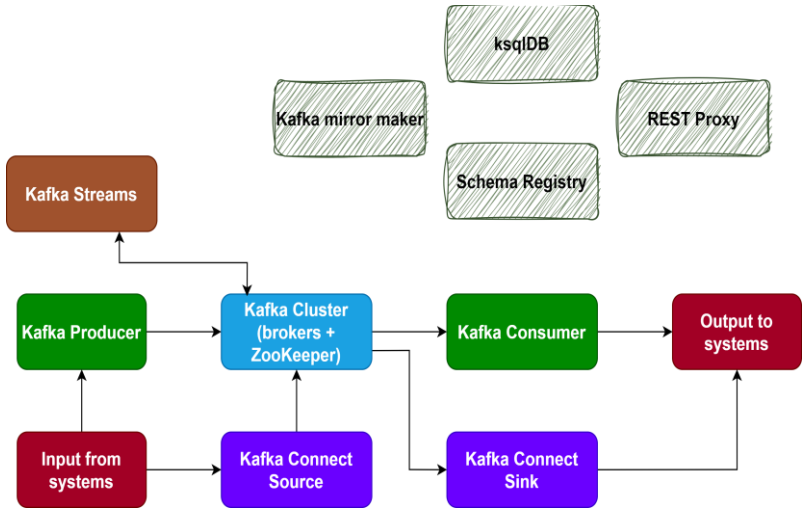
Kafka Connect: Framework to integrate data-sources & sinks with Kafka

Mirror maker: Replicate between two Kafka clusters

Schema Registry: Enforce message schema compliance for producers & consumers

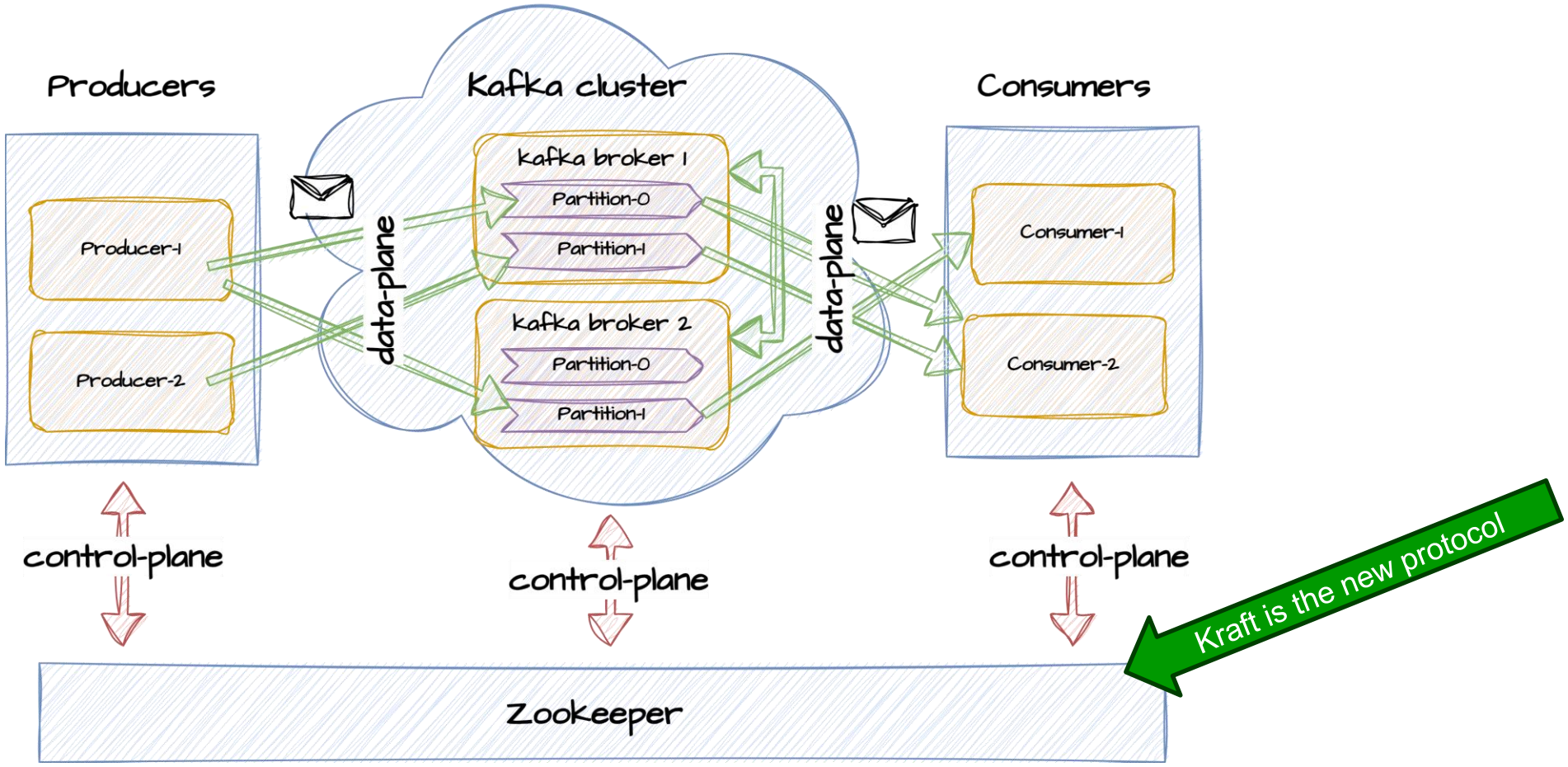
REST Proxy: Proxy for non-java clients

ksqlDB: SQL like stream processing for data analysis



Kafka overview

Lund&Bendsen
developing developers



Kafka use cases

Lund&Bendsen
developing developers

Messaging System: Kafka serves as a distributed messaging system, providing reliable and scalable message queuing. It enables communication between different components of a distributed system.

Microservices Communication: Kafka can facilitate communication between microservices. It allows for decoupling services and ensures reliable message delivery between them.

Event Sourcing: Event sourcing architectures use Kafka to store and manage events that represent changes to a system's state over time. This enables building systems that can be easily reconstructed to any point in time.

Log Aggregation: Kafka is often used to collect and aggregate log data from different services and applications

Real-time Analytics: Kafka's ability to handle high-throughput, low-latency data streams makes it suitable for real-time analytics applications

Metrics and Monitoring: Kafka can be used to collect and process metrics and monitoring data from various sources.

IoT (Internet of Things): Kafka is employed in IoT scenarios for handling large volumes of streaming data generated by sensors and devices. It provides a robust infrastructure for processing and analyzing real-time data from IoT deployments.

Replication and Backup: Kafka's replication features make it suitable for creating backups and ensuring data durability. It ensures that data is replicated across multiple brokers, reducing the risk of data loss.

How kafka works

Lund&Bendsen
developing developers

Publishing Events (Producer):

Producers send events to Kafka topics. Each event is associated with a specific topic.

Producers typically batch events for efficiency and send them to Kafka brokers over a network connection.

Producers can choose to send events to a specific partition within a topic or allow Kafka to choose a partition using a partitioning strategy (e.g., round-robin, key-based).

Storing Events (Broker):

Kafka brokers receive and store events in their distributed log data store. Each event is appended to the appropriate partition.

Events are assigned sequential offsets within their partitions. Offsets serve as unique identifiers for events within a partition.

Replication:

Kafka provides data replication for fault tolerance. Each partition has multiple replicas distributed across different brokers.

Replication ensures that events are not lost if a broker fails. One replica is designated as the leader, and others are followers.

Producers send events to the leader replica, and followers replicate the data from the leader.

How kafka works

Lund&Bendsen
developing developers

Consuming Events (Consumer):

Consumers subscribe to Kafka topics and specify the partitions they want to consume from.

Kafka allows multiple consumers to subscribe to the same topic and partition, enabling parallel processing.

Consumers read events from their assigned partitions sequentially based on their offsets.

Retention and Cleanup:

Kafka retains events for a configurable period (retention period) or until a certain size threshold is reached.

Events that have been consumed are not immediately deleted but are marked for deletion based on their offsets.

Kafka uses a garbage collection process to reclaim disk space by deleting expired events.

Scaling:

Kafka is designed to be horizontally scalable. You can add more brokers to a Kafka cluster to increase capacity.

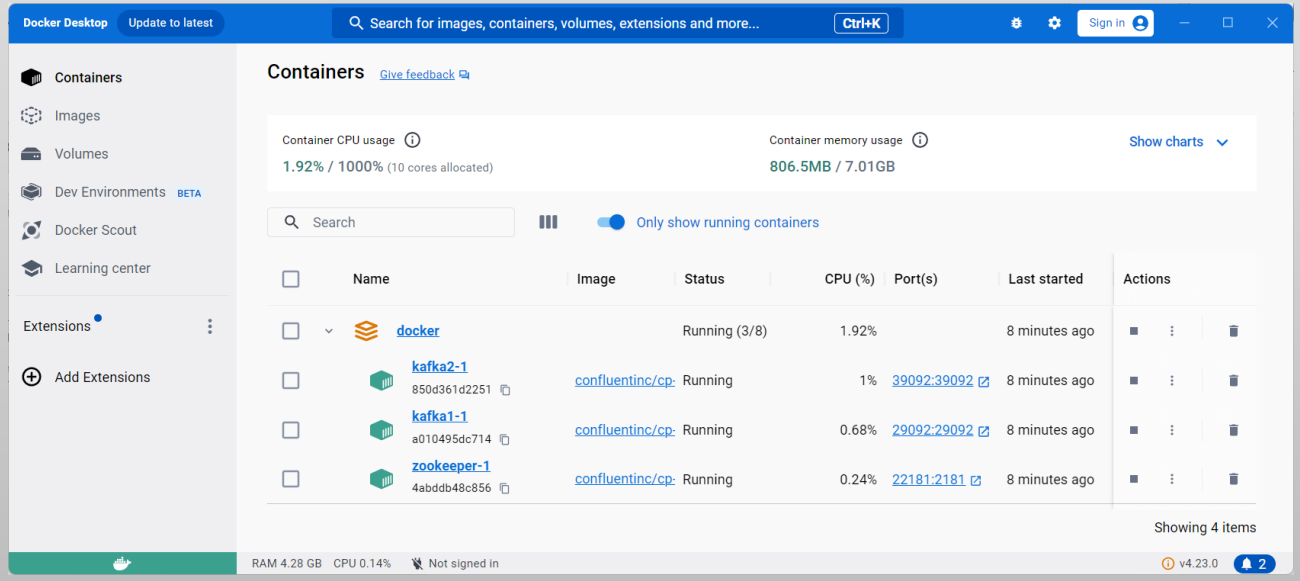
Partitions can be added to topics to distribute the workload and accommodate higher throughput.

Exercise

Lund&Bendsen
developing developers

2.2

- Open Docker Desktop
  - In the taskbar type ‘docker desktop’ and open the application
  - You should see something like the below window showing all running containers.
  - Docker desktop provides a user-friendly interface to see running containers.
  - Here you can start, stop and delete containers
  - Walk-through of the deployment and the containers



Kafka relational diagram

Lund&Bendsen
developing developers