Apache Kafka

Apache Kafka is an open-source distributed event streaming platform developed by the Apache Software Foundation. It is designed for building real-time data pipelines and streaming applications. Kafka is capable of handling high throughput and low-latency data transmission, making it a popular choice for organizations needing to process large volumes of data quickly.

Kafka's ability to handle large-scale message streaming with high fault tolerance and scalability makes it a cornerstone technology for many modern data architectures. It is widely used in industries such as finance, retail, healthcare, and technology for applications ranging from log aggregation and monitoring to real-time analytics and event-driven microservices.

Why Use Event Drive Architecture (EDA)

Loose Coupling

All services/components of your application were used to run on a single Server. In case of server failure, complete application was out of service (due to Tight Coupling i.e. all services on a single Server). Tight coupling is easy from developer point of view.

Loose Coupling is a concept of having Microservices that are combined to create a whole application. In this, if one service is down, other services are not effected and that service is restarted by creating a new Container and deploying it.

Scalability

If one service becomes loaded, we can Horizontally Scale this service only by adding containers of this service in the system.

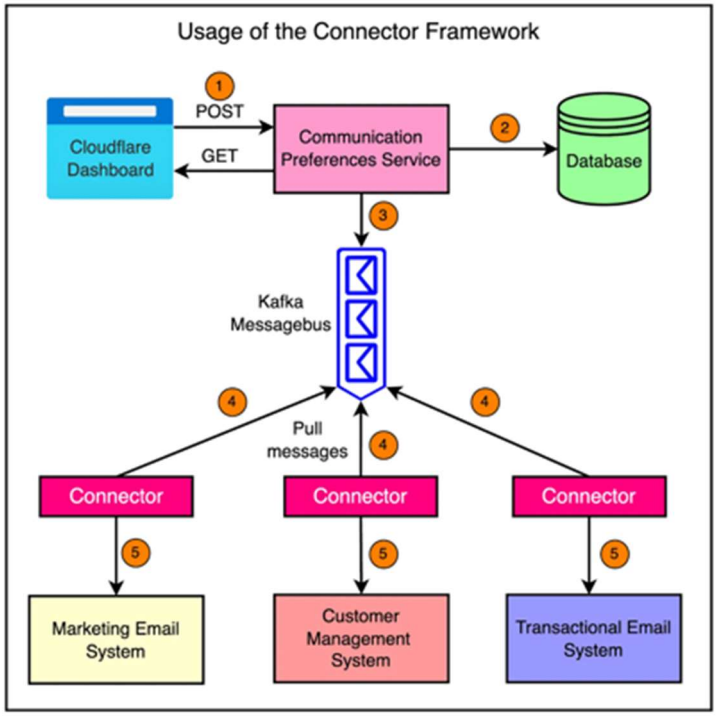
Real Time Processing

Previously an empty request was sent after some time (few seconds) to backend to retrieve if there is any new data in backend database. This will increase load to Server.

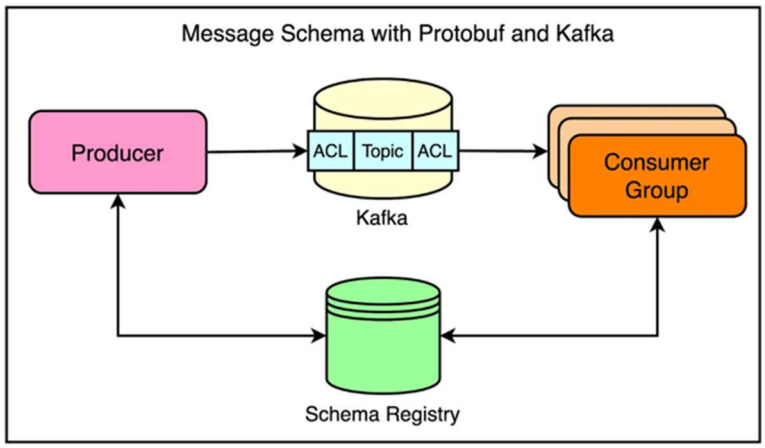
With EDA, if a new message is receive in backend, the server send the Payload to user. (Without being specifically asked.) i.e. Events can trigger actions as they happen.

Reducing Load on Database and Server

With Kafka, a message is passed to all its subscribers without involving Server and Database



Kafka Terminologies



1. **Producers**: These are clients that publish data (messages) to Kafka topics. Producers can send data to one or more topics, and they have the ability to choose which partition within a topic the data is sent to, allowing for load balancing and data organization.

2. **Consumers**: These clients read data from Kafka topics. Consumers can be part of a consumer group, which allows multiple consumers to read from the same topic concurrently, dividing the data load among them.

3. **Access Control Lists (ACLs):** Access Control Lists (ACLs) are a security feature in Apache Kafka used to control who can access and perform operations on Kafka resources such as topics, consumer groups, and the Kafka cluster itself. ACLs help enforce security by specifying which users or applications (identified by their principals) have permission to perform certain actions.

4. **Topics**: A topic is a category or feed name to which records are sent. Topics in Kafka are partitioned and replicated across multiple nodes, providing scalability and fault tolerance.

5. **Schema Registry:** The Schema Registry is a service for managing and enforcing schemas for data stored in Kafka topics. It provides a centralized repository for schemas and ensures that data producers and consumers adhere to a common schema format. The Schema Registry supports schema evolution, allowing schemas to evolve over time while maintaining compatibility.

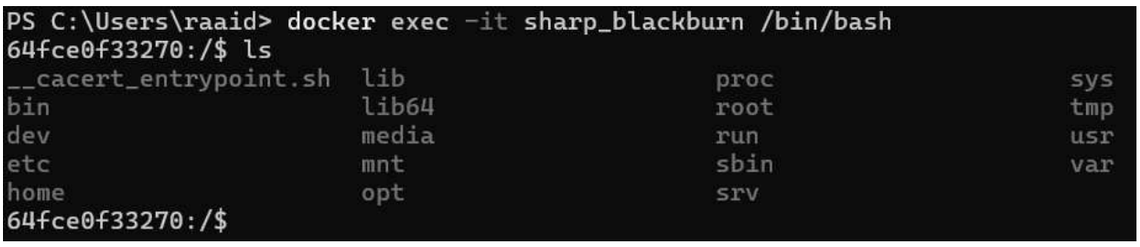
6. **Protobuf**: Protocol Buffers, commonly known as Protobuf, is a language-agnostic binary serialization format developed by Google. Protobuf is used to serialize structured data, making it efficient for both storage and communication. It is particularly popular for use in remote procedure calls (RPCs) and for data interchange between systems.

```
syntax = "proto3";

message Person {
    string name = 1;
    int32 id = 2;
    string email = 3;
}
```

BLOG: https://blog.bytebytego.com/p/cloudflares-trillion-message-kafka

Using Kafka

- Install Docker Desktop and Run It
- Pull `Apache Kafka` Docker Image using below command
  - `docker pull apache/kafka:3.7.0`
- Start the Kafka docker container
  - `docker run -p 9092:9092 apache/kafka:3.7.0`
- Switch Terminal to Docker Container inside specific Directory
  - `docker exec -it <container-name> /bin/bash`



- Switch to directory `opt/kafka/bin` which contains Kafka Commands

- Run following command in bin directory to **Create** a *Kafka Topic*
  - `/opt/kafka/bin/kafka-topics.sh --create --topic <topic name> --bootstrap-server localhost:9092`
  - topic name can be anything

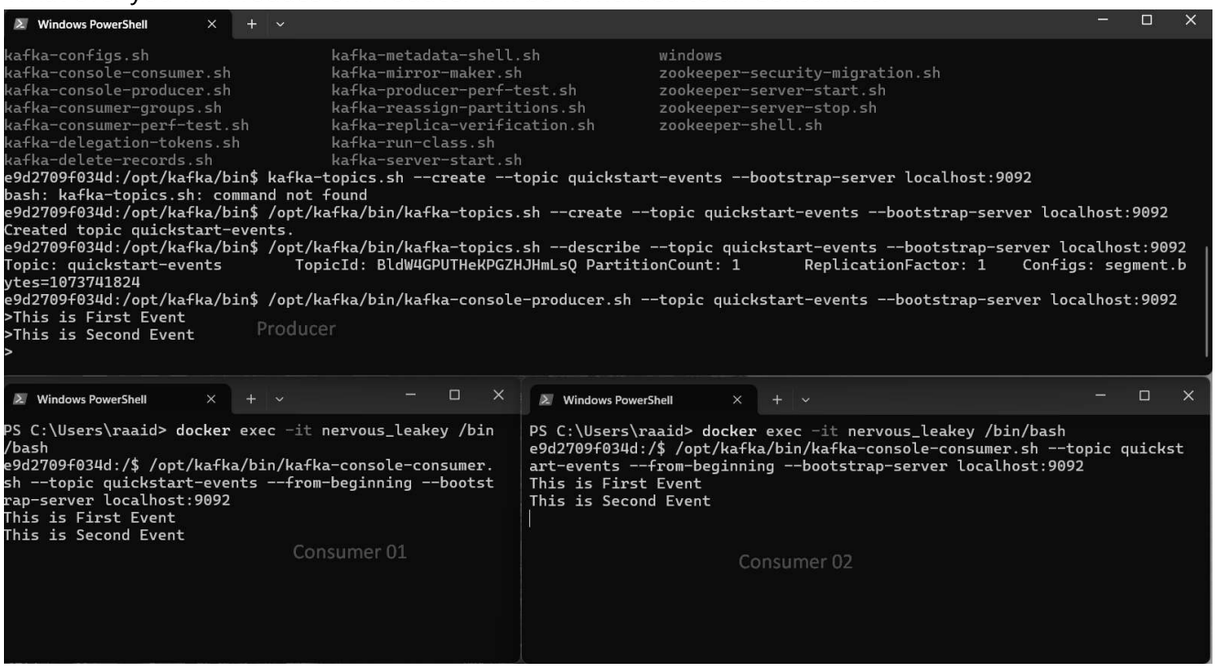
- -bootstrap-server is a server used to run kafka inside internal container. (In our case Docker Container)

```
e9d2709f034d:/opt/kafka/bin$ /opt/kafka/bin/kafka-topics.sh --create --topic quickstart-events --bootstrap-server localhost:9092
Created topic quickstart-events.
e9d2709f034d:/opt/kafka/bin$
```

- Run following command to **Describe** a *Kafka Topic*
  - `/opt/kafka/bin/kafka-topics.sh --describe --topic <topic name> --bootstrap-server localhost:9092`

```
e9d2709f034d:/opt/kafka/bin$ /opt/kafka/bin/kafka-topics.sh --describe --topic quickstart-events --bootstrap-server localhost:9092
Topic: quickstart-events        TopicId: BldW4GPUTHeKPGZHJHmLsQ PartitionCount: 1       ReplicationFactor: 1    Configs: segment.byt
es=1073741824
        Topic: quickstart-events        Partition: 0    Leader: 1       Replicas: 1     Isr: 1
e9d2709f034d:/opt/kafka/bin$
```

- Run following command to **Create** a *Producer*
  - `/opt/kafka/bin/kafka-console-producer.sh --topic quickstart-events --bootstrap-server localhost:9092`
- Run following command to **Create** a *Consumer*
  - `/opt/kafka/bin/kafka-console-consumer.sh --topic quickstart-events --from-beginning --bootstrap-server localhost:9092`
  - You can create multiple consumers. In our case 02
- What ever you will Produce on Producer will be forwarded to all Consumers.



Kafka UI

- Create a Docker Network with name *kafka-net*
  - `docker network create -d bridge kafka-net`

```
PS C:\Users\raaid> docker network create -d bridge kafka-net
d0e766120bedbe15e8f4980816734bc4dc196a1c2dbf2218a1a6641900890e81
```

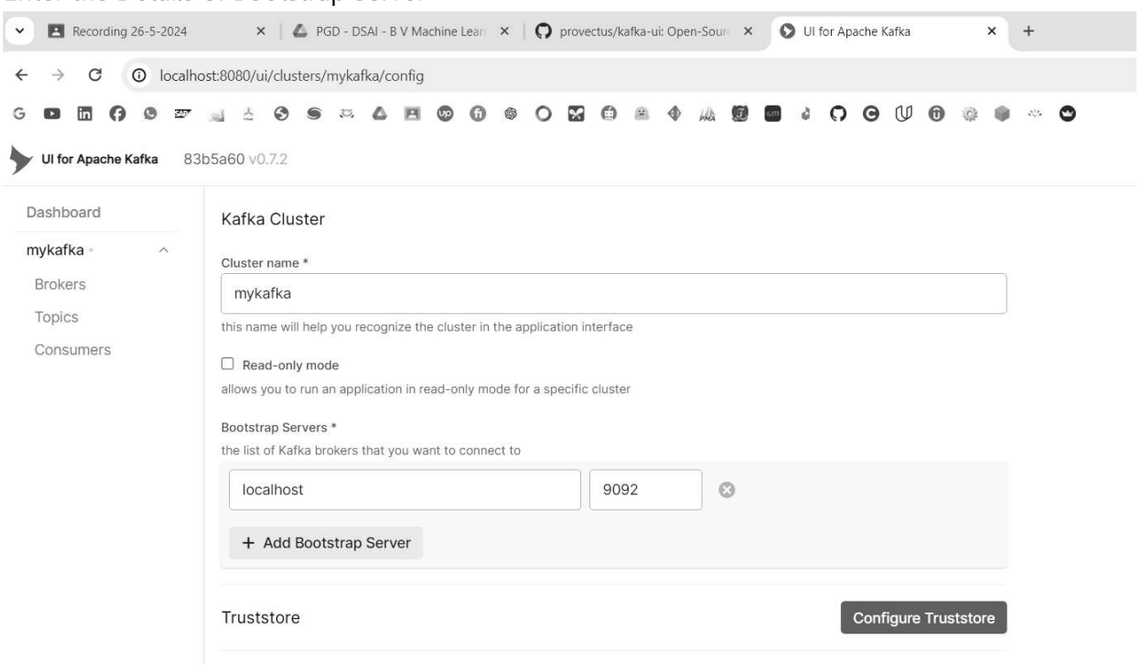
- Show all Docker Networks
    - `docker network ls`

```
PS C:\Users\raaid> docker network ls
NETWORK ID     NAME                      DRIVER    SCOPE
e9112f87877b   bridge                    bridge    local
2f301d438364   host                      host      local
d0e766120bed   kafka-net                 bridge    local
0718242c9597   none                      null      local
f93879644674   todo-app_my-api-net-todo  bridge    local
```

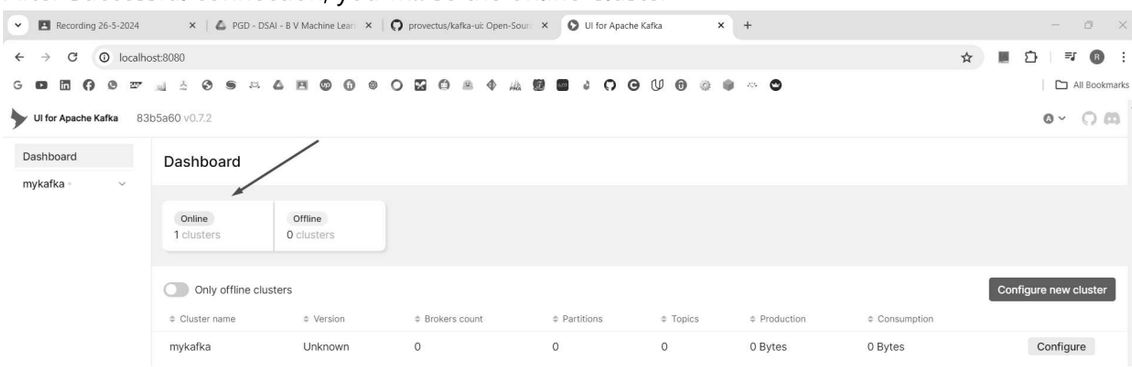
- Stop Containers already running in Docker (Specially if Container is using port 9092)
- Create a New Container named *`mykafka`* which is using Network *`kafka-net`*
    - `docker run -p 9092:9092 --network kafka-net --name mykafka apache/kafka:3.7.0`

```
PS C:\Users\raaid> docker run -p 9092:9092 --network kafka-net --name mykafka apache/kafka:3.7.0
===> User
uid=1000(appuser) gid=1000(appuser) groups=1000(appuser)
===> Setting default values of environment variables if not already set.
CLUSTER_ID not set. Setting it to default value: "5L6g3nShT-eMCtK--X86sw"
===> Configuring ...
===> Launching ...
===> Using provided cluster id 5L6g3nShT-eMCtK--X86sw ...
[2024-05-29 10:21:58,466] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistratio
n$)
[2024-05-29 10:21:58,521] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TL
S renegotiation (org.apache.zookeeper.common.X509Util)
[2024-05-29 10:21:58,604] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSigna
lHandler)
[2024-05-29 10:21:58,606] INFO [ControllerServer id=1] Starting controller (kafka.server.ControllerServer)
```

- Open a new terminal and run following command.
    - `docker run -it -p 8080:8080 --network kafka-net -e DYNAMIC_CONFIG_ENABLED=true provectuslabs/kafka-ui`
    - This will pull ***kafka-ui*** from Docker hub.
    - Go to `localhost:8080` to access *kafka-ui*
    - Enter the Details of Bootstrap Server



- After Successful connection, you will se the online Cluster



- ***This method for learning purpose only, Generally Docker-Compose is used to build Kafka Servers (defined below)***

Kafka-UI (With Docker Compose)

- Create following `docker-compose.yml` file

```
version: '3.8'

x-kafka-common: &kafka-common
  image: 'bitnami/kafka:latest'
  ports:
    - "9092"
  networks:
    - kafka-net
  healthcheck:
    test: "bash -c 'printf \"\" > /dev/tcp/127.0.0.1/9092; exit $$?;'"
    interval: 5s
    timeout: 10s
    retries: 3
    start_period: 30s
  restart: unless-stopped

x-kafka-env-common: &kafka-env-common
  ALLOW_PLAINTEXT_LISTENER: 'yes'
  KAFKA_CFG_AUTO_CREATE_TOPICS_ENABLE: 'true'
  KAFKA_CFG_CONTROLLER_QUORUM_VOTERS: 0@kafka-0:9093,1@kafka-1:9093
  KAFKA_KRAFT_CLUSTER_ID: abcdefghijklmnopqrstuv
  KAFKA_CFG_PROCESS_ROLES: controller,broker
  KAFKA_CFG_CONTROLLER_LISTENER_NAMES: CONTROLLER
  KAFKA_CFG_LISTENERS: PLAINTEXT://:9092,CONTROLLER://:9093
  EXTRA_ARGS: "-Xms128m -Xmx256m"

services:

  kafka-0:
    <<: *kafka-common
```

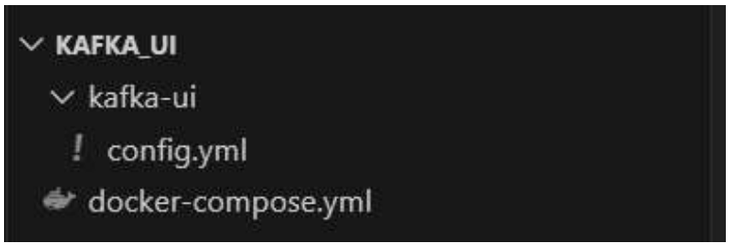
```
    environment:
      <<: *kafka-env-common
      KAFKA_CFG_NODE_ID: 0
    volumes:
      - kafka_0_data:/bitnami/kafka

  kafka-1:
    <<: *kafka-common
    environment:
      <<: *kafka-env-common
      KAFKA_CFG_NODE_ID: 1
    volumes:
      - kafka_1_data:/bitnami/kafka

  kafka_ui:
    container_name: kafka-ui
    image: provectuslabs/kafka-ui:latest
    volumes:
      - ./kafka-ui/config.yml:/etc/kafkaui/dynamic_config.yaml
    environment:
      DYNAMIC_CONFIG_ENABLED: 'true'
    depends_on:
      - kafka-0
      - kafka-1
    networks:
      - kafka-net
    ports:
      - '8080:8080'
    healthcheck:
      test: wget --no-verbose --tries=1 --spider localhost:8080 || exit 1
      interval: 5s
      timeout: 10s
      retries: 3
      start_period: 30s

networks:
  kafka-net:

volumes:
  kafka_0_data:
    driver: local
  kafka_1_data:
    driver: local
```

- Create a folder name `kafka-ui` in same Directory and create a `config.yml` file.

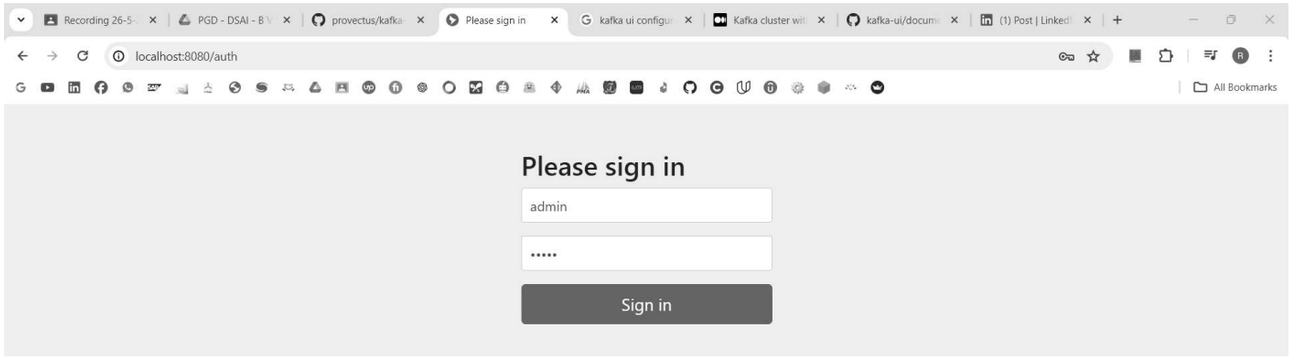
```
auth:
  type: LOGIN_FORM

spring:
  security:
    user:
```

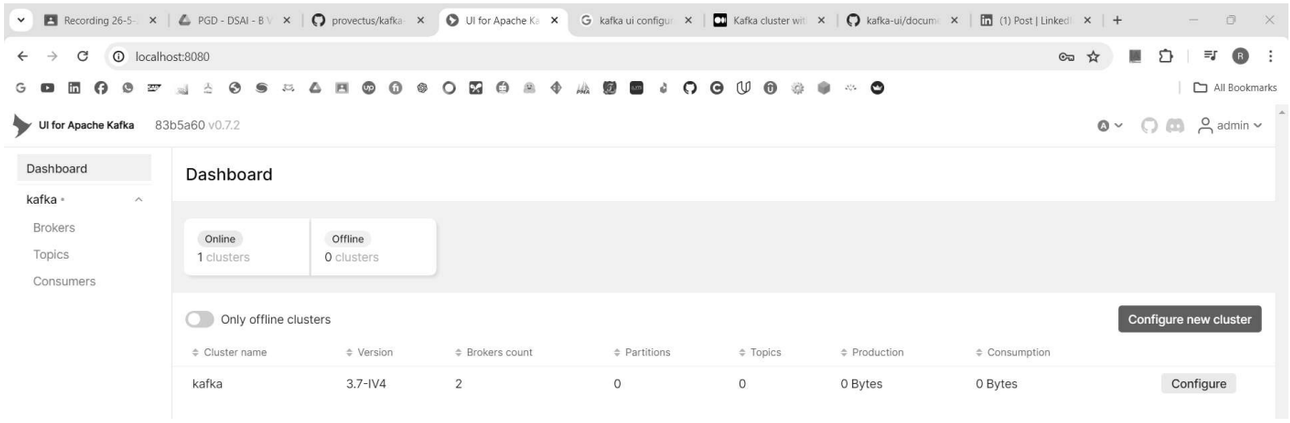
```
      name: admin
      password: admin

kafka:
  clusters:
    - bootstrapServers: kafka-0:9092,kafka-1:9092
      name: kafka
```

- Below if the file structure



- Run all Containers using `docker compose up` command.
- Go to `http://localhost:8080/` and use credentials from `config.yml` file (i.e. admin admin)





Using Kafka in Applications

- You can use Python Library `AIOKafka` to create Producers and Consumers from within Application Code.