Streamline Your Workflows: A Step-by-Step Guide to Kafka-App ConnectIntegration

From Kafka to App Connect: A Practical Guide

-Votarikari Shravan (912)

Table of Contents

Introduction

In today's data-driven world, applications need to exchange information seamlessly and efficiently. This is where Apache Kafka and IBM App Connect come into play.
Apache Kafka is a high-throughput, distributed streaming platform. It excels at handling real-time data streams, making it ideal for applications that require continuous data ingestion, processing, and delivery. Kafka acts as a central nervous system, constantly flowing data between applications.
IBM App Connect Enterprise is an integration platform that simplifies the process of connecting diverse applications, data sources, and APIs. It acts as a bridge, allowing applications to communicate and share data regardless of their underlying technologies.

Event and Event Streaming?

Event

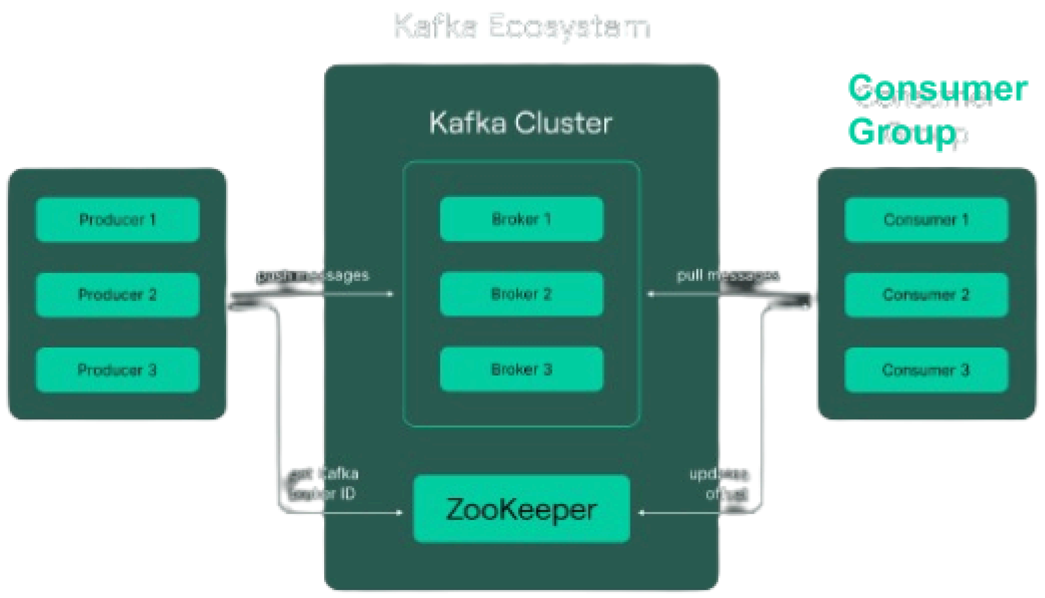
Events in a software system or its environment are essentially records of occurrences. They can range from a user logging in to a product being bought, a sensor detecting a temperature change, or a system error. These events usually include details about the incident, the timing, and other pertinent information like user ID, product specifics, or error codes.

Event Streaming

Event streaming involves the continuous transmission of real-time event data. This process includes capturing events as they occur and sending them to a central platform known as an event stream broker. The broker then stores and processes the event stream, allowing applications to utilize the data in real-time or for analysis at a later time.

Architecture of basic event stream processing

Apache Kafka is a distributed streaming platform that is designed to handle high volumes of real-time data efficiently. Its architecture is built around a few key concepts and components that work together to enable the ingestion, storage, processing, and consumption of data streams.



Below is an overview of the key components and their interactions in Apache Kafka architecture:

Topics:

- Data is organized and stored in Kafka topics.
- Topics can be thought of as a category or feed name to which records are published by producers and from which records are consumed by consumers.
- Topics are partitioned for scalability and parallelism.

Partitions:

- Each topic can be split into multiple partitions.
- Partitions allow data to be distributed across multiple brokers and can be thought of as the basic unit of parallelism in Kafka.
- Each partition is an ordered, immutable sequence of records that is continuously appended to.

Brokers:

- Kafka brokers are servers that store data and serve client requests.
- Each broker can handle a specific number of partitions and acts as a leader or follower for them.
- Brokers are responsible for storing data, handling partition leaders, and replicating data across the cluster.

Producers:

- Producers are applications that publish data records to Kafka topics.
- Producers are responsible for choosing which topic and partition to publish to and are typically designed to be highly available and fault-tolerant.

Consumers:

- Consumers are applications that subscribe to topics and process the data records published to them.
- Consumers can read data from one or more partitions of a topic and can be part of a consumer group to scale processing and balance the load.
- Consumers keep track of their offset (position) in each partition, allowing them to read data sequentially and independently.

Consumer Groups:

- Consumer groups are a way to parallelize consumption of data by allowing multiple consumer instances to divide the work and share the partitions of a topic.
- Each consumer group has one or more consumer instances, and each partition is consumed by only one consumer instance within a group.
- Consumer groups enable horizontal scalability, fault tolerance, and load balancing.

ZooKeeper (Deprecated in newer versions, but still relevant in older versions):

- ZooKeeper is used for managing and coordinating the Apache Kafka cluster.
- It helps in electing a leader for each partition, maintaining partition metadata, and managing broker failures and reassignments.
- With newer versions of Kafka, ZooKeeper is being replaced by Kafka's internal metadata quorum for improved simplicity and performance.

Kafka Connect:

- Kafka Connect is a framework for connecting Kafka with external systems such as databases, file systems, and other data sources and sinks.
- It simplifies the integration and data import/export between Kafka and other systems by providing pre-built connectors and a standardized API.

Kafka Streams:

- Kafka Streams is a client library for building real-time stream processing applications on top of Kafka.
- It allows developers to process and transform data streams from Kafka topics and produce output streams to other topics or external systems.

Pre-Requisites

- Apache Kafka Latest Release
  - Any one of binary download form https://kafka.apache.org/downloads
- App Connect Enterprise

Preparation

Following preparations need to be done to specify locations for recording logs and to store snapshots.

Note : we are assuming your operating system is windows. To run the Kafka server, all required commands to be executed are available under the bin/windows folder.

Zookeeper.properties File

- Specify a file path for dataDir property.

Server.properties File

- Specify a file path for log.dirs property. Under this location logs for kafka will be stored.

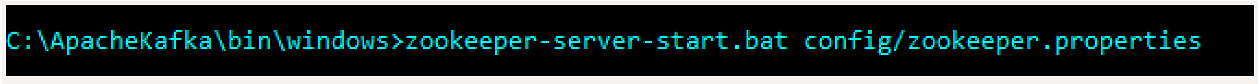
Run Apache Kafka

We assume a necessary configuration done as follows: a **preparation** in the above section. Now it is time to run zookeeper and kafka servers and enable the event-driven system by following procedure.

Note : It is an advisory to set environment variable for bin/windows folder to enable

- Apache Kafka can be started using ZooKeeper

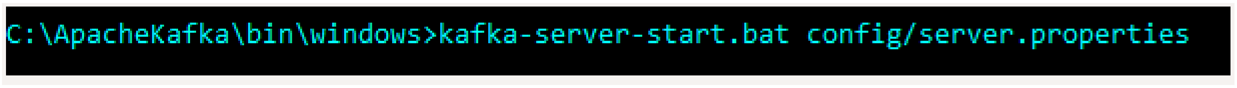
Start the ZooKeeper service by supplying the zookeeper.properties file.



```
C:\ApacheKafka\bin\windows>zookeeper-server-start.bat config/zookeeper.properties
```

- Open another terminal, run kafka broker service

Start the kafka broker service by supplying server.properties file

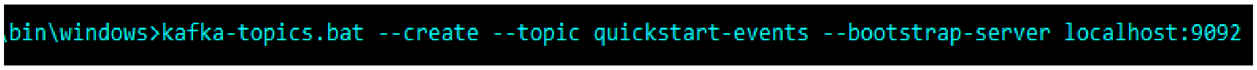


```
C:\ApacheKafka\bin\windows>kafka-server-start.bat config/server.properties
```

Test Apache Kafka

Create a Topic

To test it we need a topic, it is created by following command.

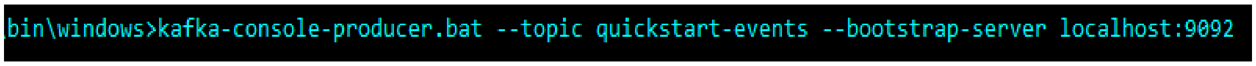


```
bin\windows>kafka-topics.bat --create --topic quickstart-events --bootstrap-server localhost:9092
```

Here the topic name is quickstart-events.

Write Events into topic

To write events into a topic we need a Producer. The following command will let you to write events into a topic

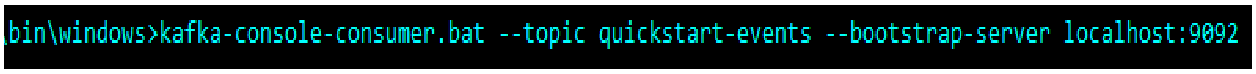


```
bin\windows>kafka-console-producer.bat --topic quickstart-events --bootstrap-server localhost:9092
```

Now you can write events(messages) on the console.

Read Events from Topic

To read events from a topic we need a consumer. The following command enables you to read events by specifying topic name and broker server address.



```
bin\windows>kafka-console-consumer.bat --topic quickstart-events --bootstrap-server localhost:9092
```

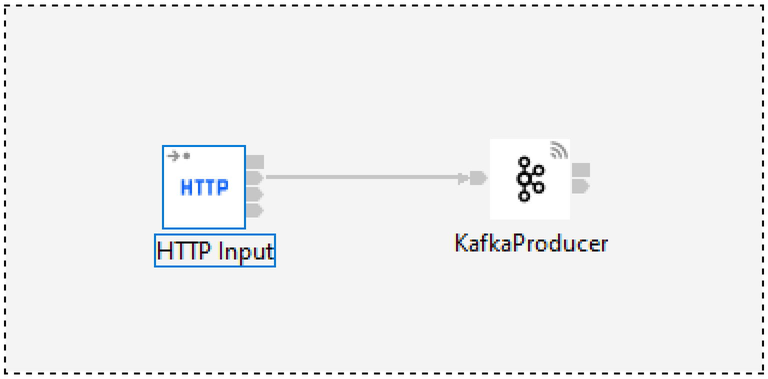
Integrate apache kafka in ibm app connect

In app connect, Kafka Producer, Consumer, Read nodes are available to work with kafka brokers.

Write a message into a topic

Preparation

Create an Application, and a message flow.
Use a kafka producer node to write messages into a Topic.



Define the required properties on the HTTP Input node and KafkaProducer node.

HTTP Input Node:

Basic - Path Suffix URL : /apache-kafka/produce

InputMessage Parsing - Message Domain : JSON

KafkaProducer:

Basic - Topic Name : quickstart-events , BootStrapServer : localhost:9092

Deploy and Run

Now it is ready to deploy into the integration server and test it by sending a message to the http input node.
URL : http://localhost:7800/apache-kafka/produce