**Apache Kafka**

**Apache Kafka is like a communication system that helps different parts of a computer system exchange data by publishing and subscribing to topics.**

**A diagram of a computer

AI-generated content may be incorrect.**

1. Sender publishes the message in Apache Kafka (That one who produces the message is called a producer.)
2. Receiver receive the message (That one who consume the message is called a consumer.)
3. The receiver will not be able to directly receive the message because the sender has published the message, so the receiver has to subscribe to it.

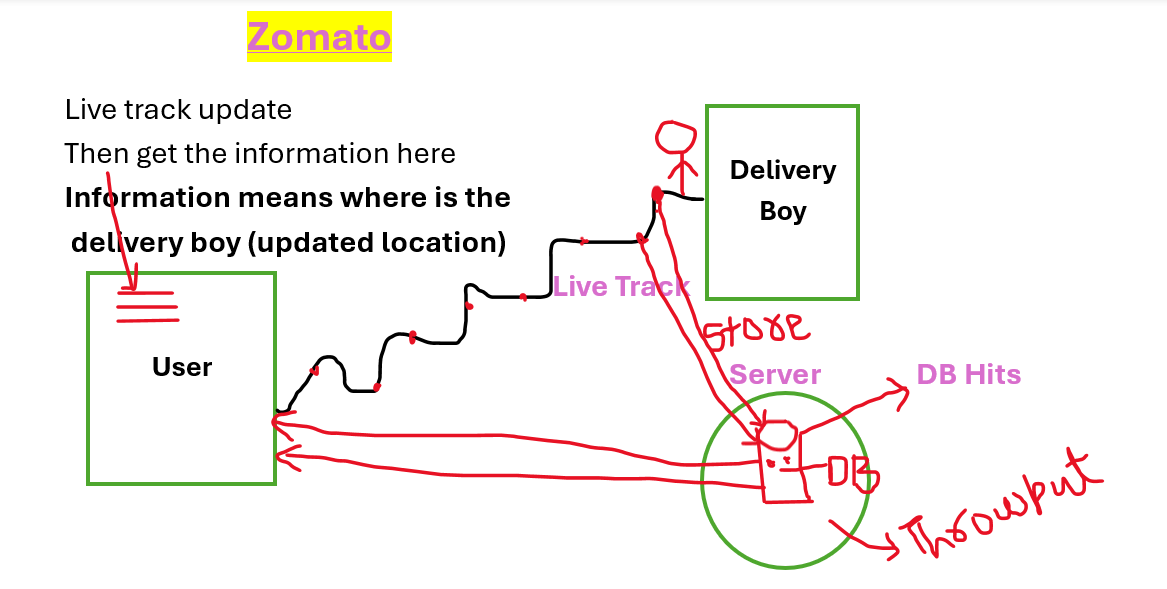
This means the sender is a type of producer who is producing the message.

while the receiver is a type of consumer who is consuming the message. But it will consume the message by subscribing to the topics or Apache Kafka.

1. The Apache Kafka here is delivering the sender's messages to the receiver with the help of the publish and subscribe model.

* **Let’s Understand with Example**

1. **OLA Driver location update**
2. **Zomato live food tracking**

****

**NOTE:** The database will crash if we keep updating the database repeatedly on location update.

**For Solving this problem, we are using Kafka.**

**A diagram of a diagram

AI-generated content may be incorrect.**

* ***Why We are using Kafka:***

1. ***High Throughput (for live tracking system, for huge amount of messaging system)***
2. ***Fault Tolerance (Replication) – Here keep data as a copy if any node has been deleted then no need to worry about it.***
3. ***Durable - It is durable because the data does not disappear; your data is available on multiple nodes, so its durability is higher.***
4. ***Scalable -*** **Its throughput is very high, so it can be easily scaled.**

**Kafka Architecture:**

***A diagram of a broken and broken system

AI-generated content may be incorrect.***

**STEPS:**

* **Producer will produce the message into Kafka.**
* **Consumer will consume the message by using subscribe the Topic.**
* **Where will data be produced?**

**- Inside Kafka. So, the first thing that comes to Kafka is the Ecosystem. In the Kafka Ecosystem, there will be Kafka clusters and Zookeeper components.**

* **Zookeeper manages the state of your Kafka and manages the Kafka brokers. Zookeeper is a type of software that helps Apache Kafka manage the state of brokers.**
* **The Kafka Cluster is our software that manages the Apache Kafka servers, which we call brokers. And we can have multiple brokers.**
* **The broker is the main server of Apache Kafka, and it is where our data comes in.**
* **To store data, brokers have Topics. There can be multiple Topics to store data, and data will come under the Topics.**

**E.g.: You can think of it like using a database, where we create tables under the database. Similarly, we create Topics under the broker. We understand the Broker as the Database, and under the database, we create tables to organize the data or categories the data.**

* **Similarly, we create topics under the broker. We understand the broker as a database, and within the database, we create tables to organize or categorize the data. Likewise, we will see topics under the broker for organizing or categorizing data.  
  E.g.: Since we have messages being published continuously, one message is related to the login system, one message is related to the order, and one message is related to the user. So, we will create 3 topics here. All the messages related to login will be placed under the login topic, and all the messages related to the user will be placed under the user topic. And all the messages related to the order will be placed under the order topic.  
  Topic is mainly used for categories the data.**
* **A lot of messages can come under one topic, so we use partitions under the topic to store data. Data is stored and managed with the help of offsets. As messages come in, they will be stored under the partition, starting with index 0, then 1, then 2, etc., which we call offsets.**

***Installation:***

1. ***Download Kafka zip file from official website***

***Link-*** [***https://archive.apache.org/dist/kafka/3.5.0/kafka-3.5.0-src.tgz.sha512***](https://archive.apache.org/dist/kafka/3.5.0/kafka-3.5.0-src.tgz.sha512)

1. ***Extract file***
2. ***Start Zookeeper***

***e.g. C:\Program Files\kafka-3.5.0-src>bin\windows\zookeeper-server-start.bat config\zookeeper.properties***

1. ***Start Kafka Server***

***Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.  
  
Steps for Kafka:***

[***Step 1: Get Kafka***](https://kafka.apache.org/quickstart#quickstart_download)

[***Download***](https://www.apache.org/dyn/closer.cgi?path=/kafka/4.1.0/kafka_2.13-4.1.0.tgz)***the latest Kafka release and extract it:***

***$ tar -xzf kafka\_2.13-4.1.0.tgz***

***$ cd kafka\_2.13-4.1.0***

[***Step 2: Start the Kafka environment***](https://kafka.apache.org/quickstart#quickstart_startserver)

***NOTE: Your local environment must have Java 17+ installed.***

***Kafka can be run using local scripts and downloaded files or the docker image.***

***Using downloaded files***

***Generate a Cluster UUID***

***$ KAFKA\_CLUSTER\_ID="$(bin/kafka-storage.sh random-uuid)"***

***Format Log Directories***

***$ bin/kafka-storage.sh format --standalone -t $KAFKA\_CLUSTER\_ID -c config/server.properties***

***Start the Kafka Server***

***$ bin/kafka-server-start.sh config/server.properties***

***Once the Kafka server has successfully launched, you will have a basic Kafka environment running and ready to use.***

***Using JVM Based Apache Kafka Docker Image***

***Get the Docker image:***

***$ docker pull apache/kafka:4.1.0***

***Start the Kafka Docker container:***

***$ docker run -p 9092:9092 apache/kafka:4.1.0***

***Using GraalVM Based Native Apache Kafka Docker Image***

***Get the Docker image:***

***$ docker pull apache/kafka-native:4.1.0***

***Start the Kafka Docker container:***

***$ docker run -p 9092:9092 apache/kafka-native:4.1.0***

[***Step 3: Create a topic to store your events***](https://kafka.apache.org/quickstart#quickstart_createtopic)

***Kafka is a distributed event streaming platform that lets you read, write, store, and process***[***events***](https://kafka.apache.org/documentation/#messages)***(also called records or messages in the documentation) across many machines.***

***Example events are payment transactions, geolocation updates from mobile phones, shipping orders, sensor measurements from IoT devices or medical equipment, and much more. These events are organized and stored in***[***topics***](https://kafka.apache.org/documentation/#intro_concepts_and_terms)***. Very simplified, a topic is similar to a folder in a filesystem, and the events are the files in that folder.***

***So, before you can write your first events, you must create a topic. Open another terminal session and run:***

***$ bin/kafka-topics.sh --create --topic quickstart-events --bootstrap-server localhost:9092***

***All of Kafka's command line tools have additional options: run the kafka-topics.sh command without any arguments to display usage information. For example, it can also show you***[***details such as the partition count***](https://kafka.apache.org/documentation/#intro_concepts_and_terms)***of the new topic:***

***$ bin/kafka-topics.sh --describe --topic quickstart-events --bootstrap-server localhost:9092***

***Topic: quickstart-events TopicId: NPmZHyhbR9y00wMglMH2sg PartitionCount: 1 ReplicationFactor: 1 Configs:***

***Topic: quickstart-events Partition: 0 Leader: 0 Replicas: 0 Isr: 0***

[***Step 4: Write some events into the topic***](https://kafka.apache.org/quickstart#quickstart_send)

***A Kafka client communicates with the Kafka brokers via the network for writing (or reading) events. Once received, the brokers will store the events in a durable and fault-tolerant manner for as long as you need—even forever.***

***Run the console producer client to write a few events into your topic. By default, each line you enter will result in a separate event being written to the topic.***

***$ bin/kafka-console-producer.sh --topic quickstart-events --bootstrap-server localhost:9092***

***>This is my first event***

***>This is my second event***

***You can stop the producer client with Ctrl-C at any time.***

[***Step 5: Read the events***](https://kafka.apache.org/quickstart#quickstart_consume)

***Open another terminal session and run the console consumer client to read the events you just created:***

***$ bin/kafka-console-consumer.sh --topic quickstart-events --from-beginning --bootstrap-server localhost:9092***

***This is my first event***

***This is my second event***

***You can stop the consumer client with Ctrl-C at any time.***

***Feel free to experiment for example, switch back to your producer terminal (previous step) to write additional events, and see how the events immediately show up in your consumer terminal.***

***Because events are durably stored in Kafka, they can be read as many times and by as many consumers as you want. You can easily verify this by opening yet another terminal session and re-running the previous command again.***

[***Step 6: Import/export your data as streams of events with Kafka Connect***](https://kafka.apache.org/quickstart#quickstart_kafkaconnect)

***You probably have lots of data in existing systems like relational databases or traditional messaging systems, along with many applications that already use these systems.***[***Kafka Connect***](https://kafka.apache.org/documentation/#connect)***allows you to continuously ingest data from external systems into Kafka, and vice versa. It is an extensible tool that runs connectors, which implement the custom logic for interacting with an external system. It is thus very easy to integrate existing systems with Kafka. To make this process even easier, there are hundreds of such connectors readily available.***

***In this quickstart we'll see how to run Kafka Connect with simple connectors that import data from a file to a Kafka topic and export data from a Kafka topic to a file.***

***First, make sure to add connect-file-4.1.0.jar to the plugin.path property in the Connect worker's configuration. For the purpose of this quickstart we'll use a relative path and consider the connectors' package as an uber jar, which works when the quickstart commands are run from the installation directory. However, it's worth noting that for production deployments using absolute paths is always preferable. See***[***plugin.path***](https://kafka.apache.org/documentation/#connectconfigs_plugin.path)***for a detailed description of how to set this config.***

***Edit the config/connect-standalone.properties file, add or change the plugin.path configuration property match the following, and save the file:***

***$ echo "plugin.path=libs/connect-file-4.1.0.jar" >> config/connect-standalone.properties***

***Then, start by creating some seed data to test with:***

***$ echo -e "foo\nbar" > test.txt***

***Or on Windows:***

***$ echo foo > test.txt***

***$ echo bar >> test.txt***

***Next, we'll start two connectors running in standalone mode, which means they run in a single, local, dedicated process. We provide three configuration files as parameters. The first is always the configuration for the Kafka Connect process, containing common configuration such as the Kafka brokers to connect to and the serialization format for data. The remaining configuration files each specify a connector to create. These files include a unique connector name, the connector class to instantiate, and any other configuration required by the connector.***

***$ bin/connect-standalone.sh config/connect-standalone.properties config/connect-file-source.properties config/connect-file-sink.properties***

***These sample configuration files, included with Kafka, use the default local cluster configuration you started earlier and create two connectors: the first is a source connector that reads lines from an input file and produces each to a Kafka topic and the second is a sink connector that reads messages from a Kafka topic and produces each as a line in an output file.***

***During startup you'll see a number of log messages, including some indicating that the connectors are being instantiated. Once the Kafka Connect process has started, the source connector should start reading lines from test.txt and producing them to the topic connect-test, and the sink connector should start reading messages from the topic connect-test and write them to the file test.sink.txt. We can verify the data has been delivered through the entire pipeline by examining the contents of the output file:***

***$ more test.sink.txt***

***foo***

***bar***

***Note that the data is being stored in the Kafka topic connect-test, so we can also run a console consumer to see the data in the topic (or use custom consumer code to process it):***

***$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic connect-test --from-beginning***

***{"schema":{"type":"string","optional":false},"payload":"foo"}***

***{"schema":{"type":"string","optional":false},"payload":"bar"}***

***…***

***The connectors continue to process data, so we can add data to the file and see it move through the pipeline:***

***$ echo "Another line" >> test.txt***

***You should see the line appear in the console consumer output and in the sink file.***

[***Step 7: Process your events with Kafka Streams***](https://kafka.apache.org/quickstart#quickstart_kafkastreams)

***Once your data is stored in Kafka as events, you can process the data with the***[***Kafka Streams***](https://kafka.apache.org/documentation/streams)***client library for Java/Scala. It allows you to implement mission-critical real-time applications and microservices, where the input and/or output data is stored in Kafka topics. Kafka Streams combines the simplicity of writing and deploying standard Java and Scala applications on the client side with the benefits of Kafka's server-side cluster technology to make these applications highly scalable, elastic, fault-tolerant, and distributed. The library supports exactly-once processing, stateful operations and aggregations, windowing, joins, processing based on event-time, and much more.***

***To give you a first taste, here's how one would implement the popular WordCount algorithm:***

***KStream<String, String> textLines = builder.stream("quickstart-events");***

***KTable<String, Long> wordCounts = textLines***

***.flatMapValues(line -> Arrays.asList(line.toLowerCase().split(" ")))***

***.groupBy((keyIgnored, word) -> word)***

***.count();***

***wordCounts.toStream().to("output-topic", Produced.with(Serdes.String(), Serdes.Long()));***

***The***[***Kafka Streams demo***](https://kafka.apache.org/documentation/streams/quickstart)***and the***[***app development tutorial***](https://kafka.apache.org/41/documentation/streams/tutorial)***demonstrate how to code and run such a streaming application from start to finish.***

[***Step 8: Terminate the Kafka environment***](https://kafka.apache.org/quickstart#quickstart_kafkaterminate)

***Now that you reached the end of the quickstart, feel free to tear down the Kafka environment—or continue playing around.***

1. ***Stop the producer and consumer clients with Ctrl-C, if you haven't done so already.***
2. ***Stop the Kafka broker with Ctrl-C.***

***If you also want to delete any data of your local Kafka environment including any events you have created along the way, run the command:***

***$ rm -rf /tmp/kafka-logs /tmp/kraft-combined-logs***

[***Congratulations!***](https://kafka.apache.org/quickstart#quickstart_kafkacongrats)

***You have successfully finished the Apache Kafka quickstart.***

***To learn more, we suggest the following next steps:***

* ***Read through the brief***[***Introduction***](https://kafka.apache.org/intro)***to learn how Kafka works at a high level, its main concepts, and how it compares to other technologies. To understand Kafka in more detail, head over to the***[***Documentation***](https://kafka.apache.org/documentation/)***.***
* ***Browse through the***[***Use Cases***](https://kafka.apache.org/powered-by)***to learn how other users in our world-wide community are getting value out of Kafka.***