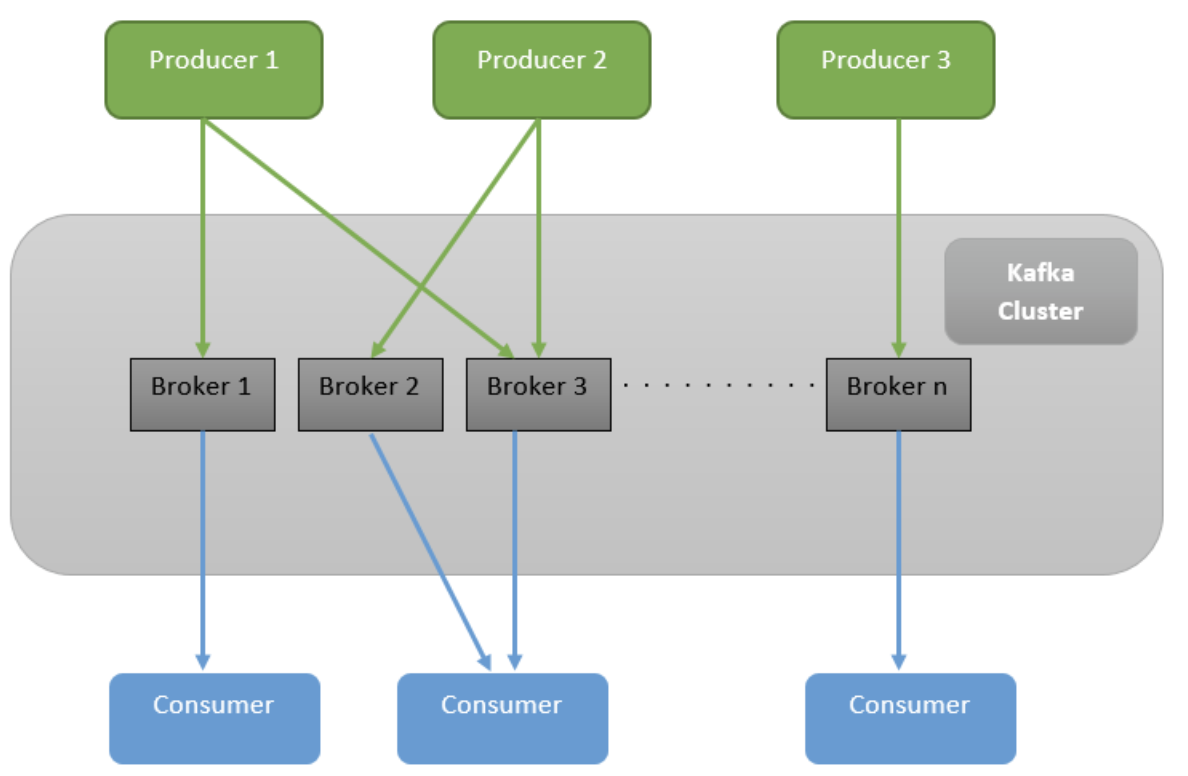
**Develop a Java Application with Kafka:**

Apache Kafka is a distributed streaming platform that utilizes the publish/subscribe message pattern to interact with applications; it’s designed to create durable messages.

### **Publish/Subscribe Messaging System With Durable Messages:**

The publish/subscribe is a common pattern in distributed systems. The image below illustrates the basic structure of this pattern within Kafka:



The image includes two components not mentioned so far: Producers and Consumers.

A Producer is an application that sends messages to the cluster. In this example, Producer 1, 2, and 3 are sending messages. The cluster then elects which broker should store them and sends it to the ones selected.

On the other side, you have consumers. A consumer is an application that connects to the cluster and receives the messages posted from producers. Any application that is interested in consuming messages sent by producers must connect into the Kafka consumer.

As Kafka stores messages for long durations (the default value is seven days), you can have many consumers receiving the same message even if they were not there when the message was sent!

### **Kafka Topics:**

When you send a message to a [Kafka broker](https://dzone.com/articles/kafka-producer-and-consumer-example), you need to specify where the message will be sent by specifying a topic. A topic is a category of messages that a consumer can subscribe to. This mechanism ensures that consumers only receive messages relevant to them, rather than receiving every message published to the cluster.

Now that you understand Kafka’s basic architecture, let’s download and install it.

## **Install and Run Kafka**

To download Kafka, [go to the Kafka website](https://kafka.apache.org/downloads). Extract the contents of this compressed file into a folder of your preference.

Inside the Kafka directory, go to the bin folder. Here, you’ll find many bash scripts that will be useful for running a Kafka application. If you are using Windows, you also have the same scripts inside the windows folder. This tutorial uses Linux commands, but you just need to use the equivalent Windows version if you’re running a Microsoft OS.

**start the Zookeeper by the command line**

->E:\apachekafka\kafka\_2.12-2.4.0\bin\windows>zookeeper-server-start.bat E:\apachekafka\kafka\_2.12-2.4.0\config\zookeeper.properties

**Start Kafka Server by the command line**

->E:\apachekafka\kafka\_2.12-2.4.0\bin\windows>kafka-server-start.bat E:\apachekafka\kafka\_2.12-2.4.0\config\server.properties

**create the topic by the command line**

->E:\apachekafka\kafka\_2.12-2.4.0\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic usertopic (**topic name**)

**to see the list of topics in the brocker by entering the command by command line**

->E:\apachekafka\kafka\_2.12-2.4.0\bin\windows>kafka-topics.bat --list --zookeeper localhost:2181

**to see the in details about the topic :**

->E:\apachekafka\kafka\_2.12-2.4.0\bin\windows>kafka-topics.bat --zookeeper localhost:2181 --describe --topic usertopic(topic name)

remove the topic:

Use kafka-topics.bat --delete to delete usertopic topic.

->E:\apachekafka\kafka\_2.12-2.4.0\bin\windows>kafka-topics.bat kafka-topics.bat --zookeeper localhost:2181 --delete --topic usertopic

after enter the command

**o/p is like**

Topic usertopic is marked for deletion.

**Note**: This will have no impact if delete.topic.enable is not set to true.

**see the list of topics:**

->E:\apachekafka\kafka\_2.12-2.4.0\bin\windows>kafka-topics.bat --list --zookeeper localhost:2181

\_\_consumer\_offsets

usertopic - marked for deletion

As you may have noticed, kafka-topics.bat --delete will only delete a topic if the topic’s leader broker is available (and can acknowledge the removal). Since the broker 100 is down and currently unavailable the topic deletion has only been recorded in Zookeeper.

$ zookeeper-shall.bat localhost:2181

ls /admin/delete\_topics

[usertopic]

**how to see the kafaka Producer message in the topic**

E:\apachekafka\kafka\_2.12-2.4.0\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 --usertopic

Spring Cloud Stream:

Spring Cloud Stream is a framework for building message-driven microservice applications. Spring Cloud Stream builds upon Spring Boot to create standalone, production-grade Spring applications and uses Spring Integration to provide connectivity to message brokers.

@EnableBinding: we can add this annotation in the application to get the immediate Connectivity to a message broker.

@StreamListener: you can add to a method to cause it to receive events for stream processing.

The @EnableBinding annotation takes one or more interfaces as parameters. An interface declares input and output channels. Spring Cloud Stream provides the Source, Sink, and Processor interfaces.

We can create our own interface

public interface Sink {

String INPUT = "input";

@Input(Sink.INPUT)

SubscribableChannel input();

}

The @Input annotation identifies an input channel, through which received messages enter the application. The @Output annotation identifies an output channel, through which published messages leave the application. The @Input and @Output annotations can take a channel name as a parameter. If a name is not provided, the name of the annotated method is used.

Spring Cloud Stream provides binding interfaces for typical message exchange contracts, which include:

* **Sink:** Identifies the contract for the message consumer by providing the destination from which the message is consumed.
* **Source:** Identifies the contract for the message producer by providing the destination to which the produced message is sent.
* **Processor:** Encapsulates both the sink and the source contracts by exposing two destinations that allow consumption and production of messages.

## How many types of Kafka binders are there under Spring Cloud Stream?

Spring Cloud Stream provides two separate binders for Kafka :

1. **spring-cloud-stream-binder-Kafka** and
2. **spring-cloud-stream-binder-kafka-streams**.

The first one is the one that you want to use if you want to write standard event-driven applications in which you want to use normal Kafka producers and consumers. On the other hand, if you want to develop stream processing applications with the Kafka Streams library, use the second binder.

## Application ID:

@SpringBootApplication

public class SimpleConsumerApplication {

@Bean

public java.util.function.Consumer<KStream<String, String>> process() {

return input ->

input.foreach((key, value) -> {

System.out.println("Key: " + key + " Value: " + value);

});

}

}

In a Kafka Streams application, application.id is a mandatory field. Without it, you cannot start a Kafka Streams application. By default, the binder will generate an application ID and assign it to the processor. It uses the function bean name as a prefix. For e.g, if you have a consumer as above, the binder will generate the application ID as process-applicationId.

## Topic to consume from

For the above processor, you can provide the topic to consumes, as follows

spring.cloud.stream.bindings.process-in-0.destination: my-input-topic

In this case, we are saying that, for the function bean (process) and its first input (in-0), it shall be bound to a Kafka topic named **my-input-topic**. If you don’t provide an explicit destination like this, the binder assumes that you are using a topic that is the same as the binding name (**process-in-0**, in this case).