

Apache Kafka 3.8 Installation on Windows OS

Table of Contents

1. Overview:	4
2. Prerequisites:	5
3. Install Kafka:	5
3.1. Download Kafka Binaries:	5
3.2. Set up Environment variables:	7
3.3. Verify Kafka Installation:	11
4. Configure Single Node-Single Broker:	12
4.1. Configure Kafka Logs:	12
4.2. Start Kafka Servers:	13
4.2.1. Start Zookeeper:	13
4.2.2. Start Kafka Broker:	14
4.3. Create Topic:	16
4.4. List Topics:	18
4.5. Describe Topics:	18
4.6. Run Kafka Producer:	18
4.7. Run Kafka Consumer:	19
4.8. Delete Topic:	20
5. Configure Single Node-Multiple Brokers:	24
5.1. Setup Multiple Brokers:	25
5.2. Start Kafka Servers:	26
5.2.1. Start Zookeeper:	26
5.2.2. Start Kafka Brokers:	28
5.3. Create Topic:	31
5.4. Describe Topic:	31
5.5. Run Kafka Producer:	32
5.6. Run Kafka Consumer:	32
5.7. Delete Topic:	33
6. Simple Kafka Project using Java:	38
6.1. Launch Java IDE:	38

6.2. Create Topic:	40
6.2.1. List Topics:	42
6.2.2. Describe Topics:	43
6.2.3. Delete Topics:	44
6.3. Create Producer:	45
6.3.1. Producer Callbacks:	48
6.3.2. Produce Data with Same Key:	50
6.3.3. Produce Data with Different Keys:	52
6.4. Create Consumer:	53
7. Spark Streaming with Kafka:	58
7.1. Verify Spark Version:	59
7.2. Start Spark Services:	60
7.3. Create Maven Java Project:	61
7.4. Create Spark Application:	65
7.5. Execute Spark Application:	70

This document outlines the steps needed to install **Apache Kafka 3.8** with single broker instance configuration and multi-broker instance configuration and integrate with Apache Spark on Windows Operating system.

1. Overview:

Apache Kafka is an open-source distributed streaming processing platform used to handle the real time data storage with a processing capacity of millions of data or messages per second. Apache Kafka was originally developed at [LinkedIn](#) and later donated to **Apache Software Foundation** and is currently maintained by [Confluent](#) under Apache Software Foundation.

Apache Kafka works as a **broker** between sender and receiver and provides a **messaging system** (*responsible for exchanging messages between two applications*) in publish-subscribe (pub-sub) model. In pub-sub model, a sender (*known as Producer or Publisher*) sends/writes a message (nothing but data) to a persisted location called **topic** and then receiver (*known as Consumer or Subscriber*) consumes a message by subscribing to a topic.

Apache Kafka include components such as **Topic, Broker, Producer, Consumer** and **Zookeeper** and provides the following features:

- Capable of handling millions of messages per second.
- Works as a mediator between source and target systems in which source system (producer) data is sent to Apache Kafka where it decouples data and target system (consumer) consumes data from Kafka.
- Extreme high performance with low latency value less than 10ms.
- Fault-tolerant, highly available and resilient to node failures.

We must also need to know the following five core APIs provided by Apache Kafka:

1. **Producer API:** This API allows an application to publish streams of records to one or more topics.
2. **Consumer API:** This API allows an application to subscribe to one or more topics and process the stream of records produced to them.
3. **Streams API:** This API allows an application to effectively transform streams of records from input topics to output topics (*i.e. an application acts as a stream processor to consume an input stream from one or more topics, and produce an output stream to one or more output topics*).
4. **Connector API:** This API allows implementing connectors to move large amounts of data from source system to Kafka or push from Kafka into some sink data system.
5. **Admin API:** This API allows us to manage and inspect topics, partitions, brokers, ACLs and other Kafka objects.

Apache Kafka is heavily used across many organizations such as Netflix, UBER, Walmart, Twitter, LinkedIn, Mozilla, Oracle etc. with several use cases such as:

- **Messaging:** Kafka works better as a replacement for traditional message broker since it provides publish-subscribe messaging system.
- **Metrics:** Kafka is used for operational monitoring data by aggregating statistics from distributed applications.
- **Event Sourcing:** Kafka is an excellent backend for applications of event sourcing since it supports very large stored data.
- **Log Aggregation:** Kafka can be used across organization to collect logs from multiple services and make them available in a standard format to multiple consumers.
- **Stream Processing:** Kafka's strong durability is useful for stream processing in case of some popular frameworks such as Storm and Spark Streaming to read data from a topic, process it and write processed data to a new topic which becomes available for users and applications.
- **Website Activity Tracking:** Kafka can be used as a user activity tracking pipeline of a website as a set of real-time publish-subscribe feeds where each site activity that includes page views, searches, etc. is published to respective central topics.

2. Prerequisites:

The following prerequisites need to be installed before running Kafka.

1. **File Archiver:** Any file archiver such as **7zip** or **WinRAR** is needed to unzip the downloaded Spark binaries. 7zip can be downloaded from the [7zip Downloads](#) website and WinRAR can be downloaded from the [RAR lab Downloads](#) website.
2. **JRE 8:** Kafka 3.x requires Java 8 runtime environment. We can either download just JRE 8 (Java Runtime Environment) for Windows offline installation from the official [Java Download for Windows Offline](#) website or download the whole JDK 8 (Java Development Kit) directly from [Oracle Java Downloads](#) website. For the complete JDK installation steps, look at [here](#).

3. Install Kafka:

Let us see the step by step procedure to install Apache Kafka in Windows.

3.1. Download Kafka Binaries:

Download the stable version of Kafka from the official [Apache Kafka Downloads](#) website. At the time of this document preparation, the most recent stable release is 3.8.0.

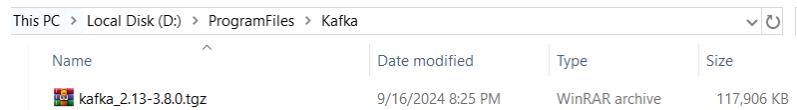
Here, we will find multiple Kafka build versions of Scala (*you can select either Scala 2.12 or Scala 2.13 version for the Scala version that you use*).

Let us select **Scala 2.13** version and so click on `kafka_2.13-3.8.0.tgz` link to download it into the **Downloads** folder in your machine.

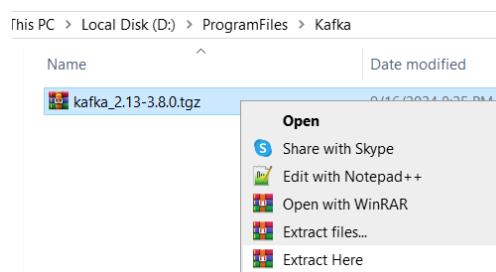
The screenshot shows the Apache Kafka Downloads page at <https://kafka.apache.org/downloads>. The page has a navigation bar with links for GET STARTED, DOCS, POWERED BY, COMMUNITY, and APACHE, and a prominent DOWNLOAD KAFKA button. Below the navigation is a section titled "DOWNLOAD" with a sub-section "SUPPORTED RELEASES". Under "3.8.0", there is a bulleted list of download options, including links for Scala 2.12 and Scala 2.13. The link for "Scala 2.13 kafka_2.13-3.8.0.tgz (asc, sha512)" is highlighted with a red box. A note below states: "We build for multiple versions of Scala. This only matters if you are using Scala and you want a version built for the same Scala version you use. Otherwise any version should work (2.13 is recommended)." At the bottom, a note says: "Kafka 3.8.0 includes a significant number of new features and fixes. For more information, please read our [blog post](#) and the detailed [Release Notes](#)".

After the binary file is downloaded, unpack it using any file archiver (**7zip** or **WinRAR**) utility as below:

- Choose the installation directory in your machine and copy `kafka_2.13-3.8.0.tgz` file to that directory. Here, we are choosing Kafka installation directory as `D:\ProgramFiles\Kafka`.



- Right click on `kafka_2.13-3.8.0.tgz` and choose **Extract Here** option.



- The tar file extraction may take few minutes to finish. After finishing, you see a folder named `kafka_2.13-3.8.0` which consists of Kafka binaries and libraries.

Name	Date modified	Type	Size
bin	7/23/2024 1:39 PM	File folder	
config	7/23/2024 1:39 PM	File folder	
libs	7/23/2024 1:39 PM	File folder	
licenses	7/23/2024 1:39 PM	File folder	
site-docs	7/23/2024 1:39 PM	File folder	
LICENSE	7/23/2024 1:34 PM	File	15 KB
NOTICE	7/23/2024 1:34 PM	File	28 KB

Note:

When your Kafka Installation path is too long, you might encounter an error “**The input line is too long**” while executing Kafka scripts. To avoid this error, move all folders and files from `D:\ProgramFiles\Kafka\kafka_2.13-3.8.0` location to `D:\ProgramFiles\Kafka` folder.

Name	Date modified	Type	Size
bin	7/23/2024 1:39 PM	File folder	
config	7/23/2024 1:39 PM	File folder	
kafka_2.13-3.8.0	9/16/2024 10:07 PM	File folder	
libs	7/23/2024 1:39 PM	File folder	
licenses	7/23/2024 1:39 PM	File folder	
site-docs	7/23/2024 1:39 PM	File folder	
kafka_2.13-3.8.0.tgz	9/16/2024 8:25 PM	WinRAR archive	117,906 KB
LICENSE	7/23/2024 1:34 PM	File	15 KB
NOTICE	7/23/2024 1:34 PM	File	28 KB

3.2. Set up Environment variables:

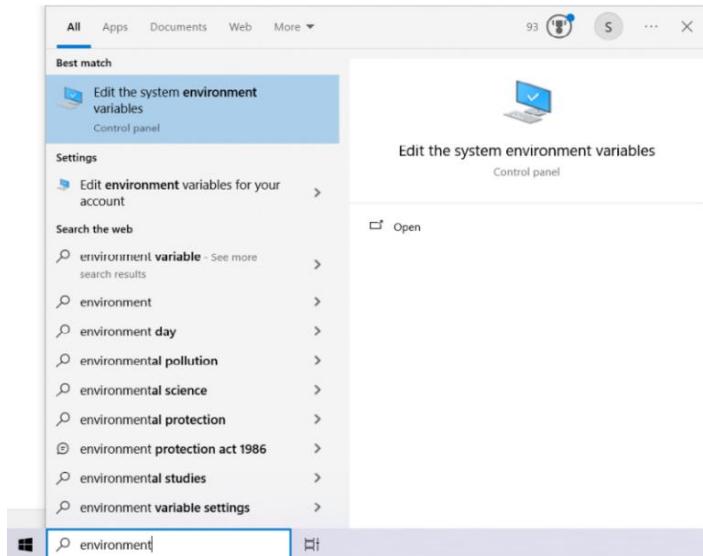
After installing pre-requisites and Kafka binaries, we should configure the below environment variables defining Java and Kafka default paths.

- JAVA_HOME:** This is the JDK installation directory path in the machine (*in my machine, it is D:\ProgramFiles\Java\jdk-1.8*). Ignore it if this is already done.
- KAFKA_HOME:** This is the Kafka installation directory path in the machine (*in our case, it is D:\ProgramFiles\Kafka*)

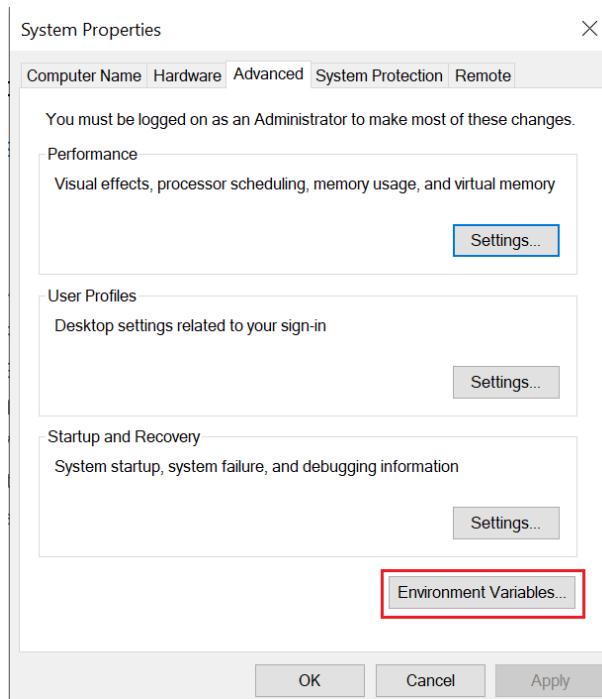
These variables need to be added to either **User environment variables** or **System environment variables** depending on Kafka configuration needed **for a single user** or **for multiple users**.

In this tutorial, we will add User environment variables since we are configuring Kafka for a single user. If you would like to configure Kafka for multiple users, then define System environment variables.

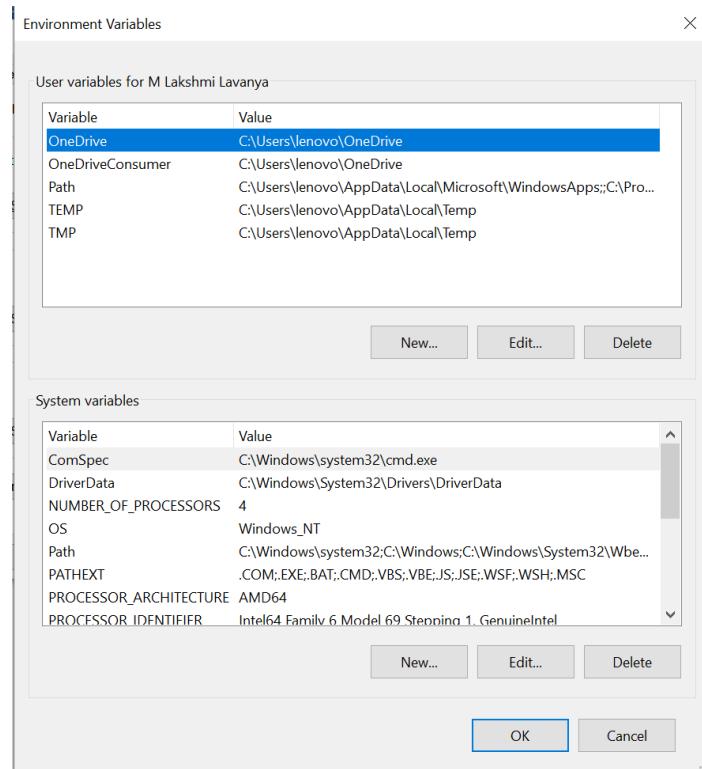
In the Windows search bar, start typing “environment variables” and select the first match which opens up **System Properties** dialog.



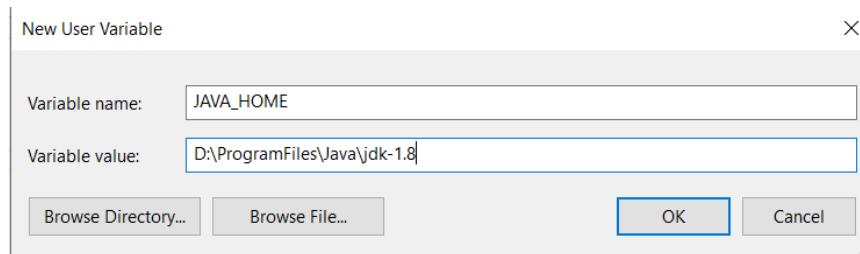
On the **System Properties** window, press **Environment Variables** button.



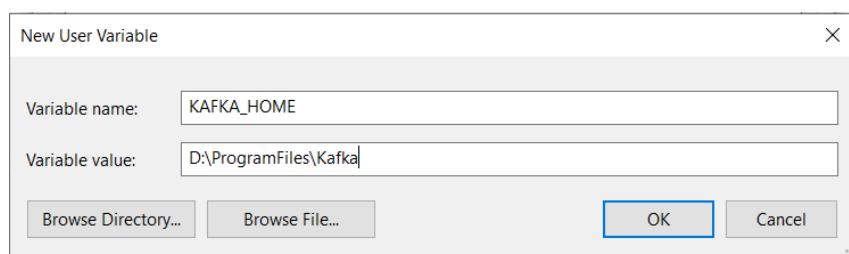
In the **Environment Variables** dialog, click on **New** under **User variables** section.



Add JAVA_HOME variable and press OK.

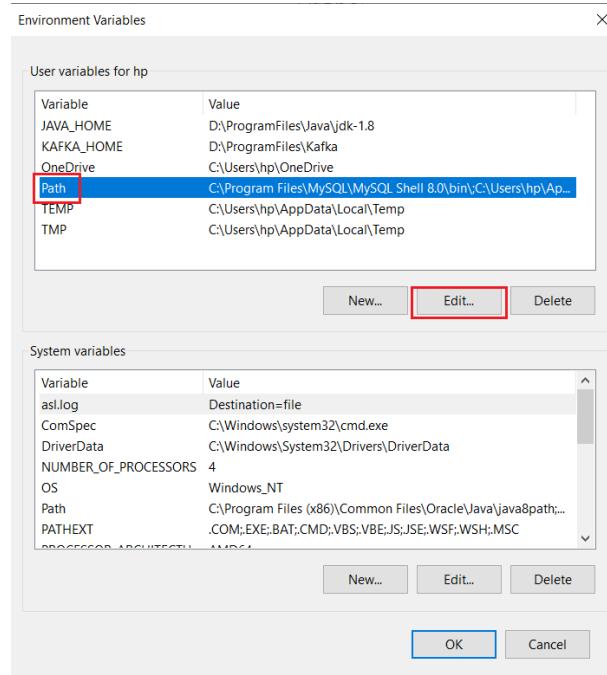


Click on **New** again and add KAFKA_HOME variable and press OK.



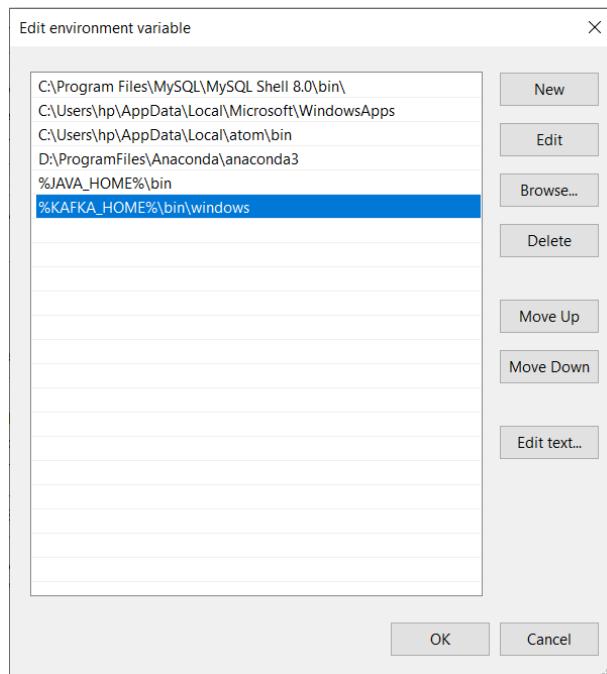
Now, we will update PATH variable to add Java, and Kafka binary paths.

Select PATH variable under **User Variables** and press **Edit** button.

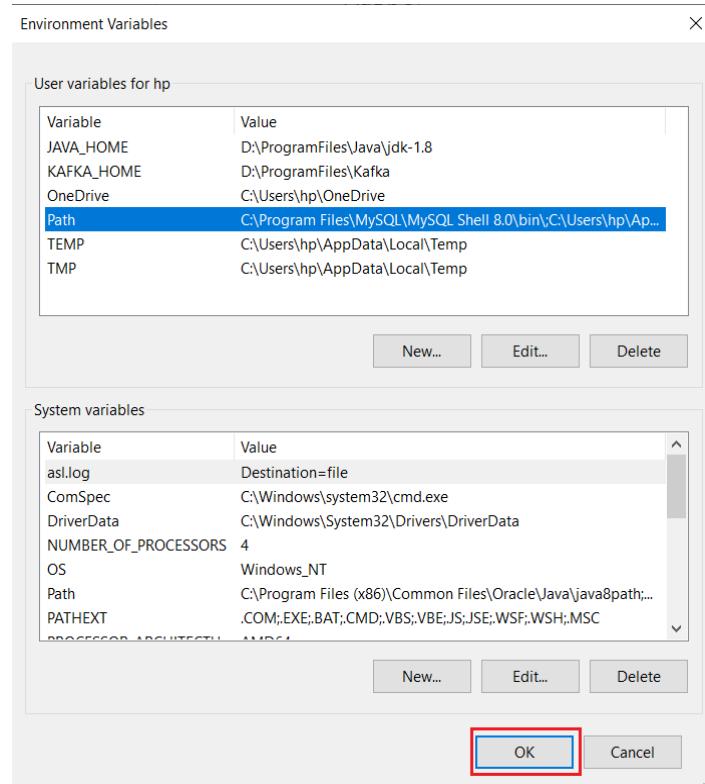


Press **New** and add the following values and press **OK**.

%JAVA_HOME%\bin
%KAFKA_HOME%\bin\windows



Press OK again to apply environment variable changes and close window.



3.3. Verify Kafka Installation:

Open **Windows PowerShell** and run the following command to verify if Kafka is installed properly:

```
kafka-topics.bat --version
```

A screenshot of a Windows PowerShell window. The title bar says 'Windows PowerShell'. The command 'kafka-topics.bat --version' is entered at the prompt. The output shows the version '3.8.0'.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\hp> kafka-topics.bat --version
3.8.0
PS C:\Users\hp>
```

Here, it shows **3.8.0** which indicates that Kafka has been installed successfully.

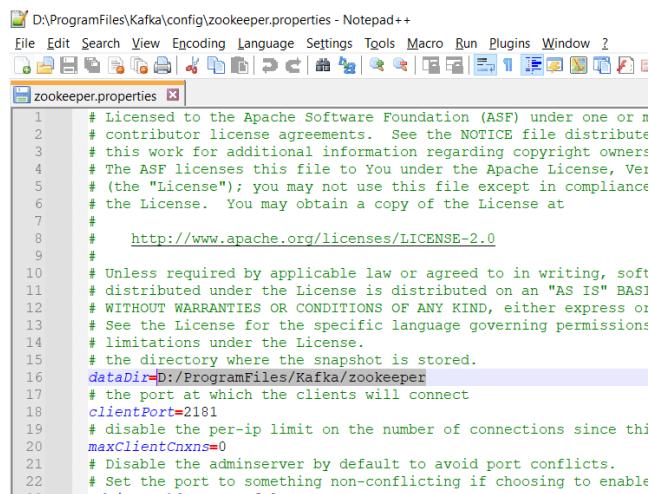
4. Configure Single Node-Single Broker:

In this configuration, we will setup a single Zookeeper with single Broker instance running on a single machine.

4.1. Configure Kafka Logs:

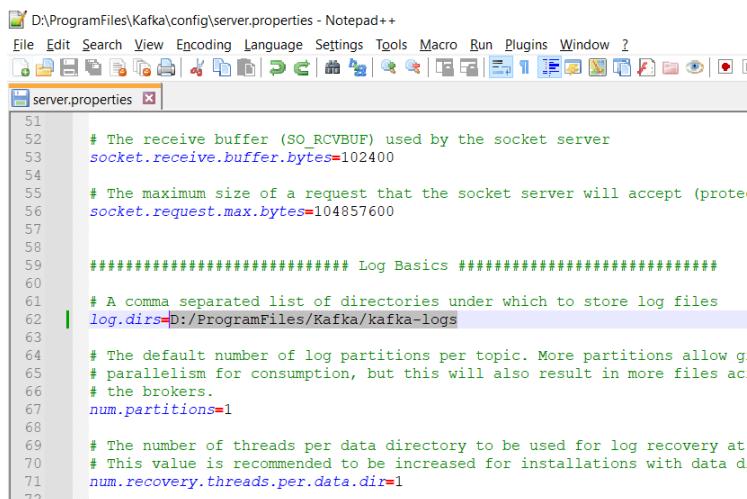
By default `zookeeper.properties` and `server.properties` files are configured with default directory locations that should be updated to the desired locations.

Open `zookeeper.properties` file in `%KAFKA_HOME%\config` folder and change the `dataDir` value from `/tmp/zookeeper` to `D:/ProgramFiles/Kafka/zookeeper` as shown below:



```
D:\ProgramFiles\Kafka\config\zookeeper.properties - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
zookeeper.properties
1 # Licensed to the Apache Software Foundation (ASF) under one or more
2 # contributor license agreements. See the NOTICE file distributed
3 # this work for additional information regarding copyright ownership.
4 # The ASF licenses this file to You under the Apache License, Version
5 # (the "License"); you may not use this file except in compliance
6 # with the License. You may obtain a copy of the License at
7 #
8 #     http://www.apache.org/licenses/LICENSE-2.0
9 #
10 # Unless required by applicable law or agreed to in writing, software
11 # distributed under the License is distributed on an "AS IS" BASIS
12 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
13 # implied. See the License for the specific language governing permissions
14 # and limitations under the License.
15 # the directory where the snapshot is stored.
16 dataDir=D:/ProgramFiles/Kafka/zookeeper
17 # the port at which the clients will connect
18 clientPort=2181
19 # disable the per-ip limit on the number of connections since this
20 maxClientCnxs=0
21 # Disable the adminserver by default to avoid port conflicts.
22 # Set the port to something non-conflicting if choosing to enable
23 #-----
```

Next, open `server.properties` file in `%KAFKA_HOME%\config` folder and scroll down to `log.dirs` and change the value from `/tmp/kafka-logs` to `D:/ProgramFiles/Kafka/kafka-logs` as shown below:



```
D:\ProgramFiles\Kafka\config\server.properties - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
server.properties
51 #
52 # The receive buffer (SO_RCVBUF) used by the socket server
53 socket.receive.buffer.bytes=102400
54 #
55 # The maximum size of a request that the socket server will accept (proto)
56 socket.request.max.bytes=104857600
57 #
58 ###### Log Basics #####
59 #
60 # A comma separated list of directories under which to store log files
61 log.dirs=D:/ProgramFiles/Kafka/kafka-logs
62 #
63 # The default number of log partitions per topic. More partitions allow greater
64 # parallelism for consumption, but this will also result in more files across
65 # the brokers.
66 num.partitions=1
67 #
68 # The number of threads per data directory to be used for log recovery at
69 # This value is recommended to be increased for installations with data dirs >
70 num.recovery.threads.per.data.dir=1
71 #
72
```

Note that `kafka-logs` and `zookeeper` directories will be created automatically in `%KAFKA_HOME%` location when Zookeeper and Kafka broker services are started.

4.2. Start Kafka Servers:

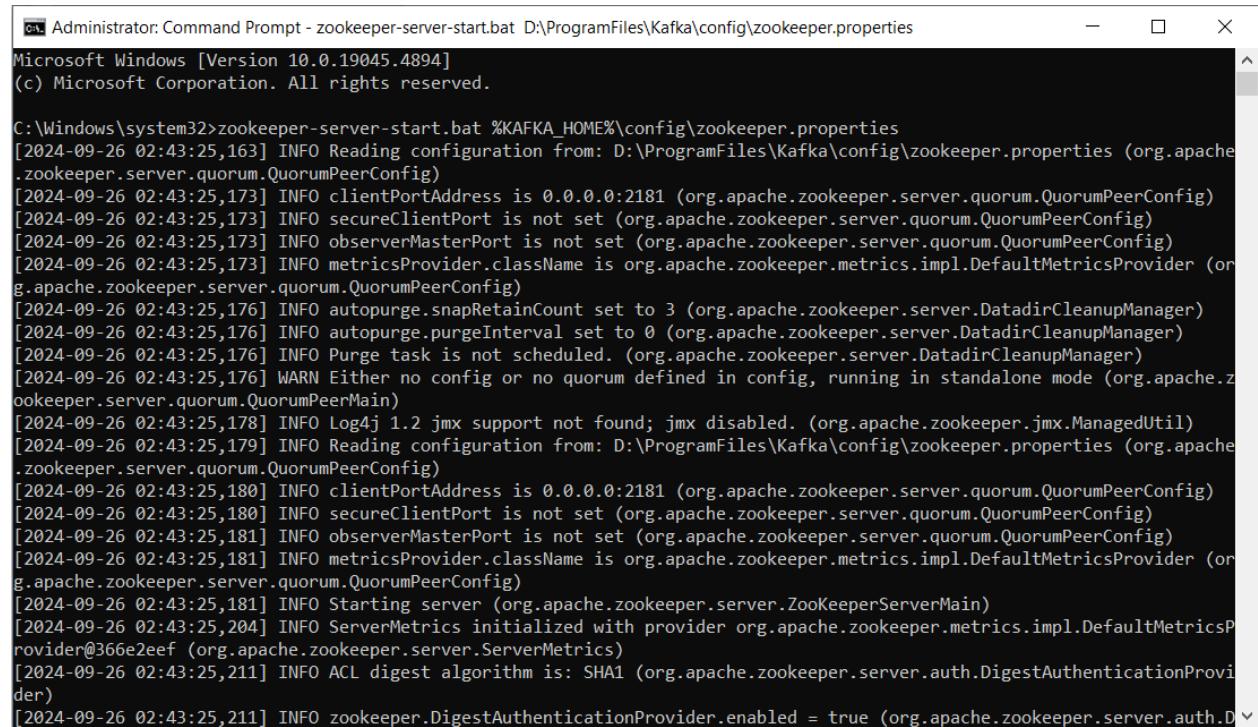
Now, we will start the Zookeeper and Kafka Broker services in the local system.

4.2.1. Start Zookeeper:

The Zookeeper is a centralized service in Apache Kafka that manages configuration changes and provides distributed configuration. It is responsible for managing the Kafka cluster by performing broker leader election and provides metadata to brokers about the processes running in the system and facilitate health checking of each broker in the cluster.

Open **Command Prompt** as **Administrator** and start Zookeeper using the below command:

```
zookeeper-server-start.bat %KAFKA_HOME%\config\zookeeper.properties
```



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt - zookeeper-server-start.bat D:\ProgramFiles\Kafka\config\zookeeper.properties". The window displays the log output of the Zookeeper server starting up. The log includes various INFO messages from the Apache Zookeeper codebase, such as reading configuration from the properties file, setting client and secure ports, and initializing metrics providers. There are also some WARN messages indicating that no config or no quorum was defined in the config file, so it's running in standalone mode. The log ends with the server starting and initializing metrics.

```
Administrator: Command Prompt - zookeeper-server-start.bat D:\ProgramFiles\Kafka\config\zookeeper.properties
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>zookeeper-server-start.bat %KAFKA_HOME%\config\zookeeper.properties
[2024-09-26 02:43:25,163] INFO Reading configuration from: D:\ProgramFiles\Kafka\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,173] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,173] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,173] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,173] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,176] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-09-26 02:43:25,176] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-09-26 02:43:25,176] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-09-26 02:43:25,176] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2024-09-26 02:43:25,178] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2024-09-26 02:43:25,179] INFO Reading configuration from: D:\ProgramFiles\Kafka\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,180] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,180] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,181] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,181] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 02:43:25,181] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2024-09-26 02:43:25,204] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@366e2eef (org.apache.zookeeper.server.ServerMetrics)
[2024-09-26 02:43:25,211] INFO ACL digest algorithm is: SHA1 (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
[2024-09-26 02:43:25,211] INFO zookeeper.DigestAuthenticationProvider.enabled = true (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
```

```
[2024-09-26 02:43:25,310] INFO Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory (org.apache.zookeeper.server.ServerCnxnFactory)
[2024-09-26 02:43:25,312] WARN maxCnxns is not configured, using default value 0. (org.apache.zookeeper.server.ServerCnxnFactory)
[2024-09-26 02:43:25,317] INFO Configuring NIO connection handler with 10s sessionless connection timeout, 1 selector threads, 8 worker threads, and 64 kB direct buffers. (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-09-26 02:43:25,326] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-09-26 02:43:25,357] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)
[2024-09-26 02:43:25,357] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)
[2024-09-26 02:43:25,359] INFO zookeeper.snapshotSizeFactor = 0.33 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 02:43:25,359] INFO zookeeper.commitLogCount=500 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 02:43:25,372] INFO zookeeper.snapshot.compression.method = CHECKED (org.apache.zookeeper.server.persistence.SnapStream)
[2024-09-26 02:43:25,372] INFO Snapshotting: 0x0 to D:\ProgramFiles\Kafka\zookeeper\version-2\snapshot.0 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-09-26 02:43:25,379] INFO Snapshot loaded in 17 ms, highest zxid is 0x0, digest is 1371985504 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 02:43:25,385] INFO Snapshotting: 0x0 to D:\ProgramFiles\Kafka\zookeeper\version-2\snapshot.0 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-09-26 02:43:25,386] INFO Snapshot taken in 1 ms (org.apache.zookeeper.server.ZooKeeperServer)
[2024-09-26 02:43:25,407] INFO PrepRequestProcessor (sid:0) started, reconfigEnabled=false (org.apache.zookeeper.server.PrepRequestProcessor)
[2024-09-26 02:43:25,408] INFO zookeeper.request_throttler.shutdownTimeout = 10000 ms (org.apache.zookeeper.server.RequestThrottler)
[2024-09-26 02:43:25,437] INFO Using checkIntervalMs=60000 maxPerMinute=10000 maxNeverUsedIntervalMs=0 (org.apache.zookeeper.server.ContainerManager)
[2024-09-26 02:43:25,439] INFO ZooKeeper audit is disabled. (org.apache.zookeeper.audit.ZKAuditProvider)
```

On the console, you will see a message “**binding to port 0.0.0.0/0.0.0.0:2181**” which indicates that the Zookeeper is up and running. Do not close this command prompt window.

4.2.2. Start Kafka Broker:

The Kafka broker manages the storage of messages in topics. Kafka cluster typically consists of multiple brokers to maintain the load balance and Kafka broker leader election is done by Zookeeper.

Open another **Command Prompt** as **Administrator** and start Kafka broker using the below command:

```
kafka-server-start.bat %KAFKA_HOME%\config\server.properties
```

```
Administrator: Command Prompt - kafka-server-start.bat D:\ProgramFiles\Kafka\config\server.properties
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>kafka-server-start.bat %KAFKA_HOME%\config\server.properties
[2024-09-26 02:45:00,704] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2024-09-26 02:45:01,162] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2024-09-26 02:45:01,165] INFO RemoteLogManagerConfig values:
    log.local.retention.bytes = -2
    log.local.retention.ms = -2
    remote.fetch.max.wait.ms = 500
    remote.log.index.file.cache.total.size.bytes = 1073741824
    remote.log.manager.copier.thread.pool.size = 10
    remote.log.manager.copy.max.bytes.per.second = 9223372036854775807
    remote.log.manager.copy.quota.window.num = 11
    remote.log.manager.copy.quota.window.size.seconds = 1
    remote.log.manager.expiration.thread.pool.size = 10
    remote.log.manager.fetch.max.bytes.per.second = 9223372036854775807
    remote.log.manager.fetch.quota.window.num = 11
    remote.log.manager.fetch.quota.window.size.seconds = 1
    remote.log.manager.task.interval.ms = 30000
    remote.log.manager.task.retry.backoff.max.ms = 30000
    remote.log.manager.task.retry.backoff.ms = 500
    remote.log.manager.task.retry.jitter = 0.2
    remote.log.manager.thread.pool.size = 10
    remote.log.metadata.custom.metadata.max.bytes = 128
    remote.log.metadata.manager.class.name = org.apache.kafka.server.log.remote.metadata.storage.TopicBasedRemoteLog
MetadataManager
    remote.log.metadata.manager.class.path = null
(kafka.zk.KafkaZkClient)
[2024-09-26 02:45:04,560] INFO Registered broker 0 at path /brokers/ids/0 with addresses: PLAINTEXT://:9092,
czxid (broker epoch): 25 (kafka.zk.KafkaZkClient)
[2024-09-26 02:45:04,654] INFO [ExpirationReaper-0-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 02:45:04,668] INFO [ExpirationReaper-0-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 02:45:04,669] INFO [ExpirationReaper-0-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 02:45:04,703] INFO Successfully created /controller_epoch with initial epoch 0 (kafka.zk.KafkaZkClient)
[2024-09-26 02:45:04,738] INFO [GroupCoordinator 0]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 02:45:04,756] INFO [GroupCoordinator 0]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 02:45:04,783] INFO [TransactionCoordinator id=0] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 02:45:04,786] INFO Feature ZK node created at path: /feature (kafka.server.FinalizedFeatureChangeListener)
[2024-09-26 02:45:04,790] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 02:45:04,835] INFO [TxnMarkerSenderThread-0]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2024-09-26 02:45:04,886] INFO [MetadataCache brokerId=0] Updated cache from existing None to latest Features(metadataVersion=3.8-IV0, finalizedFeatures={}, finalizedFeaturesEpoch=0). (kafka.server.metadata.ZkMetadataCache)
[2024-09-26 02:45:05,102] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 02:45:05,144] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2024-09-26 02:45:05,188] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Enabling request processing. (kafka.network.SocketServer)
[2024-09-26 02:45:05,198] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.DataPlaneAcceptor)
[2024-09-26 02:45:05,204] INFO [KafkaServer id=0] Start processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 02:45:05,207] INFO [KafkaServer id=0] End processing authorizer futures (kafka.server.KafkaServer)
```

On the console, you will see a message “**Registered broker 0 at path /brokers/ids/0**” which indicates that the Kafka Broker is up and running. Do not close this command prompt window.

To verify the Kafka has been started successfully, use the **jps** command as below:

```
jps
```

It should display two demons **QuorumPeerMain** and **Kafka** process which denotes **Zookeeper**, and single **Kafka Broker** services are running.

```
Command Prompt
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>jps
12196 Jps
17052 Java
5180 QuorumPeerMain
6220 Kafka

C:\Users\hp>
```

4.3. Create Topic:

Kafka provides a command line utility `kafka-topics.bat` to manage topics (*create, list, describe, delete, etc.*) on the cluster. This utility requires `--bootstrap-server` argument which takes Kafka broker instance details in `hostname:port` format.

By default, a topic is created with 1 replication factor and 1 partition unless explicitly specified. Note that you cannot specify more than 1 replication factor when a single broker is running in the Kafka cluster (*i.e. maximum replication factor value = number of Kafka brokers*).

Open **Windows PowerShell** or **Command Prompt** and run the below command to create a Kafka topic named `test-topic`:

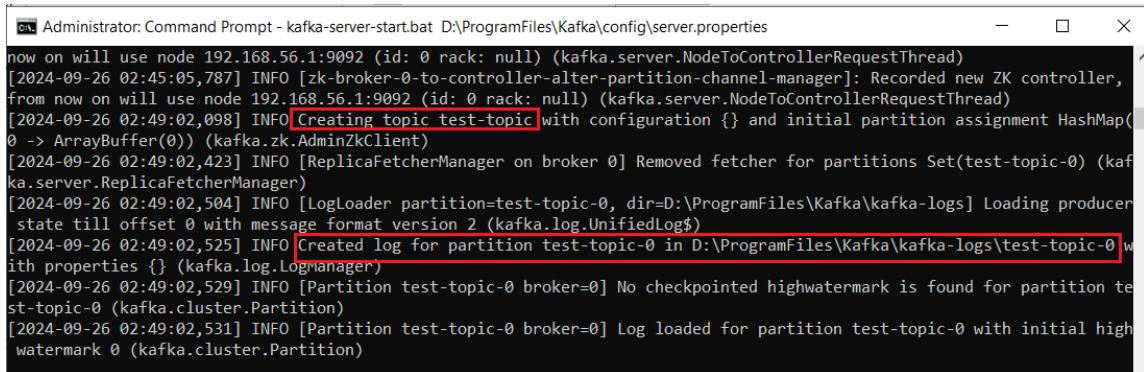
```
kafka-topics.bat --bootstrap-server localhost:9092 --create --topic test-topic
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\hp> kafka-topics.bat --bootstrap-server localhost:9092 --create --topic test-topic
Created topic test-topic.
PS C:\Users\hp>
```

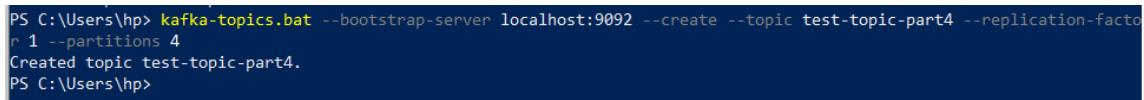
Once the topic has been created, you will see a message on the Kafka broker terminal window and the log location for the created topic `test-topic`.



```
Administrator: Command Prompt - kafka-server-start.bat D:\ProgramFiles\Kafka\config\server.properties
now on will use node 192.168.56.1:9092 (id: 0 rack: null) (kafka.server.NodeToControllerRequestThread)
[2024-09-26 02:45:05,787] INFO [zk-broker-0-to-controller-alter-partition-channel-manager]: Recorded new ZK controller, from now on will use node 192.168.56.1:9092 (id: 0 rack: null) (kafka.server.NodeToControllerRequestThread)
[2024-09-26 02:49:02,098] INFO [Creating topic test-topic] with configuration {} and initial partition assignment HashMap(0 -> ArrayBuffer(0)) (kafka.zk.AdminZkClient)
[2024-09-26 02:49:02,423] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions Set(test-topic-0) (kafka.server.ReplicaFetcherManager)
[2024-09-26 02:49:02,504] INFO [LogLoader partition=test-topic-0, dir=D:\ProgramFiles\Kafka\kafka-logs] Loading producer state till offset 0 with message format version 2 (kafka.log.UnifiedLog$)
[2024-09-26 02:49:02,525] INFO [Created log for partition test-topic-0 in D:\ProgramFiles\Kafka\kafka-logs\test-topic-0 with properties {} (kafka.log.LogManager)]
[2024-09-26 02:49:02,529] INFO [Partition test-topic-0 broker=0] No checkpointed highwatermark is found for partition test-topic-0 (kafka.cluster.Partition)
[2024-09-26 02:49:02,531] INFO [Partition test-topic-0 broker=0] Log loaded for partition test-topic-0 with initial highwatermark 0 (kafka.cluster.Partition)
```

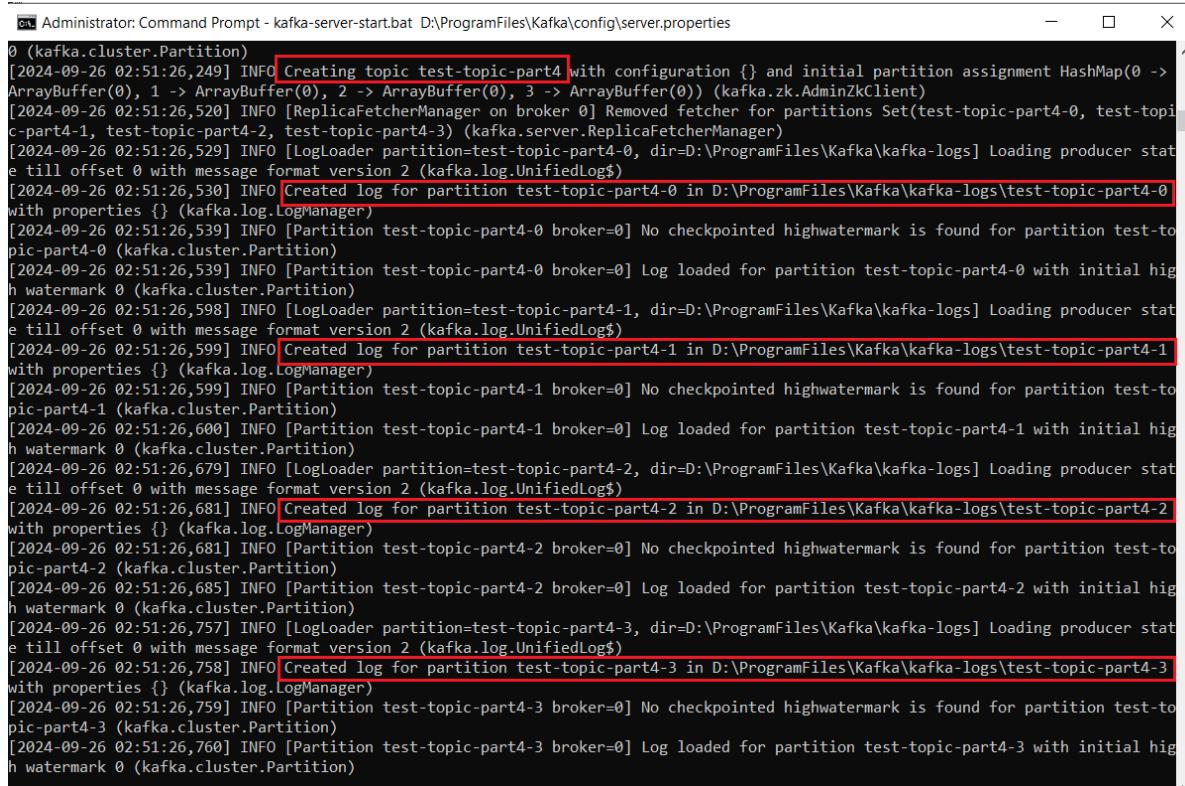
Let us run the below command to create another Kafka topic named `test-topic-part4` with 4 partitions:

```
kafka-topics.bat --bootstrap-server localhost:9092 --create --topic test-topic-part4 --replication-factor 1 --partitions 4
```



```
PS C:\Users\hp> kafka-topics.bat --bootstrap-server localhost:9092 --create --topic test-topic-part4 --replication-factor 1 --partitions 4
Created topic test-topic-part4.
PS C:\Users\hp>
```

Once the topic has been created, you will see a message on the Kafka broker terminal window and the log location for each partition of the created topic `test-topic-part4`.



```
Administrator: Command Prompt - kafka-server-start.bat D:\ProgramFiles\Kafka\config\server.properties
0 (kafka.cluster.Partition)
[2024-09-26 02:51:26,249] INFO [Creating topic test-topic-part4] with configuration {} and initial partition assignment HashMap(0 -> ArrayBuffer(0), 1 -> ArrayBuffer(0), 2 -> ArrayBuffer(0), 3 -> ArrayBuffer(0)) (kafka.zk.AdminZkClient)
[2024-09-26 02:51:26,520] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions Set(test-topic-part4-0, test-topic-part4-1, test-topic-part4-2, test-topic-part4-3) (kafka.server.ReplicaFetcherManager)
[2024-09-26 02:51:26,529] INFO [LogLoader partition=test-topic-part4-0, dir=D:\ProgramFiles\Kafka\kafka-logs] Loading producer state till offset 0 with message format version 2 (kafka.log.UnifiedLog$)
[2024-09-26 02:51:26,530] INFO [Created log for partition test-topic-part4-0 in D:\ProgramFiles\Kafka\kafka-logs\test-topic-part4-0 with properties {} (kafka.log.LogManager)]
[2024-09-26 02:51:26,539] INFO [Partition test-topic-part4-0 broker=0] No checkpointed highwatermark is found for partition test-topic-part4-0 (kafka.cluster.Partition)
[2024-09-26 02:51:26,539] INFO [Partition test-topic-part4-0 broker=0] Log loaded for partition test-topic-part4-0 with initial highwatermark 0 (kafka.cluster.Partition)
[2024-09-26 02:51:26,598] INFO [LogLoader partition=test-topic-part4-1, dir=D:\ProgramFiles\Kafka\kafka-logs] Loading producer state till offset 0 with message format version 2 (kafka.log.UnifiedLog$)
[2024-09-26 02:51:26,599] INFO [Created log for partition test-topic-part4-1 in D:\ProgramFiles\Kafka\kafka-logs\test-topic-part4-1 with properties {} (kafka.log.LogManager)]
[2024-09-26 02:51:26,599] INFO [Partition test-topic-part4-1 broker=0] No checkpointed highwatermark is found for partition test-topic-part4-1 (kafka.cluster.Partition)
[2024-09-26 02:51:26,600] INFO [Partition test-topic-part4-1 broker=0] Log loaded for partition test-topic-part4-1 with initial highwatermark 0 (kafka.cluster.Partition)
[2024-09-26 02:51:26,679] INFO [LogLoader partition=test-topic-part4-2, dir=D:\ProgramFiles\Kafka\kafka-logs] Loading producer state till offset 0 with message format version 2 (kafka.log.UnifiedLog$)
[2024-09-26 02:51:26,681] INFO [Created log for partition test-topic-part4-2 in D:\ProgramFiles\Kafka\kafka-logs\test-topic-part4-2 with properties {} (kafka.log.LogManager)]
[2024-09-26 02:51:26,681] INFO [Partition test-topic-part4-2 broker=0] No checkpointed highwatermark is found for partition test-topic-part4-2 (kafka.cluster.Partition)
[2024-09-26 02:51:26,685] INFO [Partition test-topic-part4-2 broker=0] Log loaded for partition test-topic-part4-2 with initial highwatermark 0 (kafka.cluster.Partition)
[2024-09-26 02:51:26,757] INFO [LogLoader partition=test-topic-part4-3, dir=D:\ProgramFiles\Kafka\kafka-logs] Loading producer state till offset 0 with message format version 2 (kafka.log.UnifiedLog$)
[2024-09-26 02:51:26,758] INFO [Created log for partition test-topic-part4-3 in D:\ProgramFiles\Kafka\kafka-logs\test-topic-part4-3 with properties {} (kafka.log.LogManager)]
[2024-09-26 02:51:26,759] INFO [Partition test-topic-part4-3 broker=0] No checkpointed highwatermark is found for partition test-topic-part4-3 (kafka.cluster.Partition)
[2024-09-26 02:51:26,760] INFO [Partition test-topic-part4-3 broker=0] Log loaded for partition test-topic-part4-3 with initial highwatermark 0 (kafka.cluster.Partition)
```

4.4. List Topics:

Use the below command to get the list of topics in Kafka server:

```
kafka-topics.bat --bootstrap-server localhost:9092 --list
```

```
PS C:\Users\hp> kafka-topics.bat --bootstrap-server localhost:9092 --list
test-topic
test-topic-part4
PS C:\Users\hp>
```

It has listed out two topics – test-topic and test-topic-part4 – that we created.

4.5. Describe Topics:

Use the below command to get more details including topic partitions and replication information of topics in Kafka server:

```
kafka-topics.bat --bootstrap-server localhost:9092 --describe
```

```
PS C:\Users\hp> kafka-topics.bat --bootstrap-server localhost:9092 --describe
[2024-09-26 02:56:02,694] WARN [AdminClient clientId=adminclient-1] The DescribeTopicPartitions API is not supported, using Metadata API to describe topics. (org.apache.kafka.clients.admin.KafkaAdminClient)
Topic: test-topic-part4 TopicId: 97bahCGXTJ-KjbzTfAxWbA PartitionCount: 4      ReplicationFactor: 1      Configs:
          Topic: test-topic-part4 Partition: 0      Leader: 0      Replicas: 0      Isr: 0      Elr: N/A      LastKnownElr: N/
A          Topic: test-topic-part4 Partition: 1      Leader: 0      Replicas: 0      Isr: 0      Elr: N/A      LastKnownElr: N/
A          Topic: test-topic-part4 Partition: 2      Leader: 0      Replicas: 0      Isr: 0      Elr: N/A      LastKnownElr: N/
A          Topic: test-topic-part4 Partition: 3      Leader: 0      Replicas: 0      Isr: 0      Elr: N/A      LastKnownElr: N/
A
Topic: test-topic      TopicId: 7Ecgc0NHRkC8IWynt9PKBg PartitionCount: 1      ReplicationFactor: 1      Configs:
          Topic: test-topic      Partition: 0      Leader: 0      Replicas: 0      Isr: 0      Elr: N/A      LastKnownElr: N/
A
PS C:\Users\hp>
```

It provided us the complete details of two topics – test-topic and test-topic-part4 – that we created.

4.6. Run Kafka Producer:

Kafka distribution provides a command line utility named `kafa-console-producer.bat` to publish messages into a Kafka topic.

Open **Command Prompt or Windows PowerShell** and run the following command to start producing messages:

```
kafka-console-producer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka
```

```
PS C:\Users\hp> kafka-console-producer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka
>
```

The producer will wait on input from user and publishes to Hello-Kafka topic in the Kafka cluster. Note that this topic is not available in Kafka cluster but it will be created when we start producing messages.

Now you can type a few lines of messages in the terminal:

```
PS C:\Users\hp> kafka-console-producer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka
>My first message
[2024-09-26 02:57:47,478] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 6
: {Hello-Kafka=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2024-09-26 02:57:47,602] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 7
: {Hello-Kafka=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
>My second message
>This is third message
>
```

By default, every new line entered on the producer console is published as a new message to Kafka and this configuration can be changed in

%KAFKA_HOME%\config\producer.properties file.

4.7. Run Kafka Consumer:

Kafka distribution provides a command line utility named kafka-console-consumer.bat to subscribe messages from a Kafka topic.

Open a new **Command Prompt** or **Windows PowerShell** and run the following command to start consuming messages:

```
kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka --from-beginning
```

```
ca Command Prompt - kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka --from-beginning
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka --from-beginning
My first message
My second message
This is third message
```

Here, you will see all messages that are produced to Hello-Kafka topic until now and will keep getting the new messages that will be produced to this topic.

Go to the producer window and enter a new message:

```
PS C:\Users\hp> kafka-console-producer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka
>My first message
[2024-09-26 02:57:47,478] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 6
: {Hello-Kafka=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2024-09-26 02:57:47,602] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 7
: {Hello-Kafka=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
>My second message
>This is third message
>Fourth message is here
>
```

Now, go to the consumer window where you can see the fourth message that was produced:

```
cmd Command Prompt - kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka --from-beginning
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka --from-beginning
My first message
My second message
This is third message
Fourth message is here
```

Use **Ctrl + C** to exit out of Producer and Consumer windows:

```
PS C:\Users\hp> kafka-console-producer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka
>My first message
[2024-09-26 02:57:47,478] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 6
: {Hello-Kafka=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2024-09-26 02:57:47,602] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 7
: {Hello-Kafka=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
>My second message
>This is third message
>Fourth message is here
>Terminate batch job (Y/N)? y
PS C:\Users\hp>
```

```
cmd Command Prompt
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Hello-Kafka --from-beginning
My first message
My second message
This is third message
Fourth message is here
Processed a total of 4 messages
Terminate batch job (Y/N)? y

C:\Users\hp>
```

4.8. Delete Topic:

Use the following command to delete the topic named `test-topic` in Kafka cluster.

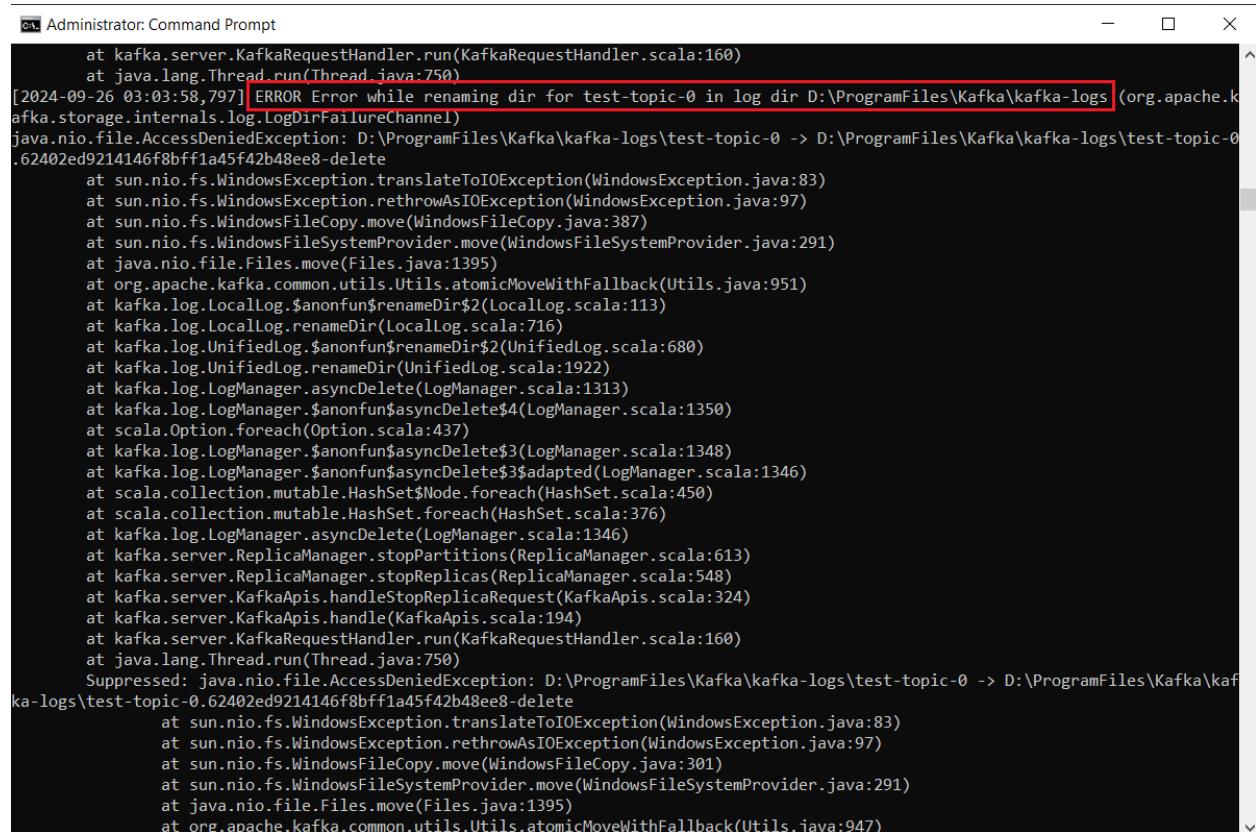
```
kafka-topics.bat --bootstrap-server localhost:9092 --delete --topic test-topic
```

```
PS C:\Users\hp> kafka-topics.bat --bootstrap-server localhost:9092 --delete --topic test-topic
PS C:\Users\hp>
```

Note:

After issuing --delete --topic command for Kafka running in Windows, it will only mark the respective topic name for deletion but will not physically delete the topic folder present in %KAFKA_HOME%\kafka-logs folder.

On the Kafka broker terminal window, you will see an *Error while renaming dir for test-topic-0 in log dir D:\ProgramFiles\Kafka\kafka-logs* and then Kafka server was shut down.



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt". The window contains a stack trace of Java code. A specific line of text is highlighted with a red rectangle: "[2024-09-26 03:03:58,797] ERROR Error while renaming dir for test-topic-0 in log dir D:\ProgramFiles\Kafka\kafka-logs (org.apache.kafka.storage.internals.log.LogDirFailureChannel)". This line indicates that Kafka is attempting to rename a directory but failing due to an error. The rest of the stack trace shows the internal logic of Kafka's log management system, including calls to WindowsException.translateToIOException, sun.nio.fs.WindowsFileCopy.move, and org.apache.kafka.common.utils.Utils.atomicMoveWithFallback.

```

, __consumer_offsets-0, __consumer_offsets-44, __consumer_offsets-39, __consumer_offsets-12, __consumer_offsets-45, __consumer_offsets-1, __consumer_offsets-5, __consumer_offsets-26, __consumer_offsets-29, __consumer_offsets-34, __consumer_offsets-10, __consumer_offsets-32, __consumer_offsets-40) (kafka.server.ReplicaLogDirsManager)
[2024-09-26 03:03:58,934] WARN [ReplicaManager broker=0] Broker 0 stopped fetcher for partitions __consumer_offsets-22,__consumer_offsets-25,__consumer_offsets-35,test-topic-part4-2,__consumer_offsets-37>Hello-Kafka-0,__consumer_offsets-13,__consumer_offsets-30,test-topic-part4-3,__consumer_offsets-8,__consumer_offsets-21,__consumer_offsets-4,__consumer_offsets-27,__consumer_offsets-7,__consumer_offsets-9,__consumer_offsets-46,__consumer_offsets-41,__consumer_offsets-33,__consumer_offsets-23,__consumer_offsets-49,__consumer_offsets-47,__consumer_offsets-16,__consumer_offsets-28,__consumer_offsets-31,__consumer_offsets-36,__consumer_offsets-42,__consumer_offsets-17,test-topic-part4-0,__consumer_offsets-48,__consumer_offsets-19,__consumer_offsets-11,__consumer_offsets-2,__consumer_offsets-43,__consumer_offsets-6,__consumer_offsets-14,__consumer_offsets-20,__consumer_offsets-0,__consumer_offsets-44,__consumer_offsets-39,__consumer_offsets-12,__consumer_offsets-45,__consumer_offsets-1,__consumer_offsets-5,__consumer_offsets-26,__consumer_offsets-29,__consumer_offsets-34,__consumer_offsets-10,__consumer_offsets-40 and stopped moving logs for partitions because they are in the failed log directory D:\ProgramFiles\Kafka\kafka-logs. (kafka.server.ReplicaManager)
[2024-09-26 03:03:58,936] WARN Stopping serving logs in dir D:\ProgramFiles\Kafka\kafka-logs (kafka.log.LogManager)
[2024-09-26 03:03:58,946] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions Set(test-topic-0) (kafka.server.ReplicaFetcherManager)
[2024-09-26 03:03:58,947] INFO [ReplicaAlterLogDirsManager on broker 0] Removed fetcher for partitions Set(test-topic-0) (kafka.server.ReplicaAlterLogDirsManager)
[2024-09-26 03:03:58,951] ERROR Shutdown broker because all log dirs in D:\ProgramFiles\Kafka\kafka-logs have failed (kafka.log.LogManager)

C:\Windows\System32>

```

Note: The above issue occurs in Windows operating system since Kafka is intended to run on Linux operating system. If we try to run Kafka in Windows natively, it may arise several issues over a period of time.

Therefore, it is recommended to run Apache Kafka on Windows through:

- **WSL2 or Docker** if using **Windows 10** or above.
- **Docker** if using **Windows 8** or below.

It is not recommended to run Kafka on the JVM on Windows, because it lacks some of the Linux specific features (for ex: POSIX). We will also run into issues at some point if we try to run Kafka on Windows without WSL2.

To resolve the above error, follow the below steps to manually delete the respective topic folder

- Open new **Command Prompt** as **Administrator** and stop the zookeeper with this command:

```

zookeeper-server-stop.bat
%KAFKA_HOME%\config\zookeeper.properties

```

```

Administrator: Command Prompt
Microsoft Windows [Version 10.0.19045.4780]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>zookeeper-server-stop.bat %KAFKA_HOME%\config\zookeeper.properties
Deleting instance \\\DESKTOP-KGH2E2G\ROOT\CIMV2:Win32_Process.Handle="2388"
Instance deletion successful.

C:\Windows\system32>

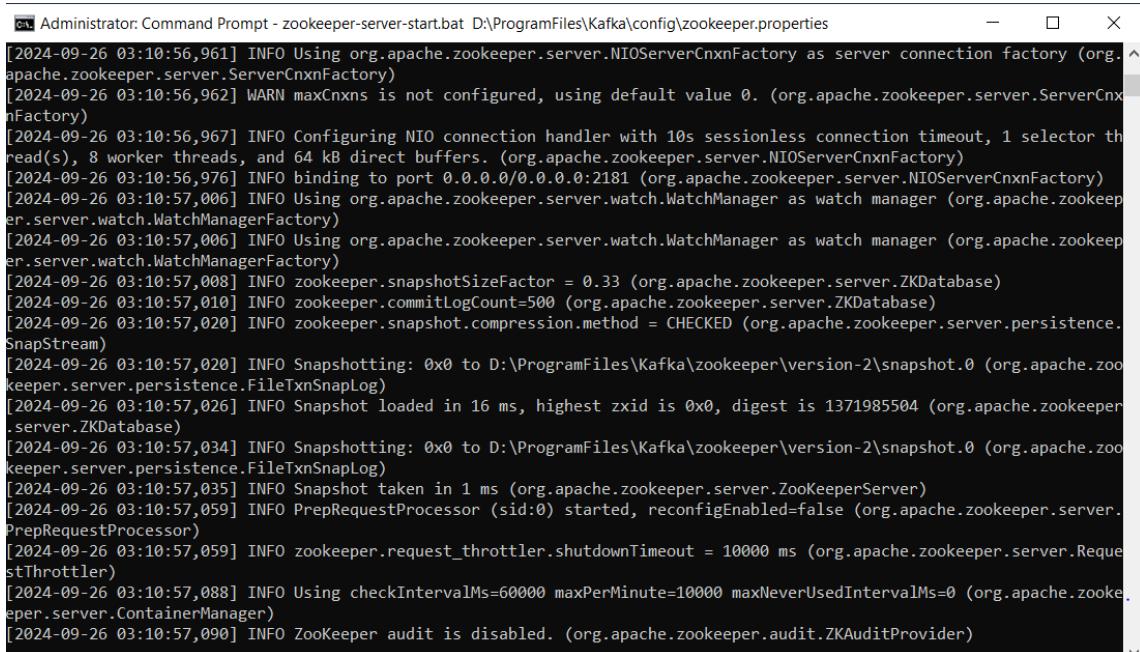
```

- Go to %KAFKA_HOME% location and delete kafka-logs and zookeeper folders

Name	Date modified	Type	Size
bin	7/23/2024 1:39 PM	File folder	
config	7/23/2024 1:39 PM	File folder	
kafka_2.13-3.8.0	Date created: 9/26/2024 Size: 514 KB	File folder	
libs	7/23/2024 1:39 PM	File folder	
licenses	7/23/2024 1:39 PM	File folder	
logs	9/26/2024 3:03 AM	File folder	
site-docs	7/23/2024 1:39 PM	File folder	
kafka_2.13-3.8.0.tgz	9/26/2024 2:31 AM	WinRAR archive	117,906 KB
LICENSE	7/23/2024 1:34 PM	File	15 KB
NOTICE	7/23/2024 1:34 PM	File	28 KB

- Open new **Command Prompt** as **Administrator** and start zookeeper with this command:

```
zookeeper-server-start.bat
%KAFKA_HOME%\config\zookeeper.properties
```



```
[2024-09-26 03:10:56,961] INFO Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory (org.apache.zookeeper.server.ServerCnxnFactory)
[2024-09-26 03:10:56,962] WARN maxCnxns is not configured, using default value 0. (org.apache.zookeeper.server.ServerCnxnFactory)
[2024-09-26 03:10:56,967] INFO Configuring NIO connection handler with 10s sessionless connection timeout, 1 selector thread(s), 8 worker threads, and 64 kB direct buffers. (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-09-26 03:10:56,976] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-09-26 03:10:57,006] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)
[2024-09-26 03:10:57,006] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)
[2024-09-26 03:10:57,008] INFO zookeeper.snapshotSizeFactor = 0.33 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 03:10:57,010] INFO zookeeper.commitLogCount=500 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 03:10:57,020] INFO zookeeper.snapshot.compression.method = CHECKED (org.apache.zookeeper.server.persistence.SnapStream)
[2024-09-26 03:10:57,020] INFO Snapshotting: 0x0 to D:\ProgramFiles\Kafka\zookeeper\version-2\snapshot.0 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-09-26 03:10:57,026] INFO Snapshot loaded in 16 ms, highest zxid is 0x0, digest is 1371985504 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 03:10:57,034] INFO Snapshotting: 0x0 to D:\ProgramFiles\Kafka\zookeeper\version-2\snapshot.0 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-09-26 03:10:57,035] INFO Snapshot taken in 1 ms (org.apache.zookeeper.server.ZooKeeperServer)
[2024-09-26 03:10:57,059] INFO PrepRequestProcessor (sid:0) started, reconfigEnabled=false (org.apache.zookeeper.server.PrePRequestProcessor)
[2024-09-26 03:10:57,059] INFO zookeeper.request_throttler.shutdownTimeout = 10000 ms (org.apache.zookeeper.server.RequestThrottler)
[2024-09-26 03:10:57,088] INFO Using checkIntervalMs=60000 maxPerMinute=10000 maxNeverUsedIntervalMs=0 (org.apache.zookeeper.server.ContainerManager)
[2024-09-26 03:10:57,090] INFO ZooKeeper audit is disabled. (org.apache.zookeeper.audit.ZKAuditProvider)
```

- Open new **Command Prompt** as **Administrator** and start Kafka broker with this command:

```
kafka-server-start.bat %KAFKA_HOME%\config\server.properties
```

```

Administrator: Command Prompt - kafka-server-start.bat D:\ProgramFiles\Kafka\config\server.properties
[2024-09-26 03:11:55,785] INFO Feature ZK node created at path: /feature (kafka.server.FinalizedFeatureChangeListener)
[2024-09-26 03:11:55,788] INFO [TxnMarkerSenderThread-0]: Starting (kafka.coordinator.transaction.TransactionMarkerChann
elManager)
[2024-09-26 03:11:55,788] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.Transaction
Coordinator)
[2024-09-26 03:11:55,836] INFO [MetadataCache brokerId=0] Updated cache from existing None to latest Features(metadataVe
rsion=3.8-IV0, finalizedFeatures={}, finalizedFeaturesEpoch=0). (kafka.server.metadata.ZkMetadataCache)
[2024-09-26 03:11:55,947] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredO
perationReaper)
[2024-09-26 03:11:56,025] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Enabling request processing. (kafka.netw
ork.SocketServer)
[2024-09-26 03:11:56,026] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationLi
stener$ChangeEventProcessThread)
[2024-09-26 03:11:56,030] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.DataPlaneAcceptor)
[2024-09-26 03:11:56,042] INFO [KafkaServer id=0] Start processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:11:56,044] INFO [KafkaServer id=0] End processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:11:56,045] INFO [KafkaServer id=0] Start processing enable request processing future (kafka.server.KafkaS
erver)
[2024-09-26 03:11:56,048] INFO [KafkaServer id=0] End processing enable request processing future (kafka.server.KafkaSer
ver)
[2024-09-26 03:11:56,051] INFO Kafka version: 3.8.0 (org.apache.kafka.common.utils.AppInfoParser)
[2024-09-26 03:11:56,056] INFO Kafka commitId: 771b9576b00ecf5b (org.apache.kafka.common.utils.AppInfoParser)
[2024-09-26 03:11:56,058] INFO Kafka startTimeMs: 1727300516048 (org.apache.kafka.common.utils.AppInfoParser)
[2024-09-26 03:11:56,061] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
[2024-09-26 03:11:56,587] INFO [zk-broker-0-to-controller-alter-partition-channel-manager]: Recorded new ZK controller,
from now on will use node 192.168.56.1:9092 (id: 0 rack: null) (kafka.server.NodeToControllerRequestThread)
[2024-09-26 03:11:56,635] INFO [zk-broker-0-to-controller-forwarding-channel-manager]: Recorded new ZK controller, from
now on will use node 192.168.56.1:9092 (id: 0 rack: null) (kafka.server.NodeToControllerRequestThread)

```

- Go to %KAFKA_HOME% location and you can see kafka-logs and zookeeper folders were created freshly.

Name	Date modified	Type	Size
bin	7/23/2024 1:39 PM	File folder	
config	7/23/2024 1:39 PM	File folder	
kafka_2.13-3.8.0	9/26/2024 2:33 AM	File folder	
kafka-logs	9/26/2024 3:12 AM	File folder	
libs	7/23/2024 1:39 PM	File folder	
licenses	7/23/2024 1:39 PM	File folder	
logs	9/26/2024 3:03 AM	File folder	
site-docs	7/23/2024 1:39 PM	File folder	
zookeeper	9/26/2024 3:10 AM	File folder	
kafka_2.13-3.8.0.tgz	9/26/2024 2:31 AM	WinRAR archive	117,906 KB
LICENSE	7/23/2024 1:34 PM	File	15 KB
NOTICE	7/23/2024 1:34 PM	File	28 KB

Note that we have deleted all topics and will have to start creating topic freshly.

5. Configure Single Node-Multiple Brokers:

In this configuration, we will have single Zookeeper and multiple broker instances running on a single machine.

5.1. Setup Multiple Brokers:

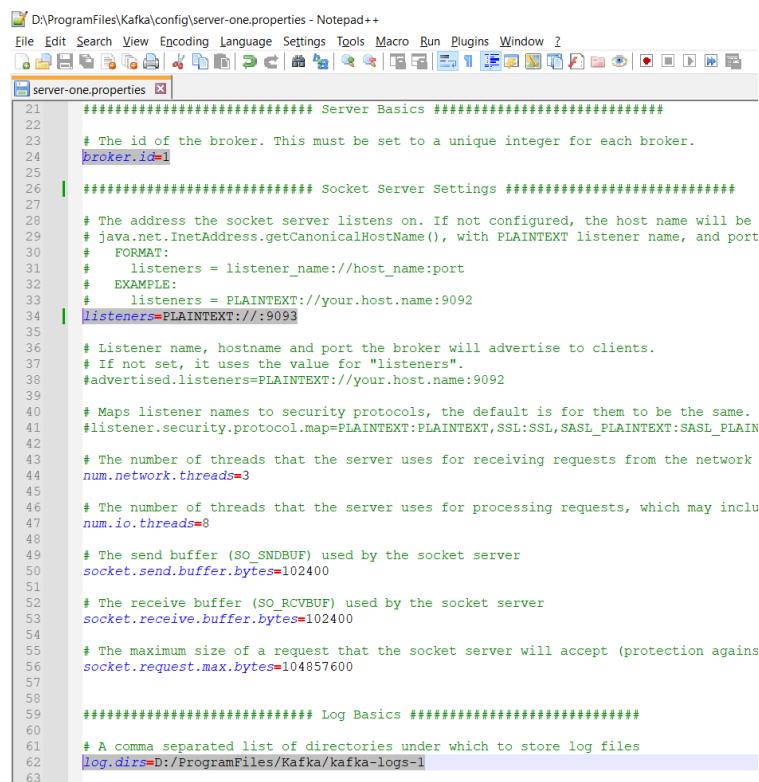
In the previous configuration, one Kafka broker instance was already configured in `server.properties` file.

To setup multiple broker instances, take two copies of `server.properties` file and rename as `server-one.properties` and `server-two.properties` in `%KAFKA_HOME%\config` location.

Name	Date modified	Type	Size
server.properties	9/26/2024 2:38 AM	Properties Source ...	7 KB
server-one.properties	9/26/2024 2:38 AM	Properties Source ...	7 KB
server-two.properties	9/26/2024 2:38 AM	Properties Source ...	7 KB
zookeeper.properties	9/26/2024 2:35 AM	Properties Source ...	2 KB
connect-console-sink.properties	7/23/2024 1:34 PM	Properties Source ...	1 KB
connect-console-source.properties	7/23/2024 1:34 PM	Properties Source ...	1 KB

Open `server-one.properties` and change the following settings:

```
broker.id=1
listeners=PLAINTEXT://:9093
log.dirs=D:/ProgramFiles/Kafka/kafka-logs-1
```

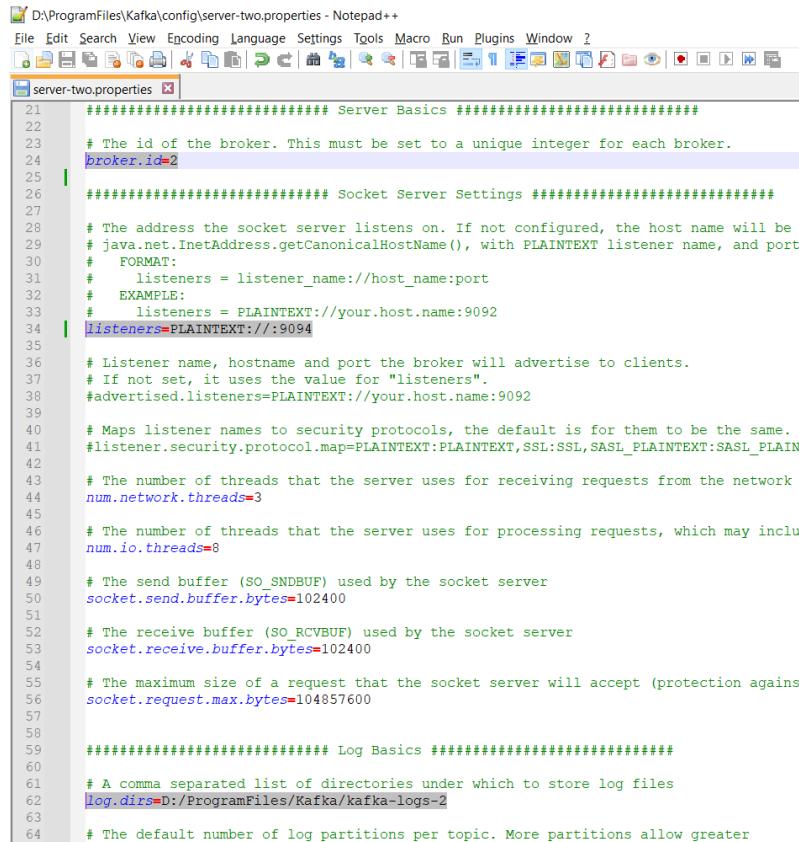


The screenshot shows the Notepad++ application window with the file `server-one.properties` open. The code editor displays the configuration settings for a Kafka broker. The `broker.id` is set to 1, and the `listeners` is set to `PLAINTEXT://:9093`. The `log.dirs` is set to `D:/ProgramFiles/Kafka/kafka-logs-1`. The code is color-coded, with comments in green and other text in black. The Notepad++ interface includes a toolbar at the top and a status bar at the bottom.

```
21 ##### Server Basics #####
22
23 # The id of the broker. This must be set to a unique integer for each broker.
24 broker.id=1
25
26 ##### Socket Server Settings #####
27
28 # The address the socket server listens on. If not configured, the host name will be
29 # java.net.InetAddress.getCanonicalHostName(), with PLAINTEXT listener name, and port
30 # FORMAT:
31 # listeners = listener_name://host_name:port
32 # EXAMPLE:
33 # listeners = PLAINTEXT://your.host.name:9092
34 listeners=PLAINTEXT://:9093
35
36 # Listener name, hostname and port the broker will advertise to clients.
37 # If not set, it uses the value for "listeners".
38 #advertised.listeners=PLAINTEXT://your.host.name:9092
39
40 # Maps listener names to security protocols, the default is for them to be the same.
41 #listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAIN_PLAIN
42
43 # The number of threads that the server uses for receiving requests from the network
44 num.network.threads=3
45
46 # The number of threads that the server uses for processing requests, which may include
47 num.io.threads=8
48
49 # The send buffer (SO_SNDBUF) used by the socket server
50 socket.send.buffer.bytes=102400
51
52 # The receive buffer (SO_RCVBUF) used by the socket server
53 socket.receive.buffer.bytes=102400
54
55 # The maximum size of a request that the socket server will accept (protection against
56 # socket.request.max.bytes=104857600
57
58 ##### Log Basics #####
59
60 # A comma separated list of directories under which to store log files
61 log.dirs=D:/ProgramFiles/Kafka/kafka-logs-1
62
```

Open `server-two.properties` and change the following settings:

```
broker.id=2
listeners=PLAINTEXT://:9094
log.dirs=D:/ProgramFiles/Kafka/kafka-logs-2
```



The screenshot shows the Notepad++ application window with the file "server-two.properties" open. The code in the editor is as follows:

```
21 ##### Server Basics #####
22
23 # The id of the broker. This must be set to a unique integer for each broker.
24 broker.id=2
25
26 ##### Socket Server Settings #####
27
28 # The address the socket server listens on. If not configured, the host name will be
29 # java.net.InetAddress.getCanonicalHostName(), with PLAINTEXT listener name, and port
30 # FORMAT:
31 #   listeners = listener_name://host_name:port
32 # EXAMPLE:
33 #   listeners = PLAINTEXT://your.host.name:9092
34 listeners=PLAINTEXT://:9094
35
36 # Listener name, hostname and port the broker will advertise to clients.
37 # If not set, it uses the value for "listeners".
38 #advertised.listeners=PLAINTEXT://your.host.name:9092
39
40 # Maps listener names to security protocols, the default is for them to be the same.
41 #listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAIN
42
43 # The number of threads that the server uses for receiving requests from the network
44 num.network.threads=3
45
46 # The number of threads that the server uses for processing requests, which may inclu
47 num.io.threads=8
48
49 # The send buffer (SO_SNDBUF) used by the socket server
50 socket.send.buffer.bytes=102400
51
52 # The receive buffer (SO_RCVBUF) used by the socket server
53 socket.receive.buffer.bytes=102400
54
55 # The maximum size of a request that the socket server will accept (protection agains
56 socket.request.max.bytes=104857600
57
58
59 ##### Log Basics #####
60
61 # A comma separated list of directories under which to store log files
62 log.dirs=D:/ProgramFiles/Kafka/kafka-logs-2
63
64 # The default number of log partitions per topic. More partitions allow greater
```

Note that kafka-logs-1 and kafka-logs-2 folders will be created automatically in %KAFKA_HOME% when Kafka brokers are started.

5.2. Start Kafka Servers:

Now, we will start the Zookeeper and multiple Kafka Broker instances in the local system.

5.2.1. Start Zookeeper:

Open **Command Prompt as Administrator** and start Zookeeper using the below command (*stop it if it is already running*)

```
zookeeper-server-stop.bat %KAFKA_HOME%\config\zookeeper.properties
```

```
zookeeper-server-start.bat %KAFKA_HOME%\config\zookeeper.properties
```

```
Administrator: Command Prompt - zookeeper-server-start.bat D:\ProgramFiles\Kafka\config\zookeeper.properties
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>zookeeper-server-start.bat %KAFKA_HOME%\config\zookeeper.properties
[2024-09-26 03:19:42,834] INFO Reading configuration from: D:\ProgramFiles\Kafka\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,844] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,844] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,845] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,845] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,847] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-09-26 03:19:42,848] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-09-26 03:19:42,848] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-09-26 03:19:42,848] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2024-09-26 03:19:42,850] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2024-09-26 03:19:42,850] INFO Reading configuration from: D:\ProgramFiles\Kafka\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,852] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,853] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,853] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,853] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-09-26 03:19:42,853] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2024-09-26 03:19:42,875] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@366e2eef (org.apache.zookeeper.server.ServerMetrics)
[2024-09-26 03:19:42,882] INFO ACL digest algorithm is: SHA1 (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
[2024-09-26 03:19:42,883] INFO zookeeper.DigestAuthenticationProvider.enabled = true (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
[2024-09-26 03:19:42,883] INFO read(s), 8 worker threads, and 64 kB direct buffers. (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-09-26 03:19:42,999] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-09-26 03:19:43,035] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)
[2024-09-26 03:19:43,035] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)
[2024-09-26 03:19:43,037] INFO zookeeper.snapshotSizeFactor = 0.33 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 03:19:43,040] INFO zookeeper.commitLogCount=500 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 03:19:43,051] INFO zookeeper.snapshot.compression.method = CHECKED (org.apache.zookeeper.server.persistence.SnapStream)
[2024-09-26 03:19:43,065] INFO Reading snapshot D:\ProgramFiles\Kafka\zookeeper\version-2\snapshot.0 (org.apache.zookeeper.server.persistence.FileSnap)
[2024-09-26 03:19:43,075] INFO The digest value is empty in snapshot (org.apache.zookeeper.server.DataTree)
[2024-09-26 03:19:43,141] INFO 29 txns loaded in 34 ms (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-09-26 03:19:43,141] INFO Snapshot loaded in 100 ms, highest zxid is 0x1d, digest is 61997912927 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 03:19:43,146] INFO Snapshotting: 0x1d to D:\ProgramFiles\Kafka\zookeeper\version-2\snapshot.1d (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-09-26 03:19:43,151] INFO Snapshot taken in 5 ms (org.apache.zookeeper.server.ZooKeeperServer)
[2024-09-26 03:19:43,174] INFO PrepRequestProcessor (sid:0) started, reconfigEnabled=false (org.apache.zookeeper.server.PrepRequestProcessor)
[2024-09-26 03:19:43,174] INFO zookeeper.request_throttler.shutdownTimeout = 10000 ms (org.apache.zookeeper.server.RequestThrottler)
[2024-09-26 03:19:43,210] INFO Using checkIntervalMs=60000 maxPerMinute=10000 maxNeverUsedIntervalMs=0 (org.apache.zookeeper.server.ContainerManager)
[2024-09-26 03:19:43,212] INFO ZooKeeper audit is disabled. (org.apache.zookeeper.audit.ZKAuditProvider)
[2024-09-26 03:20:01,523] INFO Expiring session 0x1000d60530a0000, timeout of 18000ms exceeded (org.apache.zookeeper.server.ZooKeeperServer)
[2024-09-26 03:20:01,540] INFO Creating new log file: log.1e (org.apache.zookeeper.server.persistence.FileTxnLog)
```

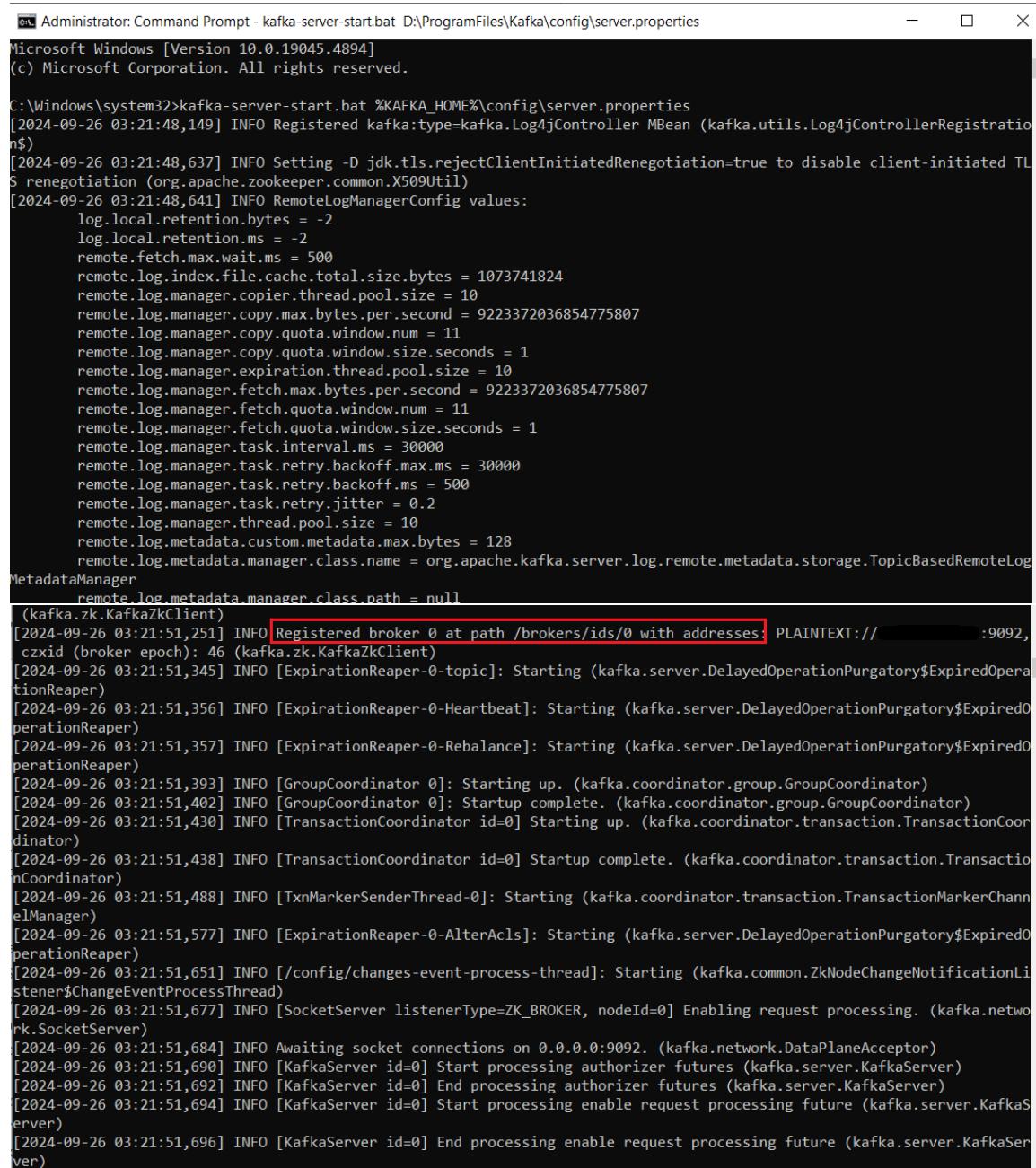
On the console, you will see a message “**binding to port 0.0.0.0/0.0.0.0:2181**” which indicates that the Zookeeper is up and running. Do not close this command prompt window.

5.2.2. Start Kafka Brokers:

Open another **Command Prompt** as **Administrator** and start the first Kafka broker instance (stop it if it is already running) using the below command:

```
kafka-server-stop.bat %KAFKA_HOME%\config\server.properties
```

```
kafka-server-start.bat %KAFKA_HOME%\config\server.properties
```



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt - kafka-server-start.bat D:\ProgramFiles\Kafka\config\server.properties". The window displays the output of the kafka-server-start.bat command, which starts the Kafka broker. The logs include various INFO messages from Kafka components like Log4jController, RemoteLogManagerConfig, and TransactionCoordinator, indicating the broker is starting up and connecting to Zookeeper.

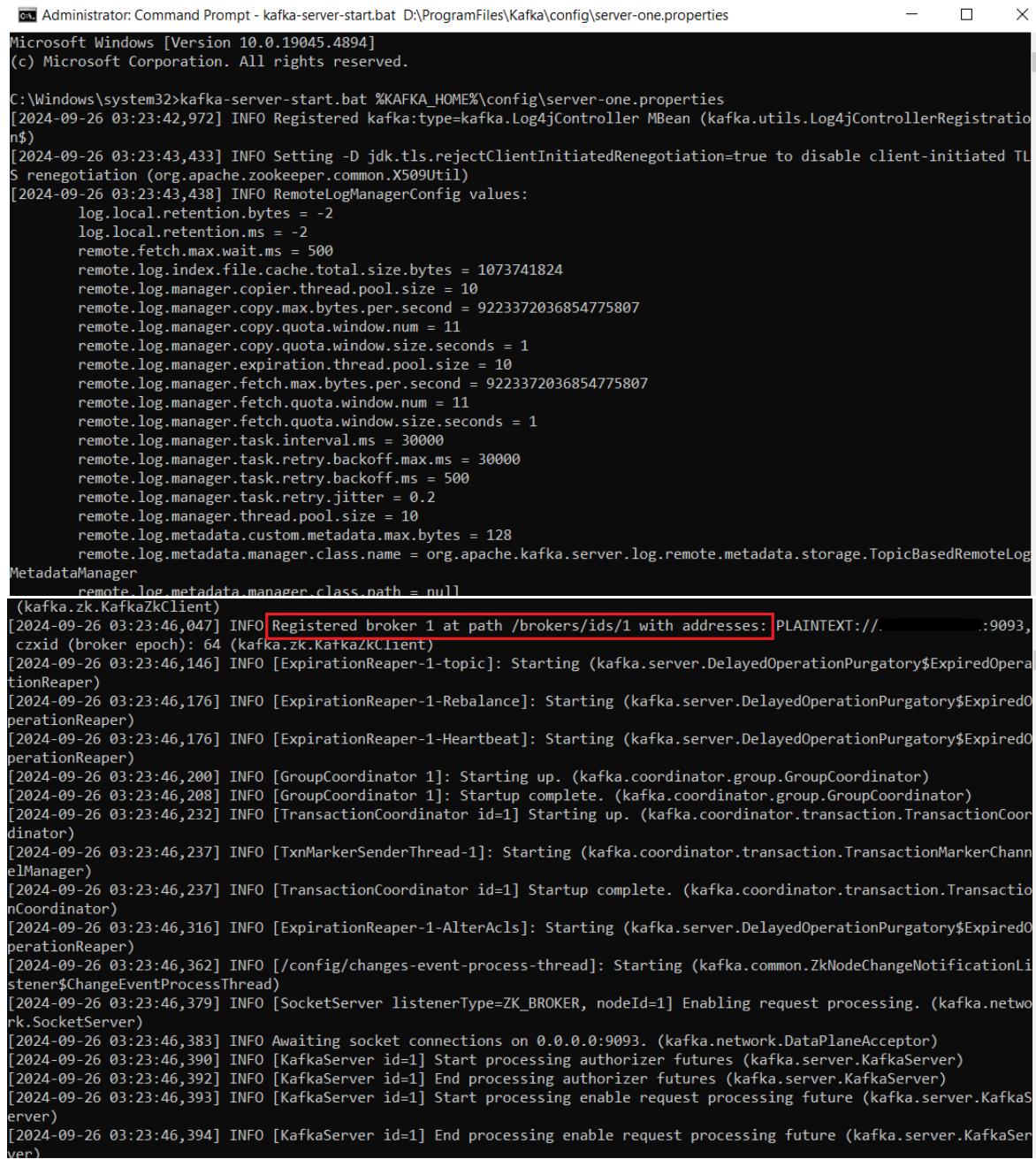
```
C:\> Administrator: Command Prompt - kafka-server-start.bat D:\ProgramFiles\Kafka\config\server.properties
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>kafka-server-start.bat %KAFKA_HOME%\config\server.properties
[2024-09-26 03:21:48,149] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2024-09-26 03:21:48,637] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2024-09-26 03:21:48,641] INFO RemoteLogManagerConfig values:
    log.local.retention.bytes = -2
    log.local.retention.ms = -2
    remote.fetch.max.wait.ms = 500
    remote.log.index.file.cache.total.size.bytes = 1073741824
    remote.log.manager.copier.thread.pool.size = 10
    remote.log.manager.copy.max.bytes.per.second = 9223372036854775807
    remote.log.manager.copy.quota.window.num = 11
    remote.log.manager.copy.quota.window.size.seconds = 1
    remote.log.manager.expiration.thread.pool.size = 10
    remote.log.manager.fetch.max.bytes.per.second = 9223372036854775807
    remote.log.manager.fetch.quota.window.num = 11
    remote.log.manager.fetch.quota.window.size.seconds = 1
    remote.log.manager.task.interval.ms = 30000
    remote.log.manager.task.retry.backoff.max.ms = 30000
    remote.log.manager.task.retry.backoff.ms = 500
    remote.log.manager.task.retry.jitter = 0.2
    remote.log.manager.thread.pool.size = 10
    remote.log.metadata.custom.metadata.max.bytes = 128
    remote.log.metadata.manager.class.name = org.apache.kafka.server.log.remote.metadata.storage.TopicBasedRemoteLogMetadataManager
        remote_log_metadata_manager.class.path = null
(kafka.zk.KafkaZkClient)
[2024-09-26 03:21:51,251] INFO Registered broker 0 at path /brokers/ids/0 with addresses: PLAINTEXT://:9092,
czxid (broker epoch): 46 (kafka.zk.KafkaZkClient)
[2024-09-26 03:21:51,345] INFO [ExpirationReaper-0-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:21:51,356] INFO [ExpirationReaper-0-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:21:51,357] INFO [ExpirationReaper-0-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:21:51,393] INFO [GroupCoordinator 0]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:21:51,402] INFO [GroupCoordinator 0]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:21:51,430] INFO [TransactionCoordinator id=0] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:21:51,438] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:21:51,488] INFO [TxnMarkerSenderThread-0]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2024-09-26 03:21:51,577] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:21:51,651] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2024-09-26 03:21:51,677] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Enabling request processing. (kafka.network.SocketServer)
[2024-09-26 03:21:51,684] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.DataPlaneAcceptor)
[2024-09-26 03:21:51,690] INFO [KafkaServer id=0] Start processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:21:51,692] INFO [KafkaServer id=0] End processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:21:51,694] INFO [KafkaServer id=0] Start processing enable request processing future (kafka.server.KafkaServer)
[2024-09-26 03:21:51,696] INFO [KafkaServer id=0] End processing enable request processing future (kafka.server.KafkaServer)
```

On the console, you will see a message “**Registered broker 0 at path /brokers/ids/0**” which indicates that the first Kafka Broker is up and running. Do not close this command prompt window.

Open another **Command Prompt as Administrator** and start the second Kafka broker instance using the below command:

```
kafka-server-start.bat %KAFKA_HOME%\config\server-one.properties
```



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt - kafka-server-start.bat D:\ProgramFiles\Kafka\config\server-one.properties". The window displays log output from the Kafka server. A red box highlights the line "[2024-09-26 03:23:46,047] INFO Registered broker 1 at path /brokers/ids/1 with addresses: PLAINTEXT://...:9093, czxid (broker epoch): 64 (kafka.zk.KafkaZkClient)". This indicates that the second Kafka broker has started successfully.

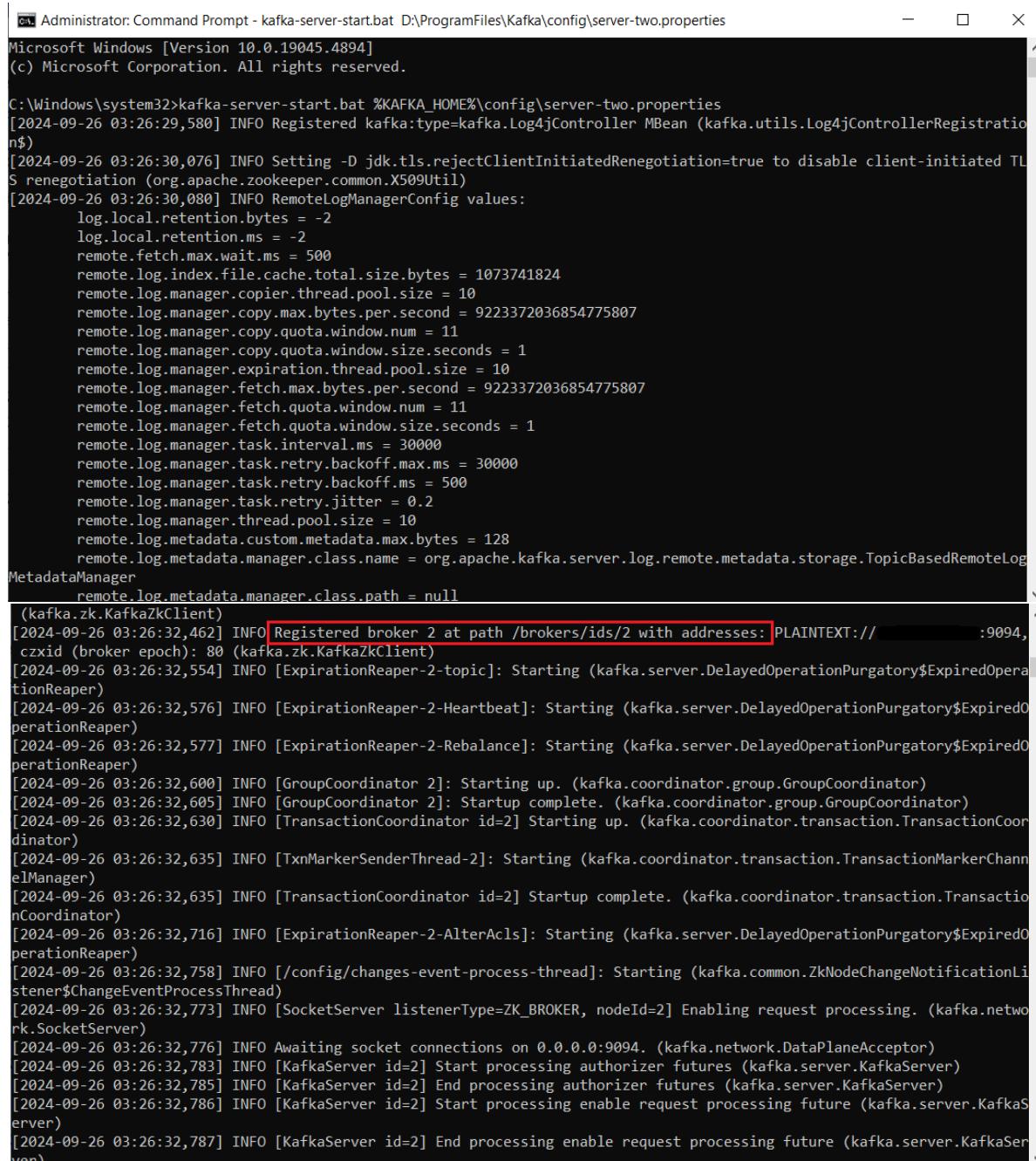
```
C:\Windows\system32>kafka-server-start.bat %KAFKA_HOME%\config\server-one.properties
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>kafka-server-start.bat %KAFKA_HOME%\config\server-one.properties
[2024-09-26 03:23:42,972] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2024-09-26 03:23:43,433] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2024-09-26 03:23:43,438] INFO RemoteLogManagerConfig values:
    log.local.retention.bytes = -2
    log.local.retention.ms = -2
    remote.fetch.max.wait.ms = 500
    remote.log.index.file.cache.total.size.bytes = 1073741824
    remote.log.manager.copier.thread.pool.size = 10
    remote.log.manager.copy.max.bytes.per.second = 9223372036854775807
    remote.log.manager.copy.quota.window.num = 11
    remote.log.manager.copy.quota.window.size.seconds = 1
    remote.log.manager.expiration.thread.pool.size = 10
    remote.log.manager.fetch.max.bytes.per.second = 9223372036854775807
    remote.log.manager.fetch.quota.window.num = 11
    remote.log.manager.fetch.quota.window.size.seconds = 1
    remote.log.manager.task.interval.ms = 30000
    remote.log.manager.task.retry.backoff.max.ms = 30000
    remote.log.manager.task.retry.backoff.ms = 500
    remote.log.manager.task.retry.jitter = 0.2
    remote.log.manager.thread.pool.size = 10
    remote.log.metadata.custom.metadata.max.bytes = 128
    remote.log.metadata.manager.class.name = org.apache.kafka.server.log.remote.metadata.storage.TopicBasedRemoteLog
MetadataManager
    remote_log_metadata_manager.class.name = null
(kafka.zk.KafkaZkClient)
[2024-09-26 03:23:46,047] INFO Registered broker 1 at path /brokers/ids/1 with addresses: PLAINTEXT://...:9093, czxid (broker epoch): 64 (kafka.zk.KafkaZkClient)
[2024-09-26 03:23:46,146] INFO [ExpirationReaper-1-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:23:46,176] INFO [ExpirationReaper-1-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:23:46,176] INFO [ExpirationReaper-1-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:23:46,200] INFO [GroupCoordinator 1]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:23:46,208] INFO [GroupCoordinator 1]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:23:46,232] INFO [TransactionCoordinator id=1] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:23:46,237] INFO [TxnMarkerSenderThread-1]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2024-09-26 03:23:46,237] INFO [TransactionCoordinator id=1] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:23:46,316] INFO [ExpirationReaper-1-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:23:46,362] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2024-09-26 03:23:46,379] INFO [SocketServer listenerType=ZK_BROKER, nodeId=1] Enabling request processing. (kafka.network.SocketServer)
[2024-09-26 03:23:46,383] INFO Awaiting socket connections on 0.0.0.0:9093. (kafka.network.DataPlaneAcceptor)
[2024-09-26 03:23:46,390] INFO [KafkaServer id=1] Start processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:23:46,392] INFO [KafkaServer id=1] End processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:23:46,393] INFO [KafkaServer id=1] Start processing enable request processing future (kafka.server.KafkaServer)
[2024-09-26 03:23:46,394] INFO [KafkaServer id=1] End processing enable request processing future (kafka.server.KafkaServer)
```

On the console, you will see a message “**Registered broker 1 at path /brokers/ids/1**” which indicates that the second Kafka Broker is up and running. Do not close this command prompt window.

Open another **Command Prompt as Administrator** and start the third Kafka broker instance using the below command:

```
kafka-server-start.bat %KAFKA_HOME%\config\server-two.properties
```



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt - kafka-server-start.bat D:\ProgramFiles\Kafka\config\server-two.properties". The window displays the log output of the Kafka server starting up. A specific line in the log is highlighted with a red box: "[2024-09-26 03:26:32,462] INFO Registered broker 2 at path /brokers/ids/2 with addresses: PLAINTEXT://:9094, czxid (broker epoch): 80 (kafka.zk.KafkaZkClient)". This indicates that the third Kafka broker has been successfully registered and is now listening on port 9094.

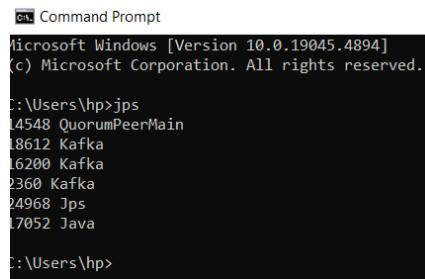
```
C:\Windows\system32>kafka-server-start.bat %KAFKA_HOME%\config\server-two.properties
[2024-09-26 03:26:29,580] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration)
[2024-09-26 03:26:30,076] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2024-09-26 03:26:30,080] INFO RemoteLogManagerConfig values:
    log.local.retention.bytes = -2
    log.local.retention.ms = -2
    remote.fetch.max.wait.ms = 500
    remote.log.index.file.cache.total.size.bytes = 1073741824
    remote.log.manager.thread.pool.size = 10
    remote.log.manager.copy.max.bytes.per.second = 9223372036854775807
    remote.log.manager.copy.quota.window.num = 11
    remote.log.manager.copy.quota.window.size.seconds = 1
    remote.log.manager.expiration.thread.pool.size = 10
    remote.log.manager.fetch.max.bytes.per.second = 9223372036854775807
    remote.log.manager.fetch.quota.window.num = 11
    remote.log.manager.fetch.quota.window.size.seconds = 1
    remote.log.manager.task.interval.ms = 30000
    remote.log.manager.task.retry.backoff.max.ms = 30000
    remote.log.manager.task.retry.backoff.ms = 500
    remote.log.manager.task.retry.jitter = 0.2
    remote.log.manager.thread.pool.size = 10
    remote.log.metadata.custom.metadata.max.bytes = 128
    remote.log.metadata.manager.class.name = org.apache.kafka.server.log.remote.metadata.storage.TopicBasedRemoteLog
MetadataManager
    remote.log.metadata.manager.class.path = null
(kafka.zk.KafkaZkClient)
[2024-09-26 03:26:32,462] INFO Registered broker 2 at path /brokers/ids/2 with addresses: PLAINTEXT://:9094, czxid (broker epoch): 80 (kafka.zk.KafkaZkClient)
[2024-09-26 03:26:32,554] INFO [ExpirationReaper-2-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:26:32,576] INFO [ExpirationReaper-2-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:26:32,577] INFO [ExpirationReaper-2-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:26:32,600] INFO [GroupCoordinator 2]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:26:32,605] INFO [GroupCoordinator 2]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:26:32,630] INFO [TransactionCoordinator id=2]: Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:26:32,635] INFO [TxnMarkerSenderThread-2]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2024-09-26 03:26:32,635] INFO [TransactionCoordinator id=2]: Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:26:32,716] INFO [ExpirationReaper-2-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:26:32,758] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2024-09-26 03:26:32,773] INFO [SocketServer listenerType=ZK_BROKER, nodeId=2]: Enabling request processing. (kafka.network.SocketServer)
[2024-09-26 03:26:32,776] INFO Awaiting socket connections on 0.0.0.0:9094. (kafka.network.DataPlaneAcceptor)
[2024-09-26 03:26:32,783] INFO [KafkaServer id=2]: Start processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:26:32,785] INFO [KafkaServer id=2]: End processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:26:32,786] INFO [KafkaServer id=2]: Start processing enable request processing future (kafka.server.KafkaServer)
[2024-09-26 03:26:32,787] INFO [KafkaServer id=2]: End processing enable request processing future (kafka.server.KafkaServer)
```

On the console, you will see a message “**Registered broker 2 at path /brokers/ids/2**” which indicates that the third Kafka Broker is up and running. Do not close this command prompt window.

Run **jps** command to verify the Kafka has been started successfully with multiple brokers

```
jps
```

It should display one **QuorumPeerMain** daemon and three **Kafka** daemons which denotes one **Zookeeper**, and three **Kafka Broker** services are running.



```
Command Prompt
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

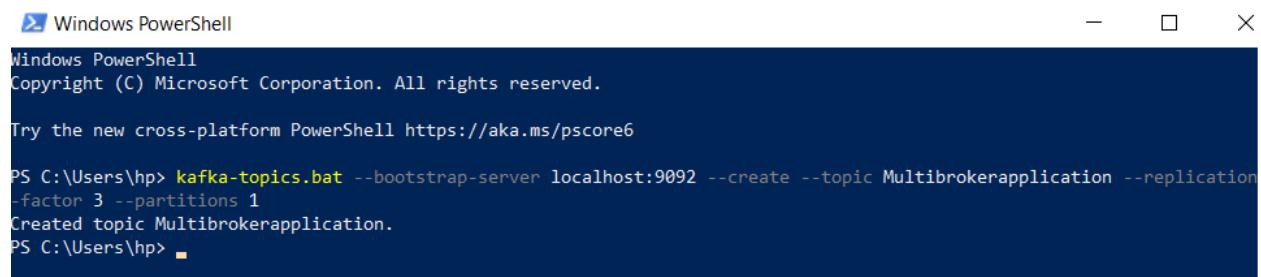
C:\Users\hp>jps
14548 QuorumPeerMain
18612 Kafka
16200 Kafka
2360 Kafka
24968 Jps
17052 Java

C:\Users\hp>
```

5.3. Create Topic:

Open **Windows PowerShell** and run the below command to create a new Kafka topic with replication factor 3 (since we have 3 broker instances running)

```
kafka-topics.bat --bootstrap-server localhost:9092 --create --topic Multibrokerapplication --replication-factor 3 --partitions 1
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\hp> kafka-topics.bat --bootstrap-server localhost:9092 --create --topic Multibrokerapplication --replication-factor 3 --partitions 1
Created topic Multibrokerapplication.
PS C:\Users\hp> ■
```

5.4. Describe Topic:

Use the below command to check which broker is running on the current created topic:

```
kafka-topics.bat --bootstrap-server localhost:9092 --describe --topic Multibrokerapplication
```

```
PS C:\Users\hp> kafka-topics.bat --bootstrap-server localhost:9092 --describe --topic Multibrokerapplication
[2024-09-26 03:30:08,687] WARN [AdminClient clientId=adminclient-1] The DescribeTopicPartitions API is not supported, using Metadata API to describe topics. (org.apache.kafka.clients.admin.KafkaAdminClient)
Topic: Multibrokerapplication TopicId: qAHJI760QAS8UNLFEMV8Q PartitionCount: 1 ReplicationFactor: 3 Configs:
          Topic: Multibrokerapplication Partition: 0 Leader: 0 Replicas: 0,2,1 Isr: 0,2,1 Elr: N/A
LastKnownElr: N/A
PS C:\Users\hp>
```

Here, we see that our first broker (**Leader 0** with `broker.id 0`) is the leader. Then **Replicas:0,2,1** means that all the brokers replicate the topic and **Isr** is the set of in-sync replicas (*a subset of replicas that are currently alive and caught up by the leader*).

5.5. Run Kafka Producer:

Open **Command Prompt or Windows PowerShell** and run the following command to start producing messages to second broker running on port 9093:

```
kafka-console-producer.bat --bootstrap-server localhost:9093 --topic Multibrokerapplication
```

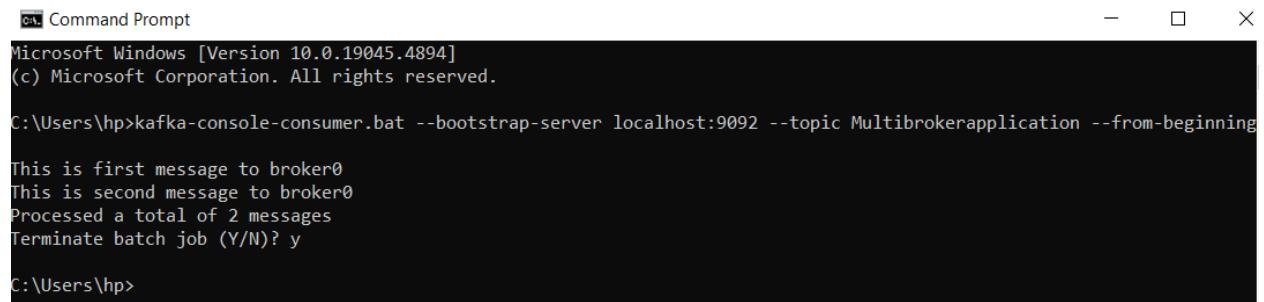
```
PS C:\Users\hp> kafka-console-producer.bat --bootstrap-server localhost:9093 --topic Multibrokerapplication
>This is first message to broker0
>This is second message to broker0
>
```

5.6. Run Kafka Consumer:

Open a new **Command Prompt or Windows PowerShell** and run the following command to start consuming messages:

Note that we can connect to any broker to see messages since we have given the replication factor as 3 while creating `Multibrokerapplication` topic.

```
kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Multibrokerapplication --from-beginning
```



A screenshot of a Microsoft Windows Command Prompt window. The title bar says "Command Prompt". The window content shows the following text:

```
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic Multibrokerapplication --from-beginning

This is first message to broker0
This is second message to broker0
Processed a total of 2 messages
Terminate batch job (Y/N)? y

C:\Users\hp>
```

```
kafka-console-consumer.bat --bootstrap-server localhost:9093 --topic Multibrokerapplication --from-beginning
```

```
C:\Users\hp>kafka-console-consumer.bat --bootstrap-server localhost:9093 --topic Multibrokerapplication --from-beginning
This is first message to broker0
This is second message to broker0
Processed a total of 2 messages
Terminate batch job (Y/N)? y
C:\Users\hp>
```

```
kafka-console-consumer.bat --bootstrap-server localhost:9094 --topic Multibrokerapplication --from-beginning
```

```
C:\Users\hp>kafka-console-consumer.bat --bootstrap-server localhost:9094 --topic Multibrokerapplication --from-beginning
This is first message to broker0
This is second message to broker0
Processed a total of 2 messages
Terminate batch job (Y/N)? y
C:\Users\hp>
```

Here, you will see all messages that are produced to **Multibrokerapplication** topic until now and will keep getting the new messages that will be produced to this topic.

Use **Ctrl + C** to exit out of Producer and Consumer windows.

5.7. Delete Topic:

Use the following command to delete the topic named `test-topic` in Kafka cluster.

```
kafka-topics.bat --bootstrap-server localhost:9094 --delete --topic Multibrokerapplication
```

```
PS C:\Users\hp> kafka-topics.bat --bootstrap-server localhost:9094 --delete --topic Multibrokerapplication
PS C:\Users\hp>
```

Note:

After issuing `--delete --topic` command for Kafka running in Windows, it will only mark the respective topic name for deletion but will not physically delete the topic folder present in `kafka-logs`, `kafka-logs-1` and `kafka-logs-2` folders in `%KAFKA_HOME%` location.

On the three Kafka broker terminal windows, you will see an *Error while renaming dir for test-topic-0 in log dir D:\ProgramFiles\Kafka\kafka-logs* and then Kafka server was shut down.

```

Administrator: Command Prompt
ic move (org.apache.kafka.common.utils.Utils)
java.nio.file.AccessDeniedException: D:\ProgramFiles\Kafka\kafka-logs-2\Multibrokerapplication-0 -> D:\ProgramFiles\Kafka\kafka-logs-2\Multibrokerapplication-0.f702bd4ff6034481a0f4d1ce45db19e5-delete
    at sun.nio.fs.WindowsException.translateToIOException(WindowsException.java:83)
    at sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:97)
    at sun.nio.fs.WindowsFileCopy.move(WindowsFileCopy.java:301)
    at sun.nio.fs.WindowsFileSystemProvider.move(WindowsFileSystemProvider.java:291)
    at java.nio.file.Files.move(Files.java:1395)
    at org.apache.kafka.common.utils.Utils.atomicMoveWithFallback(Utils.java:947)
    at kafka.log.LocalLog.$anonfun$renameDir$2(LocalLog.scala:113)
    at kafka.log.LocalLog.renameDir(LocalLog.scala:716)
    at kafka.log.UnifiedLog.$anonfun$renameDir$2(UnifiedLog.scala:680)
    at kafka.log.UnifiedLog.renameDir(UnifiedLog.scala:1922)
    at kafka.log.LogManager.asyncDelete(LogManager.scala:1313)
    at kafka.log.LogManager.$anonfun$asyncDelete$4(LogManager.scala:1350)
    at scala.Option.foreach(Option.scala:437)
    at kafka.log.LogManager.$anonfun$asyncDelete$3(LogManager.scala:1348)
    at kafka.log.LogManager.$anonfun$asyncDelete$3$adapted(LogManager.scala:1346)
    at scala.collection.mutable.HashSet$Node.foreach(HashSet.scala:450)
    at scala.collection.mutable.HashSet.foreach(HashSet.scala:376)
    at kafka.log.LogManager.asyncDelete(LogManager.scala:1346)
    at kafka.server.ReplicaManager.stopPartitions(ReplicaManager.scala:613)
    at kafka.server.ReplicaManager.stopReplicas(ReplicaManager.scala:548)
    at kafka.server.KafkaApis.handleStopReplicaRequest(KafkaApis.scala:324)
    at kafka.server.KafkaApis.handle(KafkaApis.scala:194)
    at kafka.server.KafkaRequestHandler.run(KafkaRequestHandler.scala:160)
    at java.lang.Thread.run(Thread.java:750)

[2024-09-26 03:39:17,424] ERROR Error while renaming dir for Multibrokerapplication-0 in log dir D:\ProgramFiles\Kafka\kafka-logs-2 (org.apache.kafka.storage.internals.log.LogDirFailureChannel)
java.nio.file.AccessDeniedException: D:\ProgramFiles\Kafka\kafka-logs-2\Multibrokerapplication-0 -> D:\ProgramFiles\Kafka\kafka-logs-2
[2024-09-26 03:39:17,413] INFO [ReplicaFetcherThread-0-0]: Shutdown completed (kafka.server.ReplicaFetcherThread)
[2024-09-26 03:39:17,434] WARN [ReplicaManager broker=2] Stopping serving replicas in dir D:\ProgramFiles\Kafka\kafka-logs-2 with uuid None because the log directory has failed. (kafka.server.ReplicaManager)
[2024-09-26 03:39:17,676] INFO [ReplicaFetcherManager on broker 2] Removed fetcher for partitions HashSet(_consumer_offsets-22, _consumer_offsets-4, _consumer_offsets-25, _consumer_offsets-49, _consumer_offsets-31, _consumer_offsets-37, _consumer_offsets-19, _consumer_offsets-13, _consumer_offsets-43, _consumer_offsets-1, _consumer_offsets-34, _consumer_offsets-7, _consumer_offsets-46, _consumer_offsets-16, _consumer_offsets-28, _consumer_offsets-10, _consumer_offsets-40) (kafka.server.ReplicaFetcherManager)
[2024-09-26 03:39:17,678] INFO [ReplicaAlterLogDirsManager on broker 2] Removed fetcher for partitions HashSet(_consumer_offsets-22, _consumer_offsets-4, _consumer_offsets-25, _consumer_offsets-49, _consumer_offsets-31, _consumer_offsets-37, _consumer_offsets-19, _consumer_offsets-13, _consumer_offsets-43, _consumer_offsets-1, _consumer_offsets-34, _consumer_offsets-7, _consumer_offsets-46, _consumer_offsets-16, _consumer_offsets-28, _consumer_offsets-10, _consumer_offsets-40) (kafka.server.ReplicaAlterLogDirsManager)
[2024-09-26 03:39:17,727] WARN [ReplicaManager broker=2] Broker 2 stopped fetcher for partitions _consumer_offsets-22, _consumer_offsets-4, _consumer_offsets-25, _consumer_offsets-49, _consumer_offsets-31, _consumer_offsets-37, _consumer_offsets-19, _consumer_offsets-13, _consumer_offsets-43, _consumer_offsets-1, _consumer_offsets-34, _consumer_offsets-7, _consumer_offsets-46, _consumer_offsets-16, _consumer_offsets-28, _consumer_offsets-10, _consumer_offsets-40 and stopped moving logs for partitions because they are in the failed log directory D:\ProgramFiles\Kafka\kafka-logs-2. (kafka.server.ReplicaManager)
[2024-09-26 03:39:17,731] WARN Stopping serving logs in dir D:\ProgramFiles\Kafka\kafka-logs-2 (kafka.log.LogManager)
[2024-09-26 03:39:17,742] ERROR Shutdown broker because all log dirs in D:\ProgramFiles\Kafka\kafka-logs-2 have failed (kafka.log.LogManager)

C:\Windows\System32>

```

Note: The above issue occurs in Windows operating system since Kafka is intended to run on Linux operating system. If we try to run Kafka in Windows natively, it may arise several issues over a period of time.

Therefore, it is recommended to run Apache Kafka on Windows through:

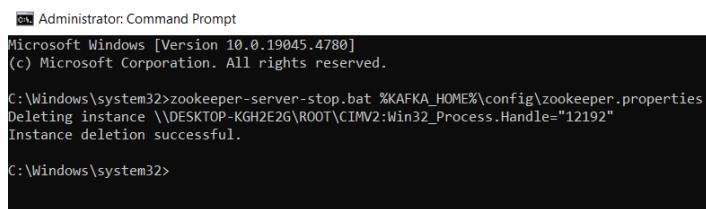
- **WSL2 or Docker** if using **Windows 10** or above.
- **Docker** if using **Windows 8** or below.

It is not recommended to run Kafka on the JVM on Windows, because it lacks some of the Linux specific features (for ex: POSIX). We will also run into issues at some point if we try to run Kafka on Windows without WSL2.

To resolve the above error, follow the below steps to manually delete the respective topic folder

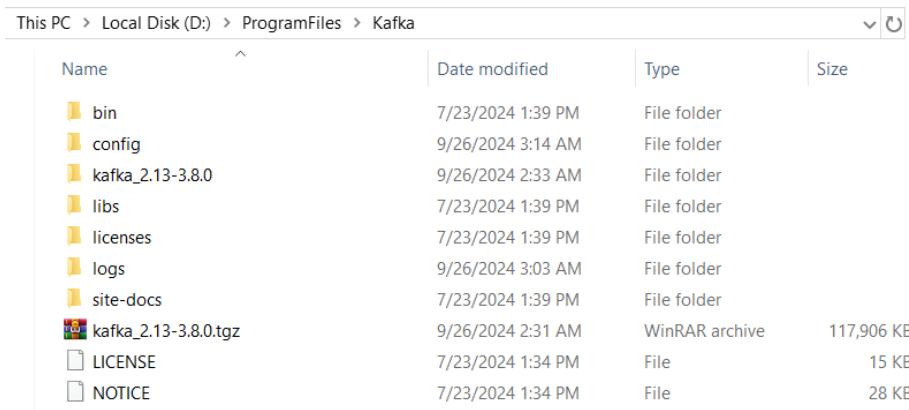
- Open new **Command Prompt** as **Administrator** and stop the zookeeper with this command:

```
zookeeper-server-stop.bat  
%KAFKA_HOME%\config\zookeeper.properties
```



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt". The text output is:
Microsoft Windows [Version 10.0.19045.4780]
(c) Microsoft Corporation. All rights reserved.
C:\Windows\system32>zookeeper-server-stop.bat %KAFKA_HOME%\config\zookeeper.properties
Deleting instance \\DESKTOP-KGH2E2G\ROOT\CIMV2:Win32_Process.Handle="12192"
Instance deletion successful.
C:\Windows\system32>

- Go to **%KAFKA_HOME%** location and delete **zookeeper**, **kafka-logs**, **kafka-logs-1** and **kafka-logs-2** folders.



This PC > Local Disk (D:) > ProgramFiles > Kafka

Name	Date modified	Type	Size
bin	7/23/2024 1:39 PM	File folder	
config	9/26/2024 3:14 AM	File folder	
kafka_2.13-3.8.0	9/26/2024 2:33 AM	File folder	
libs	7/23/2024 1:39 PM	File folder	
licenses	7/23/2024 1:39 PM	File folder	
logs	9/26/2024 3:03 AM	File folder	
site-docs	7/23/2024 1:39 PM	File folder	
kafka_2.13-3.8.0.tgz	9/26/2024 2:31 AM	WinRAR archive	117,906 KB
LICENSE	7/23/2024 1:34 PM	File	15 KB
NOTICE	7/23/2024 1:34 PM	File	28 KB

- Open new **Command Prompt** as **Administrator** and start zookeeper with this command:

```
zookeeper-server-start.bat  
%KAFKA_HOME%\config\zookeeper.properties
```

```

Administrator: Command Prompt - zookeeper-server-start.bat D:\ProgramFiles\Kafka\config\zookeeper.properties
[2024-09-26 03:48:21,922] INFO Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory (org.apache.zookeeper.server.ServerCnxnFactory)
[2024-09-26 03:48:21,924] WARN maxCnxns is not configured, using default value 0. (org.apache.zookeeper.server.ServerCnxnFactory)
[2024-09-26 03:48:21,927] INFO Configuring NIO connection handler with 10s sessionless connection timeout, 1 selector thread(s), 8 worker threads, and 64 kB direct buffers. (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-09-26 03:48:21,937] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2024-09-26 03:48:21,971] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)
[2024-09-26 03:48:21,971] INFO Using org.apache.zookeeper.server.watch.WatchManager as watch manager (org.apache.zookeeper.server.watch.WatchManagerFactory)
[2024-09-26 03:48:21,973] INFO zookeeper.snapshotSizeFactor = 0.33 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 03:48:21,974] INFO zookeeper.commitLogCount=500 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 03:48:21,986] INFO zookeeper.snapshot.compression.method = CHECKED (org.apache.zookeeper.server.persistence.SnapStream)
[2024-09-26 03:48:21,986] INFO Snapshotting: 0x0 to D:\ProgramFiles\Kafka\zookeeper\version-2\snapshot.0 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-09-26 03:48:21,993] INFO Snapshot loaded in 19 ms, highest zxid is 0x0, digest is 1371985504 (org.apache.zookeeper.server.ZKDatabase)
[2024-09-26 03:48:22,005] INFO Snapshotting: 0x0 to D:\ProgramFiles\Kafka\zookeeper\version-2\snapshot.0 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2024-09-26 03:48:22,008] INFO Snapshot taken in 3 ms (org.apache.zookeeper.server.ZooKeeperServer)
[2024-09-26 03:48:22,064] INFO zookeeper.request_throttler.shutdownTimeout = 10000 ms (org.apache.zookeeper.server.RequestThrottler)
[2024-09-26 03:48:22,065] INFO PrepRequestProcessor (sid:0) started, reconfigEnabled=false (org.apache.zookeeper.server.PrepRequestProcessor)
[2024-09-26 03:48:22,117] INFO Using checkIntervalMs=60000 maxPerMinute=10000 maxNeverUsedIntervalMs=0 (org.apache.zookeeper.server.ContainerManager)
[2024-09-26 03:48:22,120] INFO ZooKeeper audit is disabled. (org.apache.zookeeper.audit.ZKAuditProvider)

```

- Open three **Command Prompts** as **Administrator** and start three Kafka brokers with these commands:

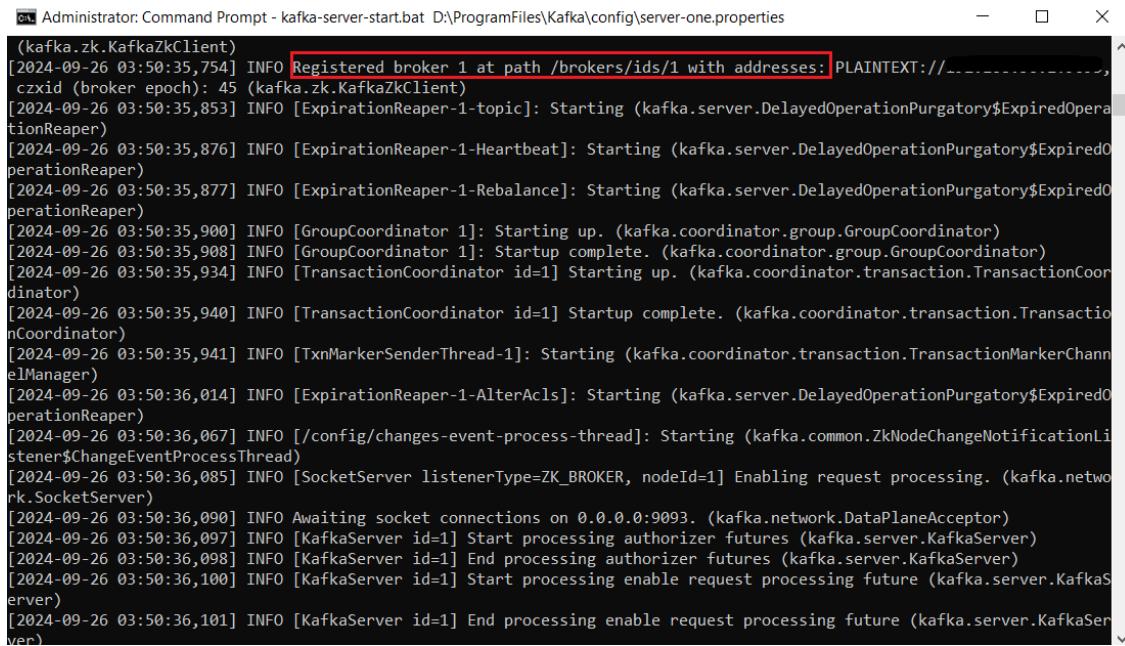
kafka-server-start.bat %KAFKA_HOME%\config\server.properties

```

Administrator: Command Prompt - kafka-server-start.bat D:\ProgramFiles\Kafka\config\server.properties
(kafka.zk.KafkaZkClient)
[2024-09-26 03:49:04,295] INFO Registered broker 0 at path /brokers/ids/0 with addresses PLAINTEXT://...:9092, (kafka.zk.KafkaZkClient)
[2024-09-26 03:49:04,373] INFO [ExpirationReaper-0-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:49:04,387] INFO [ExpirationReaper-0-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:49:04,388] INFO [ExpirationReaper-0-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:49:04,412] INFO Successfully created /controller_epoch with initial epoch 0 (kafka.zk.KafkaZkClient)
[2024-09-26 03:49:04,445] INFO [GroupCoordinator 0]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:49:04,454] INFO [GroupCoordinator 0]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:49:04,472] INFO Feature ZK node created at path: /feature (kafka.server.FinalizedFeatureChangeListener)
[2024-09-26 03:49:04,483] INFO [TransactionCoordinator id=0] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:49:04,494] INFO [TxnMarkerSenderThread-0]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2024-09-26 03:49:04,494] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:49:04,528] INFO [MetadataCache brokerId=0] Updated cache from existing None to latest Features(metadataVersion=3.8-IV0, finalizedFeatures={}, finalizedFeaturesEpoch=0). (kafka.server.metadata.ZkMetadataCache)
[2024-09-26 03:49:04,610] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:49:04,676] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2024-09-26 03:49:04,718] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Enabling request processing. (kafka.network.SocketServer)
[2024-09-26 03:49:04,731] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.DataPlaneAcceptor)
[2024-09-26 03:49:04,737] INFO [KafkaServer id=0] Start processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:49:04,739] INFO [KafkaServer id=0] End processing authorizer futures (kafka.server.KafkaServer)

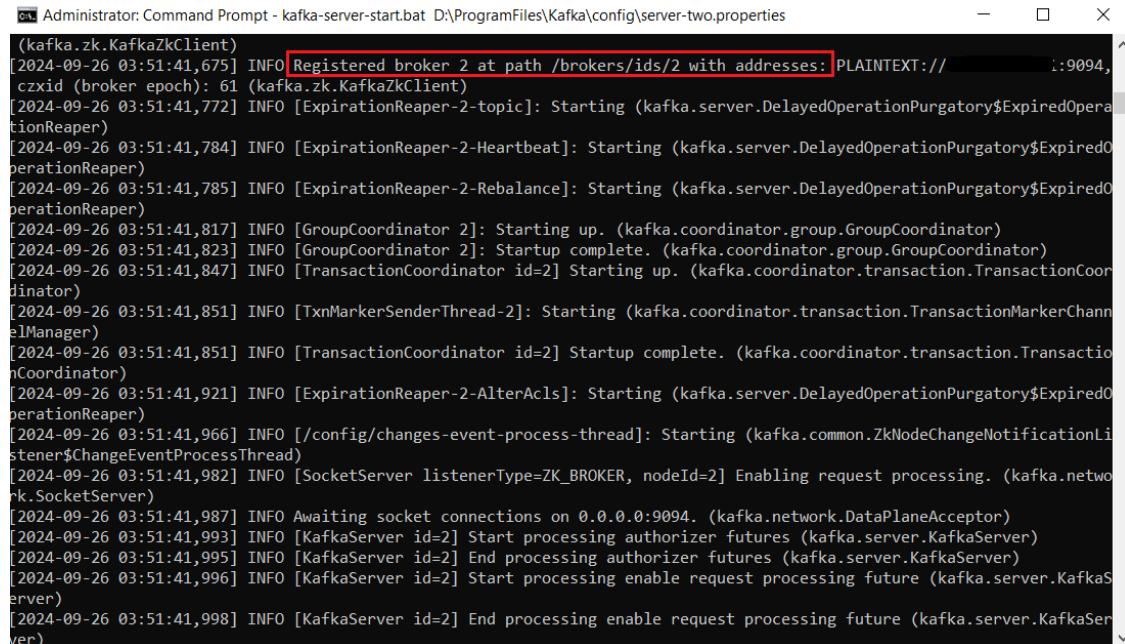
```

```
kafka-server-start.bat %KAFKA_HOME%\config\server-one.properties
```



```
(kafka_zk.KafkaZkClient)
[2024-09-26 03:50:35,754] INFO Registered broker 1 at path /brokers/ids/1 with addresses: PLAINTEXT://.....,czxid (broker epoch): 45 (kafka_zk.KafkaZkClient)
[2024-09-26 03:50:35,853] INFO [ExpirationReaper-1-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:50:35,876] INFO [ExpirationReaper-1-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:50:35,877] INFO [ExpirationReaper-1-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:50:35,900] INFO [GroupCoordinator 1]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:50:35,908] INFO [GroupCoordinator 1]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:50:35,934] INFO [TransactionCoordinator id=1] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:50:35,940] INFO [TransactionCoordinator id=1] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:50:35,941] INFO [TxnMarkerSenderThread-1]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2024-09-26 03:50:36,014] INFO [ExpirationReaper-1-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:50:36,067] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2024-09-26 03:50:36,085] INFO [SocketServer listenerType=ZK_BROKER, nodeId=1] Enabling request processing. (kafka.network.SocketServer)
[2024-09-26 03:50:36,090] INFO Awaiting socket connections on 0.0.0.0:9093. (kafka.network.DataPlaneAcceptor)
[2024-09-26 03:50:36,097] INFO [KafkaServer id=1] Start processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:50:36,098] INFO [KafkaServer id=1] End processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:50:36,100] INFO [KafkaServer id=1] Start processing enable request processing future (kafka.server.KafkaServer)
[2024-09-26 03:50:36,101] INFO [KafkaServer id=1] End processing enable request processing future (kafka.server.KafkaServer)
```

```
kafka-server-start.bat %KAFKA_HOME%\config\server-two.properties
```



```
(kafka_zk.KafkaZkClient)
[2024-09-26 03:51:41,675] INFO Registered broker 2 at path /brokers/ids/2 with addresses: PLAINTEXT://.....:9094, czxid (broker epoch): 61 (kafka_zk.KafkaZkClient)
[2024-09-26 03:51:41,772] INFO [ExpirationReaper-2-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:51:41,784] INFO [ExpirationReaper-2-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:51:41,785] INFO [ExpirationReaper-2-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:51:41,817] INFO [GroupCoordinator 2]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:51:41,823] INFO [GroupCoordinator 2]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2024-09-26 03:51:41,847] INFO [TransactionCoordinator id=2] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:51:41,851] INFO [TxnMarkerSenderThread-2]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2024-09-26 03:51:41,851] INFO [TransactionCoordinator id=2] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
[2024-09-26 03:51:41,921] INFO [ExpirationReaper-2-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2024-09-26 03:51:41,966] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
[2024-09-26 03:51:41,982] INFO [SocketServer listenerType=ZK_BROKER, nodeId=2] Enabling request processing. (kafka.network.SocketServer)
[2024-09-26 03:51:41,987] INFO Awaiting socket connections on 0.0.0.0:9094. (kafka.network.DataPlaneAcceptor)
[2024-09-26 03:51:41,993] INFO [KafkaServer id=2] Start processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:51:41,995] INFO [KafkaServer id=2] End processing authorizer futures (kafka.server.KafkaServer)
[2024-09-26 03:51:41,996] INFO [KafkaServer id=2] Start processing enable request processing future (kafka.server.KafkaServer)
[2024-09-26 03:51:41,998] INFO [KafkaServer id=2] End processing enable request processing future (kafka.server.KafkaServer)
```

Note that we have deleted all topics and will have to start creating topic freshly.

6. Simple Kafka Project using Java:

In the previous section, we used Kafka command line tools such as `kafka-topics.bat`, `kafka-console-producer.bat` and `kafka-console-consumer.bat` to create a Kafka topic and start producing and subscribing data. In this section, we will create a simple Kafka project to do the same activities using Java programming language.

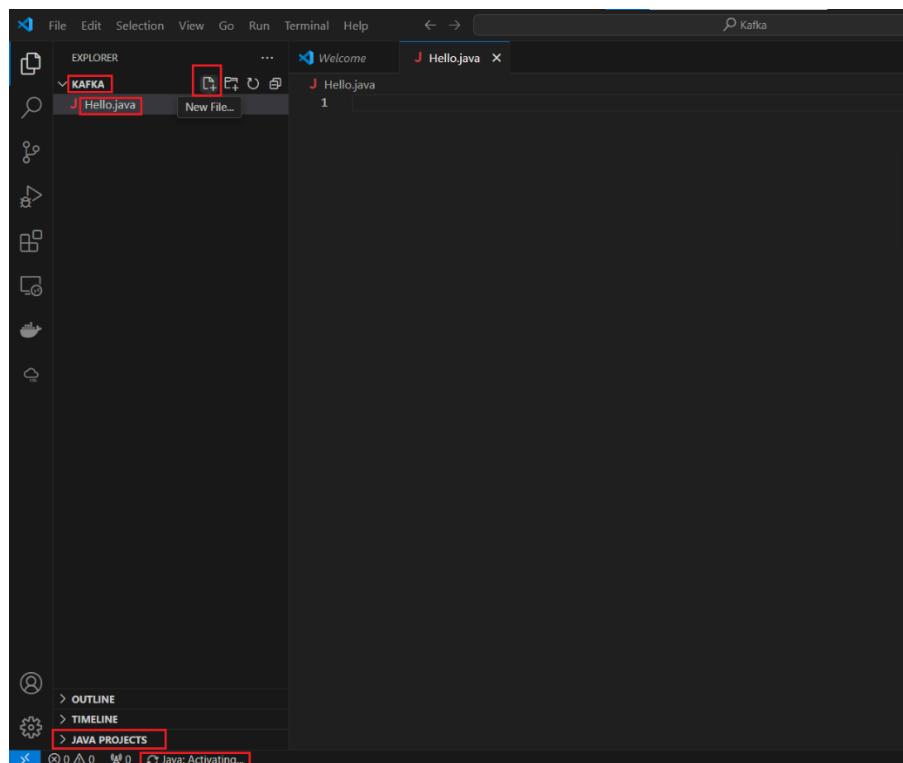
6.1. Launch Java IDE:

To create a Kafka project using Java, we need an **IDE tool** such as Eclipse, VS Code, IntelliJ IDEA, Notepad, etc. and **JDK 1.8** or higher.

In this tutorial, we will use **VS Code** to create a java project. If you do not have it installed already, you can download from [official Microsoft site](#) and install [Extension Pack for Java](#) extension.

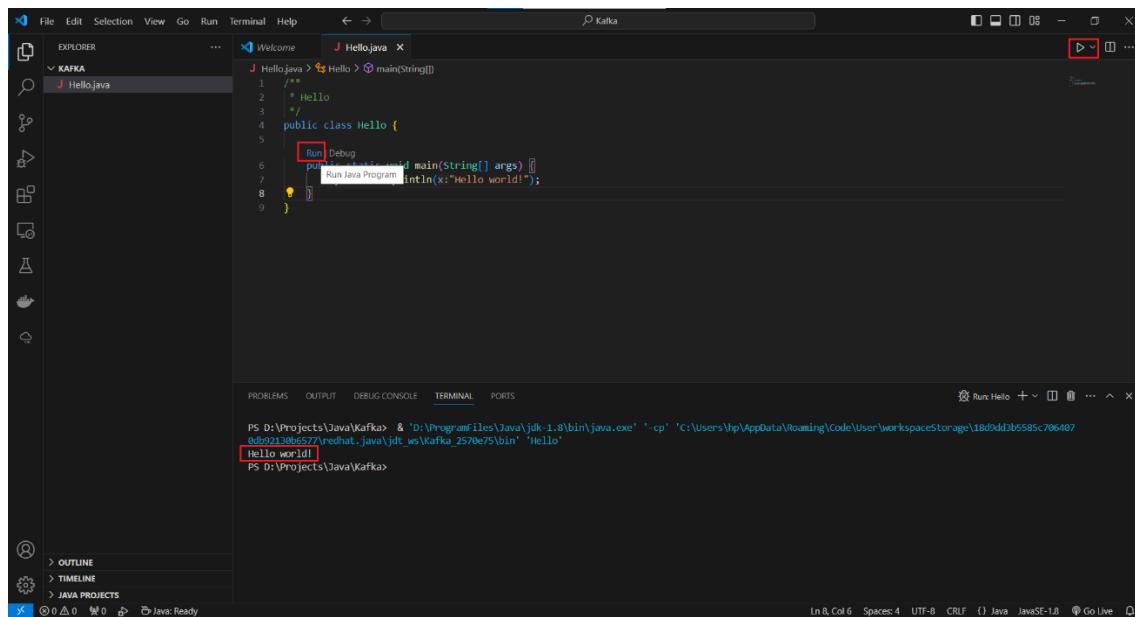
In **VS Code** application, open a new folder and navigate to D drive to create a folder named **Kafka** and select it in your local system.

On the **Explorer** window, click on **New File** button next to **KAFKA** folder and name it with .java extension (for example, `Hello.java`) after which it starts activating Java extension and you can see **JAVA PROJECTS** section on the bottom left of the application.

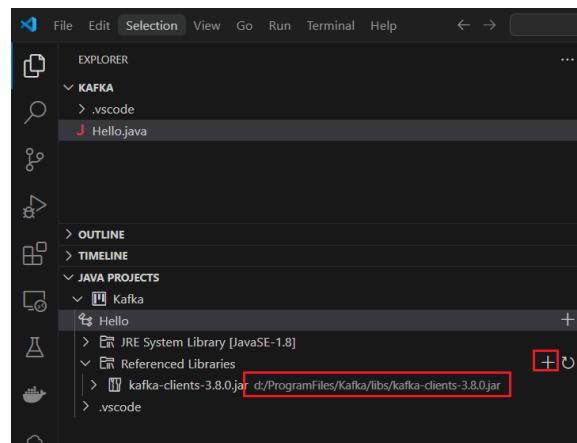


Once the Java extension is ready, enter the below code in Hello.java file and click on **Run** link or button to see the output of java program on the **Terminal**.

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```



Now that our sample Java code is working, we will write java programs to create a Kafka topic, producer and consumer application. To do this, expand **JAVA PROJECTS** window on the left bottom window of the VS Code, then expand **KAFKA** project and click on **+** icon next to **Referenced Libraries** and select `D:\ProgramFiles\Kafka\libs\kafka-clients-3.8.0.jar` file.



Note that Apache Kafka provides `kafka-clients-xxx.jar` library (*located at %KAFKA_HOME%\libs folder*) which contains admin API, producer API and consumer APIs using which we will create a topic, send data and read data programmatically.

Before proceeding further, make sure Kafka services are up and running. If not already running, open 4 **Command Prompts** in **Administrator** mode and run the following commands to start Zookeeper first and then 3 Kafka brokers:

```
zookeeper-server-start.bat %KAFKA_HOME%\config\zookeeper.properties
```

```
kafka-server-start.bat %KAFKA_HOME%\config\server.properties
```

```
kafka-server-start.bat %KAFKA_HOME%\config\server-one.properties
```

```
kafka-server-start.bat %KAFKA_HOME%\config\server-two.properties
```

6.2. Create Topic:

Apache Kafka provides the `admin` API in which we need to start with using `create()` method of `Admin` interface to establish a connection to Kafka cluster and create a topic.

- First, we should create an `Admin` object by providing a set of key-value pairs as configuration properties to `create()` method. It is required to specify at least `bootstrap.servers` property which takes a list of host/port pairs used to establish an initial connection to the Kafka cluster. For the complete list of `admin` properties, refer to [Apache Kafka Admin Client Configs Documentation](#).

Note: In the `Properties` object, we can set the property `bootstrap.servers` directly or by calling `BOOTSTRAP_SERVERS_CONFIG` variable in `AdminClientConfig` class.

- Once the `Admin` object is created, one or more topics can be created using `createTopics()` method which takes a collection of `NewTopic` objects where we can specify topic name, number of partitions and replication factor for a specific topic. If you omit partitions and (or) replicationFactor arguments in `NewTopic` object then `topicName` is created with default partitions and replication factor values that are mentioned in `%KAFKA_HOME%\config\server.properties` file.
- `createTopics()` method returns `CreateTopicsResult` object of which `values()` method of which `get()` method is needed to get the result of a specific topic created. This `get()` method returns `KafkaFuture<Void>` object.

- KafkaFuture<Void> object has get() method which helps to wait until the topic creation is either completed or failed. This method can throw 2 exceptions - InterruptedException and ExecutionException - in case of any failure.

In VS Code, create a new file named CreateTopic.java and enter the following code:

```

import java.util.Collections;
import java.util.Properties;
import java.util.concurrent.ExecutionException;

import org.apache.kafka.clients.admin.Admin;
import org.apache.kafka.clients.admin.AdminClientConfig;
import org.apache.kafka.clients.admin.CreateTopicsResult;
import org.apache.kafka.clients.admin.NewTopic;
import org.apache.kafka.common.KafkaFuture;

public class CreateTopic {
    public static void main(String[] args) {
        // Set topic name, partitions and replication factor
        String topic = "topic-1";
        int partitions = 4;
        short replicationFactor = 3;

        // Create instance for properties to access admin configs
        Properties properties = new Properties();
        // Assign bootstrap servers config value
        //properties.setProperty("bootstrap.servers", "localhost:9092");
        properties.setProperty(AdminClientConfig.BOOTSTRAP_SERVERS_CONFIG,
        "localhost:9092");

        // Create instance for admin client
        Admin admin = Admin.create(properties);

        try {
            // Create a compacted topic
            CreateTopicsResult createResult =
admin.createTopics(Collections.singleton(new NewTopic(topic, partitions,
replicationFactor)));
            // Call values() to get result for a specific topic
            KafkaFuture<Void> future = createResult.values().get(topic);

            // Call get() to block until the topic creation is complete or
has failed.
            // Incase failed, the ExecutionException displays the underlying
cause.
            future.get();
            System.out.println("'" + topic + "' topic has been created
successfully");
            } catch (InterruptedException | ExecutionException e) {
                e.printStackTrace();
            }
        }
    }
}

```

After the above code is entered, click on **+** button on **Terminal** window and select **Command Prompt** where you need to run the following commands to compile the **CreateTopic.java** file and run it.

```
javac -cp %KAFKA_HOME%\libs\* CreateTopic.java
java -cp %KAFKA_HOME%\libs\*;. CreateTopic
```

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a project structure with a **Kafka** folder containing **CreateTopic.java** and **Hello.java**.
- Code Editor:** Displays the **CreateTopic.java** code, which contains Java code for creating a topic named "topic-1".
- Terminal:** Shows the command line output of the compilation and execution process. It includes the command `javac -cp %KAFKA_HOME%\libs* CreateTopic.java`, the resulting class file `D:\Projects\Java\Kafka\CreateTopic.class`, and the output of the Java application showing the topic creation message: `'topic-1' topic has been created successfully`.
- Command Palette:** A dropdown menu is open, with the **Command Prompt** option highlighted and selected.

You can see that a topic name `topic-1` has been created.

6.2.1. List Topics:

We can also list out the available topics in the cluster using `listTopics()` method of `Admin` object which returns `ListTopicsResult` object of which `names()` method of which `get()` method can be used to retrieve the list of topic names available in Kafka cluster.

In VS Code, enter the following code before `catch()` block in `CreateTopic.java` file. Since `topic-1` is already created, change the topic variable value to `topic-2` to create a new topic and then list out both topics:

```
// List all topics in Kafka cluster
ListTopicsResult listResult = admin.listTopics();
System.out.println("List of all topics:");
System.out.println(listResult.names().get());
```

Once the above code is entered, compile CreateTopic.java file and run it

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows a project named "KAFKA" containing ".vscode", "CreateTopic.class", "CreateTopic.java", and "Hello.java".
- TERMINAL:** Shows the command "D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%\libs* CreateTopic" being run.
- OUTPUT:** Shows log messages from log4j:
 - "topic-1" topic has been created successfully
 - "topic-2" topic has been created successfully
 - list of all topics: [topic-1, topic-2]
- PROBLEMS:** Shows no problems.
- DEBUG CONSOLE:** Shows no content.
- TERMINAL:** Shows the command "D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%\libs* CreateTopic".
- PORTS:** Shows no content.

You can see that it created a new topic `topic-2` and listed out both topics in the output window.

6.2.2. Describe Topics:

Similar to `listTopics()`, we can get the topic details using `describeTopics()` method of `Admin` object which returns `DescribeTopicsResult` object of which `values()` method of which `get()` method is needed to get the result of a specific topic created. This `get()` method returns `KafkaFuture<Void>` object of which `allTopicNames()` method of which `get()` method can be used to retrieve the details such of topic name, partitions, leader, replication factor etc.

In VS Code, enter the following code before `catch()` block in `CreateTopic.java` file. Since `topic-2` is already created, change the topic variable value to `topic-3` to create a new topic and provide details of each topic:

```
// Describe all topics
DescribeTopicsResult descResult =
admin.describeTopics(listResult.names().get());
System.out.println("Describe all topics:");
System.out.println(descResult.allTopicNames().get());
```

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows a project named "KAFKA" containing files ".vscode", "CreateTopic.class", "CreateTopic.java", and "Hello.java".
- TERMINAL**: Displays the output of running the Java code. It shows the creation of a topic named "topic-2" and the listing of all topics ("topic-1", "topic-2", "topic-3").
- OUTPUT**: Shows log messages from Kafka, indicating no appenders were found for the logger.
- PROBLEMS**: No problems are listed.
- DEBUG CONSOLE**: No content.
- TERMINAL**: Shows the command run: "D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%libs* CreateTopic". The output includes log4j WARN messages about no appenders and Kafka topic creation logs.
- PORTS**: No content.

You can see that it created a new topic `topic-2` and described all topics in the output window.

6.2.3. Delete Topics:

Similar to `createTopics()`, we can delete one or more topics using `deleteTopics()` method of Admin object which returns `DeleteTopicsResult` object which has `topicNameValues()` method of which `get()` method returns `KafkaFuture<Void>` object. The `get()` method of `KafkaFuture<Void>` objects ensures to wait until the topic deletion is either completed or failed and throws 2 exceptions - `InterruptedException` and `ExecutionException` - in case of any failure.

The following code deletes 3 topics in the Kafka cluster:

```
// Delete multiple topics
ArrayList<String> topics= new ArrayList<String>();
topics.add("topic-1");
topics.add("topic-2");
topics.add("topic-3");
DeleteTopicsResult delResult = admin.deleteTopics(topics);
for (String topicName : topics) {
    KafkaFuture<Void> delFuture =
delResult.topicNameValues().get(topicName);
    // Call get() to block until the topic deletion is complete
or has failed
    delFuture.get();
    System.out.println(topicName + " has been deleted
successfully");
}
```

Note:

After executing the above code, it appears that topics were deleted, but it marks for deletion without deleting the underlying files in %KAFA_HOME%\kafka-logs folder and shuts down all Kafka brokers abruptly. This is a known issue in Windows operating system. To resolve this, we should manually delete all files under kafka-logs, kafka-logs-1, kafka-logs-2, and zookeeper folders in %KAFA_HOME% location then restart Zookeeper and Kafka brokers services.

6.3. Create Producer:

Now that topics are created in Kafka, let us create a Producer application.

Apache Kafka provides the `producer API` in which we need to start with using `KafkaProducer` class which implements `Producer` class to establish a connection to Kafka cluster and produce data.

- First, we should instantiate the `Producer` class by providing a set of key-value pairs as configuration properties. Some of the required properties are highlighted below:
 - `bootstrap.servers`: A list of host/port pairs (in the form of host1:port1, host2:port2,...) used for establishing initial connection to Kafka cluster.
 - `key.serializer`: A serializer class for key that implements the `org.apache.kafka.common.serialization.StringSerializer` interface.
 - `value.serializer`: A serializer class for value that implements the `org.apache.kafka.common.serialization.StringSerializer` interface.

Other commonly used properties are `client.id` (unique id for the producer application), `producer.type` (sync or async), `retries` (auto-retry on failure), `batch.size` (batch records), `buffer.memory` (memory to buffer records to be sent to server).

For the complete list of `producer` properties, refer to [Apache Kafka Producer Configs Documentation](#).

Note: In the `Properties` object, we can set properties `bootstrap.servers`, `key.serializer` and `value.serializer` directly or by calling `BOOTSTRAP_SERVERS_CONFIG`, `KEY_SERIALIZER_CLASS_CONFIG` and `VALUE_SERIALIZER_CLASS_CONFIG` variables in `ProducerConfig` class.

- The `send()` method of `KafkaProducer` class is used to send a producer record to Kafka. This method takes `ProducerRecord` object and `callback` object (this is optional) as arguments.
- The `ProducerRecord` class manages records to be sent to Kafka for which we need to pass `topic` (topic name) and `value` (data) as mandatory arguments to `ProducerRecord` constructor. Additionally, we can provide `key`, `partition` and `timestamp` values as well to `ProducerRecord`.
- It is important to flush the data sent and close the producer using `flush()` and `close()` methods provided by `producer` object to avoid any resource leaks.

In VS Code, create a new file named `Producer.java` and enter the following code:

```
import java.util.Properties;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;

public class Producer {
    public static void main(String[] args) {
        if (args.length < 2) {
            System.err.println("Invalid arguments passed, you need to pass
topic name and message parameters");
            System.exit(1);
        }

        // Read topic name and message passed as input arguments
        String topicName = args[0];
        String message = args[1];

        // Create instance for properties to access admin configs
        Properties properties = new Properties();

        // Assign bootstrap servers, key and value serializer config values
        //properties.put("bootstrap.servers", "localhost:9092");

        //properties.put("key.serializer", "org.apache.kafka.common.serialization.Ser
ializer");
        //properties.put("value.serializer", "org.apache.kafka.common.serialization.S
erializer");
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
        "localhost:9092");

        properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.kafk
a.common.serialization.Serializer");

        properties.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apa
che.kafka.common.serialization.Serializer");
    }
}
```

```

        // Create instance for Kafka Producer
        KafkaProducer<String, String> producer = new KafkaProducer<String,
String>(properties);

        // Create instance for ProducerRecord using topic name and message
        ProducerRecord<String, String> producerRecord = new
ProducerRecord<String, String>(topicName, message);

        // Send producer record
        producer.send(producerRecord);
        System.out.println("Data produced successfully");

        // Flush the producer data
        producer.flush();

        // Close the producer object
        producer.close();
    }
}

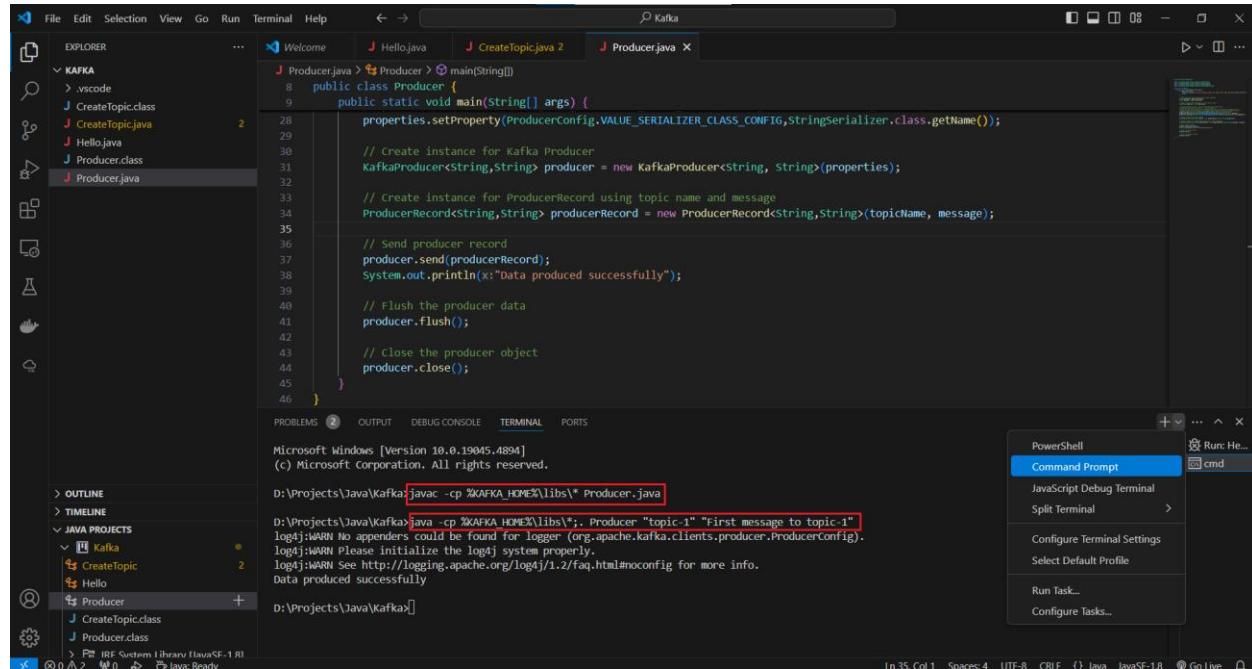
```

After the above code is entered, click on **+** button on **Terminal** window and select **Command Prompt** where you need to run the following commands to compile the **Producer.java** file and run it.

```

javac -cp %KAFKA_HOME%\libs\* Producer.java
java -cp %KAFKA_HOME%\libs\*;. Producer "topic-1" "First message to
topic-1"

```



You can see that the data has been produced successfully into the topic. To verify the same, open **Command Prompt** and run the following consumer command to see data in topic-1.

```
kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic topic-1 --from-beginning
```



```
cmd Command Prompt - kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic topic-1 --from-beginning
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic topic-1 --from-beginning
First message to topic-1
```

In **VS Code**, run the following command on **Terminal** window to produce a second message to topic-1.

```
java -cp %KAFKA_HOME%\libs\*;. Producer "topic-1" "Second message to topic-1"
```

You can see this second message in consumer window



```
cmd Command Prompt - kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic topic-1 --from-beginning
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic topic-1 --from-beginning
First message to topic-1
Second message to topic-1
```

6.3.1. Producer Callbacks:

We can do Producer callbacks to understand how a producer sends data to Kafka i.e. if the data was correctly produced, where it was produced, its offset value, partition value, etc. The callback function used by producer is `onCompletion()` which is implemented for asynchronously handling the request completion and is implemented in the block where producer sends data to Kafka.

The `onCompletion()` method expects two arguments – `RecordMetadata` (*metadata of record regarding partition and offset*) and `Exception` (*two possible exceptions – Retirable exception which indicates the message may be sent and Non-retirable exception which indicates message will never be sent*).

In Producer.java file, modify the producer.send() line with the below code to display recordMetadata output once the record is published into Kafka.

```
// Send producer record
producer.send(producerRecord, new Callback() {
    //Record metadata on completion
    public void onCompletion(RecordMetadata recordMetadata,
Exception e) {
        if (e == null) {
            System.out.println("Data produced successfully with the
following metadata");
            System.out.println("\t Topic: " +
recordMetadata.topic());
            System.out.println("\t Partition: " +
recordMetadata.partition());
            System.out.println("\t Offset: " +
recordMetadata.offset());
            System.out.println("\t Timestamp: " +
recordMetadata.timestamp());
        }
        else {
            System.err.println("Exception while producing data: " +
e);
        }
    }
});
```

Note that the above code will automatically import
org.apache.kafka.clients.producer.Callback class.

Then compile and run Producer.java program with these commands:

```
javac -cp %KAFKA_HOME%\libs\* Producer.java
java -cp %KAFKA_HOME%\libs\*;. Producer "topic-1" "Third message to
topic-1"
```

```

File Edit Selection View Go Run Terminal Help ← → ⌂ Kafka
EXPLORER ... Welcome J Hello.java J CreateTopic.java 2 J Producer.java
KAFKA > vscode
J CreateTopic.class
J CreateTopic.java 2
J Hello.java
J Producer.class
J Producer.java
J Producer$1.class
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data produced successfully
D:\Projects\Java\Kafka>javac -cp %KAFKA_HOME%\libs\* Producer.java
D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%\libs\* Producer "topic-1" "third message to topic-1"
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Data produced successfully with the following metadata
Topic: topic-1
Partition: 0
Offset: 0
Timestamp: 1727196681084
Run: Hel... cmd
cmd
D:\Projects\Java\Kafka>

```

6.3.2. Produce Data with Same Key:

When a same key is used to produce messages, those will go into same partition.

In VS Code, modify Producer.java code with the following lines:

```

import java.util.Properties;
import java.util.concurrent.ExecutionException;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.serialization.StringSerializer;

public class Producer {
    public static void main(String[] args) throws InterruptedException,
ExecutionException {
        // Create instance for properties to access admin configs
        Properties properties = new Properties();

        // Assign bootstrap servers, key and value serializer config values
        //properties.put("bootstrap.servers", "localhost:9092");

        //properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        //properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
"localhost:9092");
    }
}

```

```

properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

properties.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

        // Create instance for Kafka Producer
        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(properties);

        // Generate producer records iteratively
        for (int i = 0; i < 10; i++) {
            String topic = "topic-1";
            String key = "id_" + Integer.toString(i);
            String value = "Message " + Integer.toString(i) + " sent";
            ProducerRecord<String, String> producerRecord = new
ProducerRecord<String, String>(topic, key, value);

            // Send producer record
            producer.send(producerRecord, new Callback() {
                //Record metadata on completion
                public void onCompletion(RecordMetadata recordMetadata,
Exception e) {
                    if (e == null) {
                        System.out.println("Produced the following data: ");
                        System.out.println("\t Key: " + producerRecord.key() +
", Value: " + producerRecord.value());
                        System.out.println("Successfully received the
following metadata:");
                        System.out.printf("\t Topic: %s, Partition: %d,
Offset: %d, Timestamp: %d \n",
recordMetadata.topic(), recordMetadata.partition(),
recordMetadata.offset(), recordMetadata.timestamp());
                    }
                    else {
                        System.err.println("Exception while producing data:
" + e);
                    }
                }
            }).get(); //sending synchronous data forcefully
        }

        // Flush the producer data
        producer.flush();

        // Close the producer object
        producer.close();
    }
}

```

Then compile and run `Producer.java` program with these commands:

```

javac -cp %KAFKA_HOME%\libs\* Producer.java
java -cp %KAFKA_HOME%\libs\*;. Producer

```

The screenshot shows the VS Code interface with the Kafka project open. The Explorer sidebar shows files like Hello.java, CreateTopic.java, Producer.java, and their corresponding class files. The Terminal tab shows the command `javac -cp %KAFKA_HOME%\libs* Producer.java` being run, followed by the log output from the Kafka producer. The log output indicates that 10 messages were produced, each with key `key_0` and value `Message 0 sent` through `Message 9 sent`, all ending up in partition 1.

```

javac -cp %KAFKA_HOME%\libs\* Producer.java
D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%\libs\* Producer
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Produced the following data:
[Key: key_0] Value: Message 0 sent
Successfully received the following metadata:
Topic: topic-1, Partition: 1 Offset: 2, Timestamp: 1727303690933
Produced the following data:
[Key: key_0] Value: Message 1 sent
Successfully received the following metadata:
Topic: topic-1, Partition: 1 Offset: 3, Timestamp: 1727303690997
Produced the following data:
[Key: key_0] Value: Message 2 sent
Successfully received the following metadata:
Topic: topic-1, Partition: 1 Offset: 4, Timestamp: 1727303691008
Produced the following data:
[Key: key_0] Value: Message 3 sent
Successfully received the following metadata:
Topic: topic-1, Partition: 1 Offset: 5, Timestamp: 1727303691022
Produced the following data:
[Key: key_0] Value: Message 4 sent
Successfully received the following metadata:
Topic: topic-1, Partition: 1 Offset: 6, Timestamp: 1727303691034

```

Here, you can see that all messages are going into partition 1 since we used the same key named `key_0`.

6.3.3. Produce Data with Different Keys:

The key in Kafka help to uniquely identify a partition from other partitions. We can send synchronous messages to Kafka forcefully using `get()` method of `ProducerRecord.send()` method.

In VS Code, modify `Producer.java` code with the highlighted line:

```

// Generate producer records iteratively
for (int i = 0; i < 10; i++) {
    String topic = "topic-1";
    String key = "id "+ Integer.toString(i);
    String value = "Message " + Integer.toString(i) + " sent";
    ProducerRecord<String, String> producerRecord = new
ProducerRecord<String, String>(topic, key, value);

```

Then compile and run `Producer.java` program with these commands:

```

javac -cp %KAFKA_HOME%\libs\* Producer.java
java -cp %KAFKA_HOME%\libs\*;. Producer

```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows files like Hello.java, CreateTopic.java, Producer.java, and Producer\$1.class.
- Terminal:** Displays the command `javac -cp %KAFKA_HOME%\libs* Producer.java` followed by the log output of the producer sending messages to a topic with partitions 0, 1, 2, 3, and 4.
- Output:** Shows the log output from the terminal.
- Status Bar:** Shows the current file is Producer.java, and the status "Java Ready".

```

D:\Projects\Java\Kafka>javac -cp %KAFKA_HOME%\libs\* Producer.java
D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%\libs\* Producer
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Produced the following data:
    Key: id_0 Value: Message 0 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 3] Offset: 0, Timestamp: 1727303413813
Produced the following data:
    Key: id_1 Value: Message 1 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 3] Offset: 1, Timestamp: 1727303414237
Produced the following data:
    Key: id_2 Value: Message 2 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 1] Offset: 0, Timestamp: 1727303414249
Produced the following data:
    Key: id_3 Value: Message 3 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 3] Offset: 2, Timestamp: 1727303414380
Produced the following data:
    Key: id_4 Value: Message 4 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 2] Offset: 0, Timestamp: 1727303414391
  
```

Here, you can see that records with keys id_0 and id_1 are sent to Partition 3, id_2 sent to partition 1, id_3 sent to partition3 and id_4 sent to partition 2.

6.4. Create Consumer:

Until now, we create producer to send messages to Kafka cluster. Now, let us create a consumer to consume messages from Kafka cluster.

Apache Kafka provides the `Consumer` API in which we need to start with using `KafkaConsumer` class which implements `Consumer` class to establish a connection to Kafka cluster and consume data.

- First, we should instantiate the `Consumer` class by providing a set of key-value pairs as configuration properties. Some of the required properties are highlighted below:
 - `bootstrap.servers`: A list of host/port pairs (in the form of host1:port1, host2:port2,...) used for establishing initial connection to Kafka cluster.
 - `key.deserializer`: A serializer class for key that implements the `org.apache.kafka.common.serialization.StringDeserializer` interface.
 - `value.deserializer`: A serializer class for value that implements the `org.apache.kafka.common.serialization.StringDeserializer` interface.

Other commonly used properties are:

- `group.id`: A unique identifier for a consumer group. This property is needed when the consumer uses group management functionality which is used to achieve the load balancing of consuming records from a Kafka topic (*i.e balancing topic partitions between all consumers in Consumer group so that one partition is assigned to exactly one consumer in a group*)
- `enable.auto.commit`: If set to true, then it commits offset value automatically to Zookepeer, otherwise not committed. By default, this value is set to true.
- `auto.commit.interval.ms`: The frequency of auto committing consumed offsets to Zookeeper. The default value is 5000 (i.e. 5 seconds).
- `auto.offset.reset`: This property is needed when no initial offset is present of current offset does not exist on server. It can have values – `earliest` (*set to earliest offset to read data from the beginning in a partition*), `latest` (*set to latest offset to read data from the last committed offset value*), `none` (*throws error when no previous offset found*). By default, it is set to `latest` value.

For the complete list of consumer properties, refer to [Apache Kafka Consumer Configs Documentation](#).

Note: In the `Properties` object, we can set properties `bootstrap.servers`, `key.deserializer` and `value.deserializer` directly or by calling the respective `BOOTSTRAP_SERVERS_CONFIG`, `KEY_DESERIALIZER_CLASS_CONFIG` and `VALUE_DESERIALIZER_CLASS_CONFIG` variables in `ConsumerConfig` class.

- The `KafkaConsumer` class provides `subscribe()` method to subscribe to a given topic in Kafka cluster.
- The `poll()` method of `KafkaConsumer` class allows to fetch data from topic partitions. This returns error if topics are not subscribed before polling.
- It is important to close the consumer using `close()` methods provided by `consumer` object to avoid any resource leaks.

In VS Code, create a new file named `Consumer.java` and enter the following code:

```
import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
```

```

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.serialization.StringDeserializer;

public class Consumer {
    public static void main(String[] args) {
        // Create instance for properties to access admin configs
        Properties properties = new Properties();

        // Assign bootstrap servers, key and value deserializer and other
        config values
        properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
"localhost:9092");
        properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());

properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserial
izer.class.getName());
        properties.put(ConsumerConfig.GROUP_ID_CONFIG, "consumer-grp1");
        properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        //properties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");

//properties.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");

        // Create instance for Kafka Consumer
        KafkaConsumer<String, String> consumer = new KafkaConsumer<String,
String>(properties);

        // Subscribe topic
        consumer.subscribe(Arrays.asList("topic-1"));

        // Polling for records
        while(true) {
            ConsumerRecords<String, String> records =
consumer.poll(Duration.ofMillis(100));
            for (ConsumerRecord<String, String> record : records) {
                System.out.printf("Topic: %s, Partition: %d, Offset: %d,
Key: %s, Value: %s\n",
record.topic(), record.partition(), record.offset(), record.key(),
record.value());
            }
        }
        //consumer.close();
    }
}

```

After the above code is entered, click on **+** button on **Terminal** window and select **Command Prompt** where you need to run the following commands to compile the `Consumer.java` file and run it.

```

javac -cp %KAFKA_HOME%\libs\* Consumer.java
java -cp %KAFKA_HOME%\libs\*;. Consumer

```

```

D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%\libs\*;. Consumer
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/Faq.html#noconfig for more info.
Topic: topic-1, Partition: 0, Key: id_2, Value: Message 2 sent
Topic: topic-1, Partition: 1, Key: id_8, Value: Message 8 sent
Topic: topic-1, Partition: 1, Offset: 2, Key: key_0, Value: Message 0 sent
Topic: topic-1, Partition: 1, Offset: 3, Key: key_0, Value: Message 1 sent
Topic: topic-1, Partition: 1, Offset: 4, Key: key_0, Value: Message 2 sent
Topic: topic-1, Partition: 1, Offset: 5, Key: key_0, Value: Message 3 sent
Topic: topic-1, Partition: 1, Offset: 6, Key: key_0, Value: Message 4 sent
Topic: topic-1, Partition: 1, Offset: 7, Key: key_0, Value: Message 5 sent
Topic: topic-1, Partition: 1, Offset: 8, Key: key_0, Value: Message 6 sent
Topic: topic-1, Partition: 1, Offset: 9, Key: key_0, Value: Message 7 sent
Topic: topic-1, Partition: 1, Offset: 10, Key: key_0, Value: Message 8 sent
Topic: topic-1, Partition: 1, Offset: 11, Key: key_0, Value: Message 9 sent
Topic: topic-1, Partition: 0, Key: id_0, Value: Message 0 sent
Topic: topic-1, Partition: 0, Offset: 1, Key: id_1, Value: Message 1 sent
Topic: topic-1, Partition: 0, Offset: 2, Key: id_3, Value: Message 3 sent
Topic: topic-1, Partition: 0, Offset: 3, Key: id_5, Value: Message 5 sent
Topic: topic-1, Partition: 0, Offset: 4, Key: id_7, Value: Message 7 sent
Topic: topic-1, Partition: 0, Offset: 5, Key: id_9, Value: Message 9 sent
Topic: topic-1, Partition: 1, Offset: 0, Key: id_4, Value: Message 4 sent
Topic: topic-1, Partition: 1, Offset: 1, Key: id_6, Value: Message 6 sent
Topic: topic-1, Partition: 1, Offset: 2, Key: id_9, Value: Message 9 sent

```

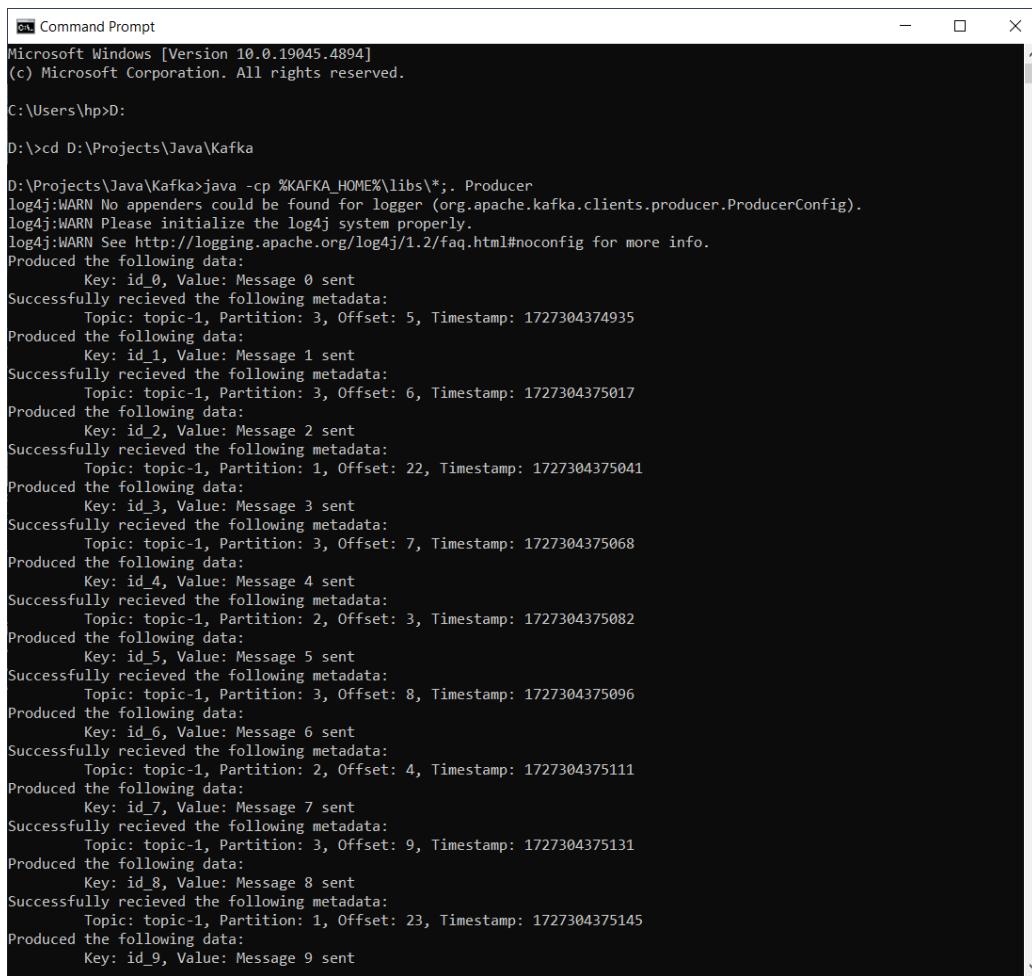
You can see that it keeps polling the topic `topic-1` every 100 milliseconds (0.1 second) and displays data fetched from Kafka. Note that although we set `auto.offset.reset` configuration to `earliest`, it works as `latest` and consumes the latest data produced to topic after the consumer was run (somehow, `earliest` offset is not working here). Enter **Ctrl + c** to exit out of consumer application.

To take the advantage of Consumer group functionality, open two command prompts and navigate to the location where `Consumer.java` file exists and run it:

```
D:
cd D:\Projects\Java\Kafka
java -cp %KAFKA_HOME%\libs\*;. Consumer
```

Open another command prompt and navigate to the location where `Producer.java` file exists and run it to produce some data into Kafka topic:

```
D:
cd D:\Projects\Java\Kafka
java -cp %KAFKA_HOME%\libs\*;. Producer
```



The screenshot shows a Microsoft Windows Command Prompt window titled "Command Prompt". The window displays the following text output from a Kafka producer:

```
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>D:

D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%\libs\*;. Producer
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.producer.ProducerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Produced the following data:
    Key: id_0, Value: Message 0 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 3, Offset: 5, Timestamp: 1727304374935
Produced the following data:
    Key: id_1, Value: Message 1 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 3, Offset: 6, Timestamp: 1727304375017
Produced the following data:
    Key: id_2, Value: Message 2 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 1, Offset: 22, Timestamp: 1727304375041
Produced the following data:
    Key: id_3, Value: Message 3 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 3, Offset: 7, Timestamp: 1727304375068
Produced the following data:
    Key: id_4, Value: Message 4 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 2, Offset: 3, Timestamp: 1727304375082
Produced the following data:
    Key: id_5, Value: Message 5 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 3, Offset: 8, Timestamp: 1727304375096
Produced the following data:
    Key: id_6, Value: Message 6 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 2, Offset: 4, Timestamp: 1727304375111
Produced the following data:
    Key: id_7, Value: Message 7 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 3, Offset: 9, Timestamp: 1727304375131
Produced the following data:
    Key: id_8, Value: Message 8 sent
Successfully received the following metadata:
    Topic: topic-1, Partition: 1, Offset: 23, Timestamp: 1727304375145
Produced the following data:
    Key: id_9, Value: Message 9 sent
```

On the producer terminal, you can see that it has produced 2 messages into partition 1, 3 messages into partition 2 and 5 messages into partition 3.

```
C:\ Command Prompt - java -cp D:\ProgramFiles\Kafka\libs\*, Consumer
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>d:

D:\>cd D:\Projects\Java\Kafka

D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%\libs\*;. Consumer
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Topic: topic-1, Partition: 1, Offset: 22, Key: id_2, Value: Message 2 sent
Topic: topic-1, Partition: 1, Offset: 23, Key: id_8, Value: Message 8 sent

C:\ Command Prompt - java -cp D:\ProgramFiles\Kafka\libs\*, Consumer
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>d:

D:\>cd D:\Projects\Java\Kafka

D:\Projects\Java\Kafka>java -cp %KAFKA_HOME%\libs\*;. Consumer
log4j:WARN No appenders could be found for logger (org.apache.kafka.clients.consumer.ConsumerConfig).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Topic: topic-1, Partition: 3, Offset: 5, Key: id_0, Value: Message 0 sent
Topic: topic-1, Partition: 3, Offset: 6, Key: id_1, Value: Message 1 sent
Topic: topic-1, Partition: 3, Offset: 7, Key: id_3, Value: Message 3 sent
Topic: topic-1, Partition: 2, Offset: 3, Key: id_4, Value: Message 4 sent
Topic: topic-1, Partition: 3, Offset: 8, Key: id_5, Value: Message 5 sent
Topic: topic-1, Partition: 2, Offset: 4, Key: id_6, Value: Message 6 sent
Topic: topic-1, Partition: 3, Offset: 9, Key: id_7, Value: Message 7 sent
Topic: topic-1, Partition: 2, Offset: 5, Key: id_9, Value: Message 9 sent
```

On the first consumer terminal, you can see that it read 2 messages from partition 1 and on the second consumer terminal, you can see that it read 8 messages from partitions 2 and 3.

7. Spark Streaming with Kafka:

Now, we will create a simple application in Java using Spark and integrate with Kafka to consume messages from a topic and display the frequency of words in messages.

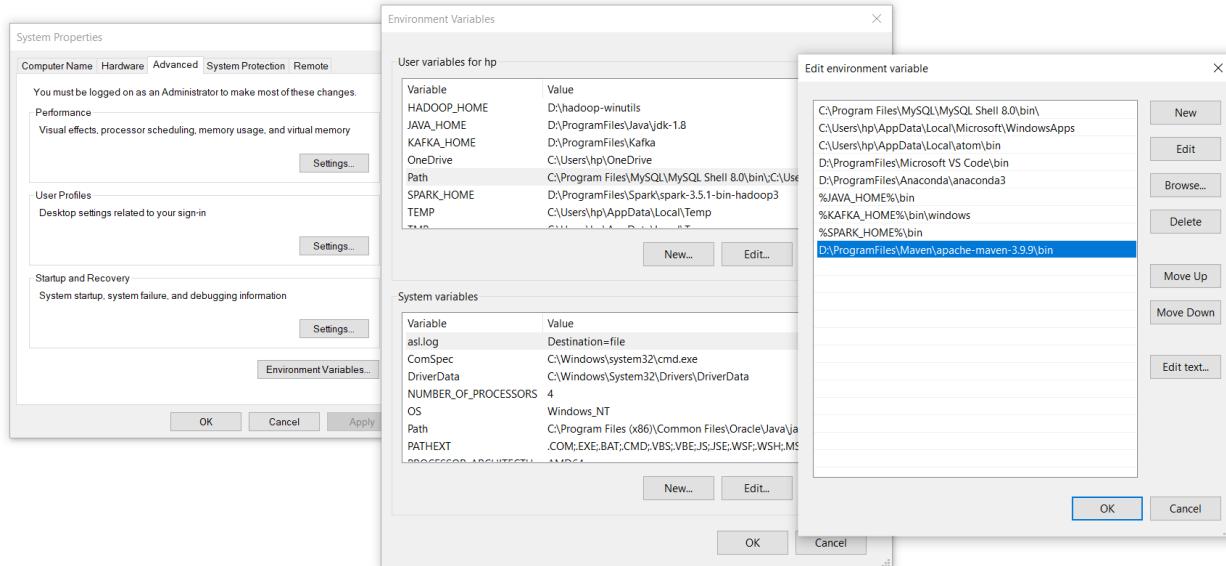
If you do not have Spark installed already, I would suggest you to install it by referring [these steps](#).

To create this project, we need an **IDE tool** such as VS Code, a **build tool** such as Apache Maven and **JDK 1.8** or higher.

Apache Maven is a project management and build tool which is designed based on the concept of project object model (POM). Maven provides developers to manage builds, documentation, reporting, dependencies, SCMs, releases, repositories, distribution, mailing list etc.

Three important terminologies that we should be aware of while using Maven is **groupId** (*uniquely identify the project or application group*), **artifactId** (*name of the project file to define the artifact's location within the repository*) and **Version** (*version number of the project or application created by user*).

If you do not have Maven installed already, then install the latest `apache-maven-x.x.x-bin.zip` file from the [official Apache Maven website](#) and set the maven install bin location in the PATH environment variable as shown below:



7.1. Verify Spark Version:

It is important to know the installed spark version so that the respective version of Spark Streaming API can be used to create our application.

Open **Command Prompt** and run the following command:

```
spark-shell --version
```

You can see that **Spark version 3.5.1** with **Scala version 2.12.18** was installed.

7.2. Start Spark Services:

Next, start the Spark Master and Worker services to access the Spark cluster.

Open a new **Command Prompt** and run the following command to start the Spark Master service:

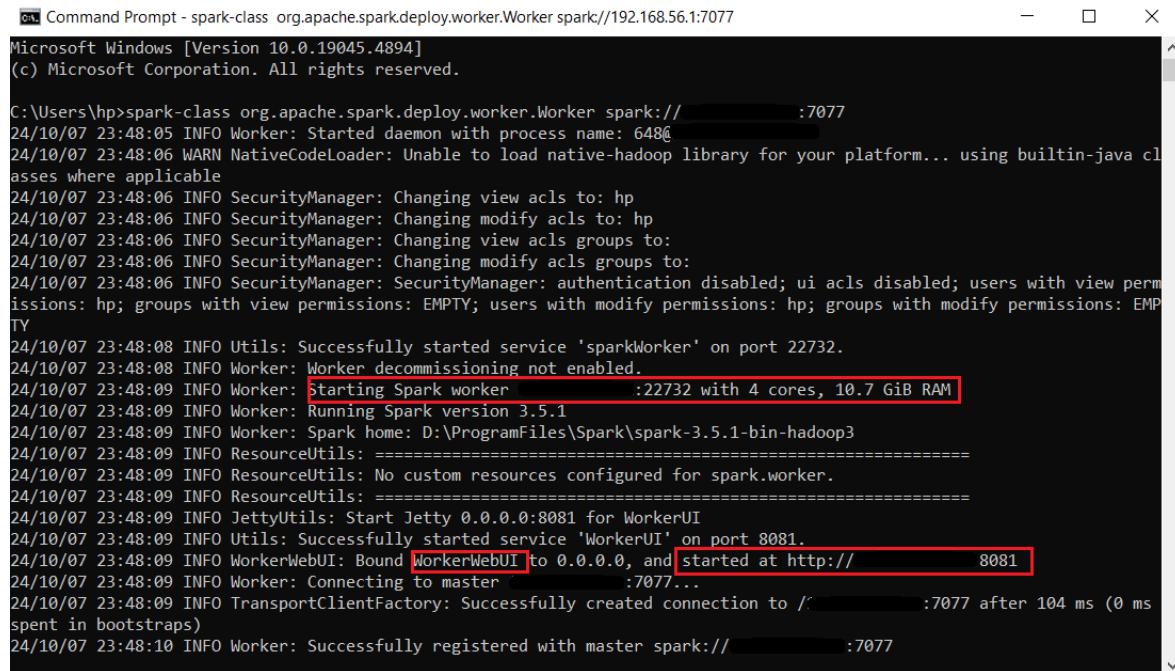
```
spark-class org.apache.spark.deploy.master.Master
```

```
C:\Users\hp>spark-class org.apache.spark.deploy.master.Master
24/10/07 23:43:03 INFO Master: Started daemon with process name: 7072@[REDACTED]
24/10/07 23:43:04 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/10/07 23:43:04 INFO SecurityManager: Changing view acls to: hp
24/10/07 23:43:04 INFO SecurityManager: Changing modify acls to: hp
24/10/07 23:43:04 INFO SecurityManager: Changing view acls groups to:
24/10/07 23:43:04 INFO SecurityManager: Changing modify acls groups to:
24/10/07 23:43:04 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: hp; groups with view permissions: EMPTY; users with modify permissions: hp; groups with modify permissions: EMPTY
24/10/07 23:43:06 INFO Utils: Successfully started service 'sparkMaster' on port 7077.
24/10/07 23:43:06 INFO Master: Starting Spark master at spark://[REDACTED]:7077
24/10/07 23:43:06 INFO Master: Running Spark version 3.5.1
24/10/07 23:43:06 INFO JettyUtils: Start Jetty 0.0.0.0:8080 for MasterUI
24/10/07 23:43:07 INFO Utils: Successfully started service 'MasterUI' on port 8080.
24/10/07 23:43:07 INFO MasterWebUI: Bound MasterWebUI to 0.0.0.0, and started at http://[REDACTED]:8080
24/10/07 23:43:07 INFO Master: I have been elected leader! New state: ALIVE
```

Here, we can see that Spark master is started at <spark://<ipaddress>:7077> i.e listening on default port 7077 and **MasterWebUI** is started at <http://localhost:8080/> by default.

Now, start the Spark Worker service using the below command in a new **Command Prompt**
(Make sure that you replace <ipaddress> with IP address mentioned in your Spark Master URL)

```
spark-class org.apache.spark.deploy.worker.Worker  
spark://<ipaddress>:7077
```



```
Command Prompt - spark-class org.apache.spark.deploy.worker.Worker spark://192.168.56.1:7077
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

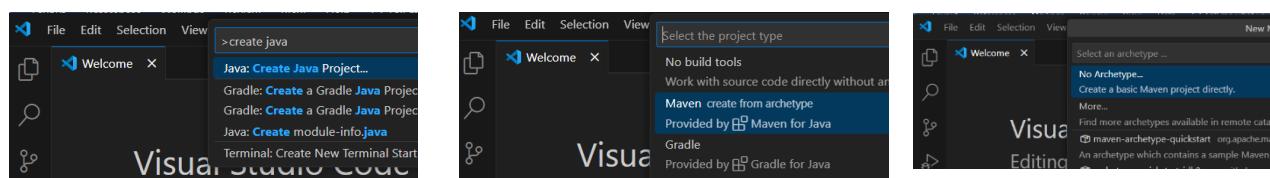
C:\Users\hp>spark-class org.apache.spark.deploy.worker.Worker spark://<ipaddress>:7077
24/10/07 23:48:05 INFO Worker: Started daemon with process name: 648@<ipaddress>
24/10/07 23:48:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/10/07 23:48:06 INFO SecurityManager: Changing view acls to: hp
24/10/07 23:48:06 INFO SecurityManager: Changing modify acls to: hp
24/10/07 23:48:06 INFO SecurityManager: Changing view acls groups to:
24/10/07 23:48:06 INFO SecurityManager: Changing modify acls groups to:
24/10/07 23:48:06 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: hp; groups with view permissions: EMPTY; users with modify permissions: hp; groups with modify permissions: EMPTY
24/10/07 23:48:08 INFO Utils: Successfully started service 'sparkWorker' on port 22732.
24/10/07 23:48:08 INFO Worker: Worker decommissioning not enabled.
24/10/07 23:48:09 INFO Worker: Starting Spark worker :22732 with 4 cores, 10.7 GiB RAM
24/10/07 23:48:09 INFO Worker: Running Spark version 3.5.1
24/10/07 23:48:09 INFO Worker: Spark home: D:\ProgramFiles\Spark\spark-3.5.1-bin-hadoop3
24/10/07 23:48:09 INFO ResourceUtils: -----
24/10/07 23:48:09 INFO ResourceUtils: No custom resources configured for spark.worker.
24/10/07 23:48:09 INFO ResourceUtils: -----
24/10/07 23:48:09 INFO JettyUtils: Start Jetty 0.0.0.0:8081 for WorkerUI
24/10/07 23:48:09 INFO Utils: Successfully started service 'WorkerUI' on port 8081.
24/10/07 23:48:09 INFO WorkerWebUI: Bound WorkerWebUI to 0.0.0.0, and started at http://<ipaddress>:8081
24/10/07 23:48:09 INFO Worker: Connecting to master :7077...
24/10/07 23:48:09 INFO TransportClientFactory: Successfully created connection to />:7077 after 104 ms (0 ms spent in bootstraps)
24/10/07 23:48:10 INFO Worker: Successfully registered with master spark://<ipaddress>:7077
```

Here, we can see that Spark worker is started at <ipaddress>:22732 (on a random port) with available number of cores and RAM running on the machine. **WorkerWebUI** is started at <http://localhost:8081/> by default.

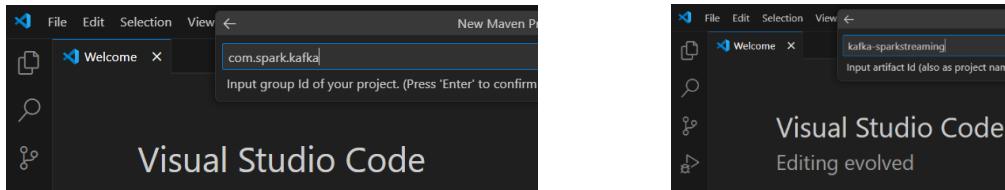
7.3. Create Maven Java Project:

Now, let us create a Maven java project in VS Code application.

In **VS Code**, open **Command Palette** (in **View** menu or press **Ctrl + Shift + P**) where you can type **create java** and choose for **Java: Create Java Project** option. Then choose **Maven** build tool and select **No Archetype** to create a basic maven project.

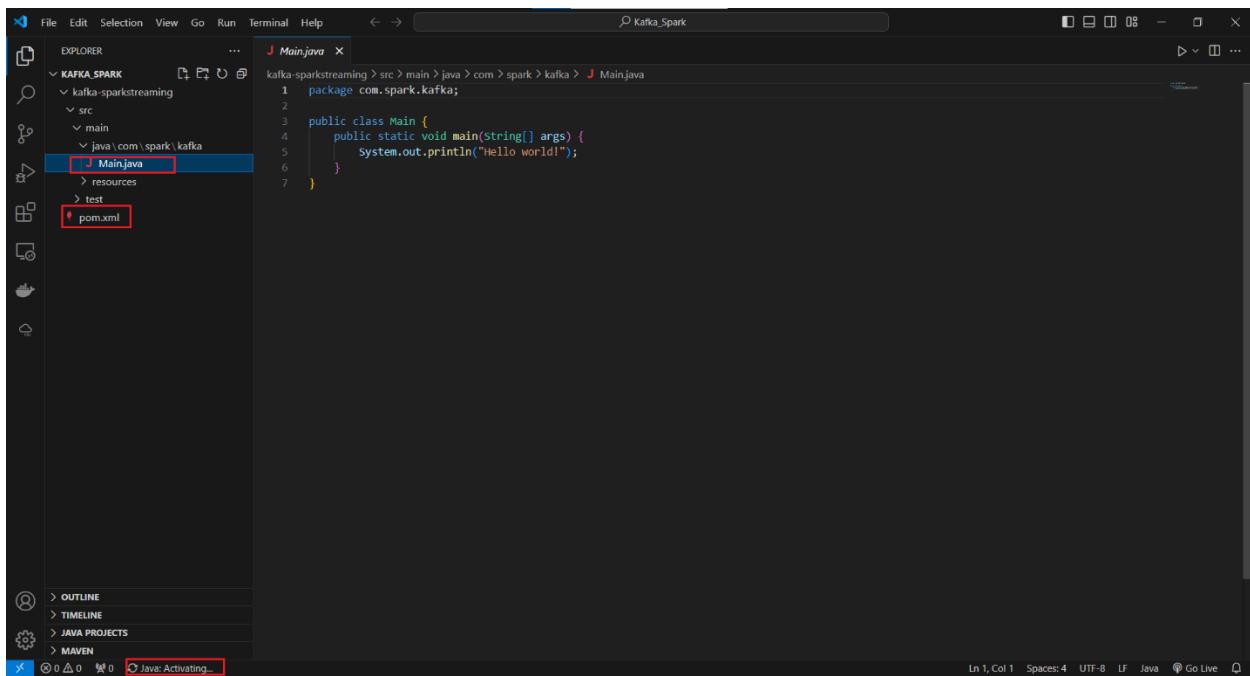


Then it asks you to enter group id where you can enter com.spark.kafka following that it asks you to enter the artifact id where you can enter kafka-sparkstreaming.



Then, it asks you to choose the destination location where you want this project to be created (*I am choosing D:\Projects\Java\Kafka_Spark as my location*) and open that destination folder.

When the destination folder is opened, you can see that java extension got activated and a Main.java file is available by default under kafka-sparkstreaming -> src -> main -> java\com\spark\kafka folder and a pom.xml file is created under kafka-streaming folder.



Once the Java extension is ready, click **Run** link in Main.java file to see the output of java program on the **Terminal**.

```

    package com.spark.kafka;
    public class Main {
        public static void main(String[] args) {
            System.out.println("Hello world!");
        }
    }

```

PS D:\Projects\Java\Kafka_Spark> & 'C:\Users\hp\.vscode\extensions\redhat.java-1.35.1-win32-x64\jre\17.0.12-win32-x86_64\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'D:\Projects\Java\Kafka_Spark\kafka-sparkstreaming\target\classes' 'com.spark.kafka.Main'
Hello world!
PS D:\Projects\Java\Kafka_Spark>

Now that our sample Java code is working, we will write java program to read data from Kafka topic and process through Spark. The integration between Kafka and Spark can be established using the most popular spark libraries `spark-streaming` and `spark-streaming-kafka-0-10`.

Go to `pom.xml` under `kafka-streaming` folder in VS Code and add dependencies to pull `spark-streaming` and `spark-streaming-kafka-0-10` from Maven repository. We should also add `maven-jar-plugin` to create our application jar file. Also, make sure `maven.compiler.source` and `maven.compiler.target` properties are set to **1.8** version in pom file since we are using JDK 1.8 compiler.

The overall `pom.xml` should look like:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.spark.kafka</groupId>
    <artifactId>kafka-sparkstreaming</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    
```

```

</properties>

<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-
streaming -->
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming_2.12</artifactId>
        <version>3.5.1</version>
        <scope>provided</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-
streaming-kafka-0-10 -->
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-streaming-kafka-0-10_2.12</artifactId>
        <version>3.5.1</version>
    </dependency>
</dependencies>
<build>
    <pluginManagement>
        <plugins>
            <plugin>
                <artifactId>maven-jar-plugin</artifactId>
                <version>3.0.2</version>
                <configuration>
                    <archive>
                        <manifest>
                            <addClasspath>true</addClasspath>
                            <classpathPrefix>lib/</classpathPrefix>
<mainClass>com.spark.kafka.WordCount</mainClass>
                            </manifest>
                        </archive>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>

```

Note that I am using spark-streaming_2.12 and spark-streaming-kafka-0-10_2.12 dependencies of Spark 3.5.1 in the above POM since I have **Spark 3.5.1 with Scala 2.12** version installed in my machine. If you have different Spark version installed, then update the version accordingly in pom.xml file.

Once you save the POM file, it asks you to sync the Java class path and click **Yes** to confirm.

A screenshot of the Visual Studio Code interface. The left sidebar shows a project structure under 'KAFKA_SPARK' with 'src' containing 'main' and 'resources', and 'target' containing 'pom.xml'. The main editor window displays the contents of 'pom.xml', which includes dependencies for 'org.apache.spark' and 'spark-streaming-kafka-0-10_2.12'. The terminal at the bottom shows the command 'java -cp' being run, followed by the output 'Hello world!'. A tooltip in the bottom right corner says 'A build file was modified. Do you want to synchronize the Java classpath/configuration?' with options 'Yes', 'Always', and 'Never'. The status bar at the bottom indicates 'Ln 27, Col 1' and other file details.

Before proceeding further, make sure Kafka services are up and running. If not running already, open 4 **Command Prompts** in **Administrator** mode and run the following commands to start Zookeeper first and then 3 Kafka brokers:

```
zookeeper-server-start.bat %KAFKA_HOME%\config\zookeeper.properties
```

```
kafka-server-start.bat %KAFKA_HOME%\config\server.properties
```

```
kafka-server-start.bat %KAFKA_HOME%\config\server-one.properties
```

```
kafka-server-start.bat %KAFKA_HOME%\config\server-two.properties
```

7.4. Create Spark Application:

Now, we create a simple Spark application that consumes live streaming data from a Kafka topic and displays count of each word in the consumed messages.

Apache Spark Streaming API enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Using this API, we can ingest live data from many sources such as Kafka, Twitter, Flume etc., process it using complex algorithms and sync the processed data into many systems such as file systems, databases, live dashboards etc.

We begin with **SparkConf** API that allows us to set configuration properties as key-value pairs for a Spark application.

- First, we should create a `SparkConf` object and set our application name using `setAppName()` method. This helps us to track our application status on the Spark Master. `SparkConf` API provides other methods such as `setMaster()` to set Spark master URL, `set()`, `get()`, etc. To know more details of Spark properties, read [Apache Spark Properties Documentation](#).
- Next, create a `JavaStreamingContext` object which is the main entry point for Spark Streaming functionality. While creating this object, we need to specify `SparkConf` and `Duration` parameters. This duration tells Spark for how much time it has to fetch the incoming data so that Spark divides the incoming live data into batches of given duration, processes it and generates the final result in batches. For example, if the given Duration is 30 seconds, then it would fetch data from source for every 30 seconds.

Note: `JavaStreamingContext` is a Java friendly version of `StreamingContext`.

- Next, create `kafkaParams` map object and set configuration properties for Kafka as key-value pairs. We need to specify properties such as `bootstrap.servers`, `key.deserializer`, `value.deserializer`, `group.id`, `auto.offset.reset`, `enable.auto.commit` for our Kafka cluster.
- Now, we will use `KafkaUtils` API to connect to Kafka cluster through Spark Streaming. You can find this API in `spark-streaming-kafka-0-10` library. `KafkaUtils` API has `CreateDirectStream()` method which is used to create an input stream that pulls messages from Kafka topics. This method has different signatures with different types of arguments but we use a signature of passing `JavaStreamingContext`, `LocationStrategy` and `ConsumerStrategy` as parameters to the method and it returns `ConsumerRecord` as `JavaInputDStream`.
 - `LocationStrategy` is a way of distributing processing of Kafka input stream of messages across Spark executors to achieve optimum performance. The most preferred `LocationStrategy` is `PreferConsistent` which distributes Kafka partitions evenly across available Spark executions. When your Spark executors are located on the same hosts as your Kafka brokers, then choose `PreferBrokers` strategy. When you notice a significant skew in load or load is uneven across partitions, then use `PreferFixed` strategy which allows to map specific topic partition to specific host. Note that `PreferConsistent`,

- `PreferBrokers`, `PreferFixed` are available as methods in `LocationStrategies` factory object.
- `ConsumerStrategy` allows users how to configure Kafka consumers in Spark Streaming. Spark provides three consumer strategies - `Subscribe` which allows to subscribe to a fixed collection of topics, `SubscribePattern` which allows to use regex to specify topics of interest and `Assign` which allows to specify a fixed collection of topic partitions. Note that `ConsumerStrategies` factory object provides `Subscribe`, `SubscribePattern` and `Assign` as methods to use. We will use `ConsumerStrategies.Subscribe()` method which requires us to pass `topics` collection and `kafkaParams` arguments.
 - Next, we perform a series of operations on `JavaInputDStream` to obtain word frequencies in the messages.
 - Using `map()` function, call `value()` method of `JavaInputDStream ConsumerRecord` to retrieve the actual message in a Kafka record (*since Kafka topic stores messages in key-value pairs*) and return `JavaDStream` messages.
 - Using `flatMap()` function, split each message line in the above `JavaDStream` by a space which returns an array of `JavaDStream` words.
 - Using `mapToPair()` function, map the each word in the above `JavaDStream` to a tuple containing a word and 1 which returns `JavaPairDStream` object.
 - Finally, apply the `reduceByKey()` function on the above `JavaPairDStream` using lambda expression that counts by each word and returns `JavaPairDStream` object.
 - The processed `JavaPairDStream` contains collection of elements as `javaPairRDDs` in batches. We can iterate over this `JavaPairDstream` to display each word and its number of occurrences.
 - Since this is a streaming application, we need to keep this running by calling `start()` and `awaitTermination()` methods of `JavaStreamingContext`.

In VS Code, create a new file named `WordCount.java` and enter the following code:

```
package com.spark.kafka;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.consumer.ConsumerRecord;
```

```

import org.apache.kafka.common.serialization.StringDeserializer;
import org.apache.spark.SparkConf;
import org.apache.spark.streaming.Durations;
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaInputDStream;
import org.apache.spark.streaming.api.java.JavaPairDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import org.apache.spark.streaming.kafka010.ConsumerStrategies;
import org.apache.spark.streaming.kafka010.KafkaUtils;
import org.apache.spark.streaming.kafka010.LocationStrategies;

import scala.Tuple2;

@SuppressWarnings("deprecation")
public class WordCount {
    public static void main(String[] args) throws InterruptedException {
        // Create instance to access Spark streaming
        SparkConf sparkConf = new SparkConf();
        sparkConf.setAppName("Kafka Spark Streaming");

        // Create map object to set kafka parameters
        Map<String, Object> kafkaParams = new HashMap<>();
        // Assign bootstrap servers, key and value serializers, group id and
        other config values
        kafkaParams.put("bootstrap.servers", "localhost:9092");
        kafkaParams.put("key.deserializer", StringDeserializer.class);
        kafkaParams.put("value.deserializer", StringDeserializer.class);
        kafkaParams.put("group.id", "word-count-streaming");
        kafkaParams.put("auto.offset.reset", "earliest");
        kafkaParams.put("enable.auto.commit", false);

        // Create collection object and assign list of existing Kafka topics
        // to read
        Collection<String> topicList = Arrays.asList("topic-1");

        // Create java streaming context using spark config with duration of
        // 2 seconds
        JavaStreamingContext streamingContext = new
        JavaStreamingContext(sparkConf, Durations.seconds(30));

        // Get consumer record stream using KafkaUtils API with streaming
        // context and subscribed to given Kafka topics
        JavaInputDStream<ConsumerRecord<String, String>> consumerRecordStream =
            KafkaUtils.createDirectStream(streamingContext,
        LocationStrategies.PreferConsistent(),
        ConsumerStrategies.Subscribe(topicList, kafkaParams));

        // Take value from consumer record stream which has actual data
        JavaDStream<String> messages = consumerRecordStream.map(record ->
        record.value());

        // Split each record by a space and map each word to 1 and apply
        // reduceByKey() to get count of words in a stream
        JavaDStream<String> words = messages.flatMap(line ->
        Arrays.asList(line.split("\\s+")).iterator());
    }
}

```

```

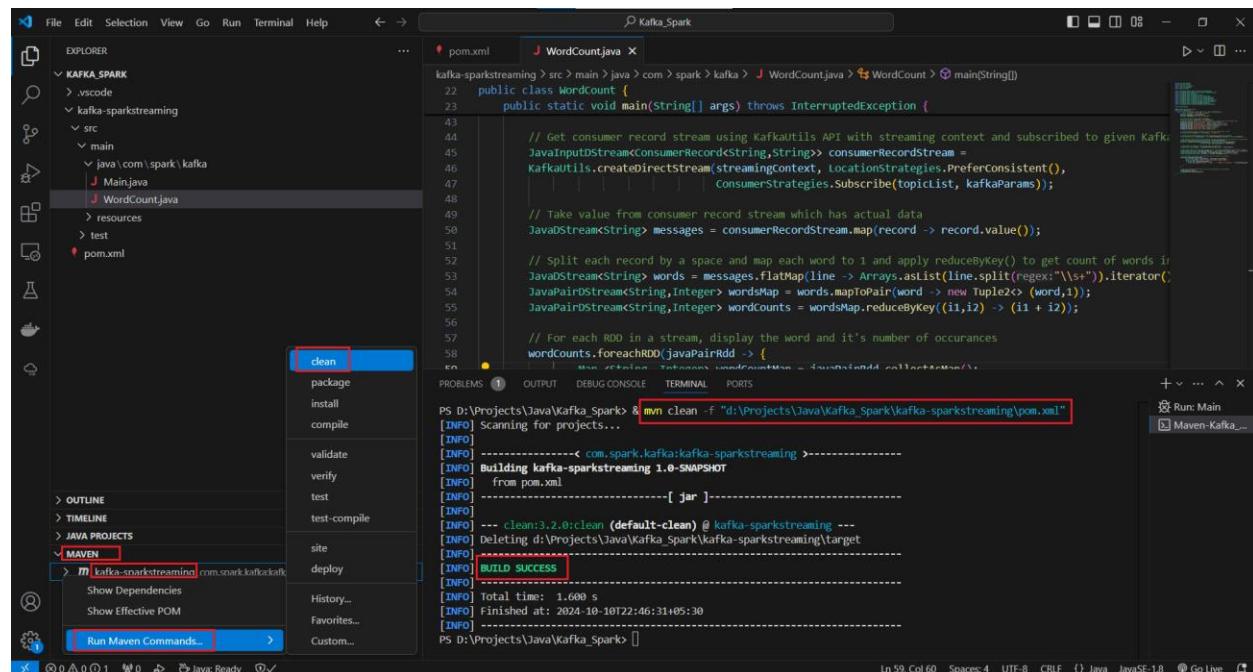
        JavaPairDStream<String, Integer> wordsMap = words.mapToPair(word ->
new Tuple2<> (word,1));
        JavaPairDStream<String, Integer> wordCounts =
wordsMap.reduceByKey((i1,i2) -> (i1 + i2));

        // For each RDD in a stream, display the word and it's number of
occurrences
        wordCounts.foreachRDD(javaPairRdd -> {
            Map <String, Integer> wordCountMap =
javaPairRdd.collectAsMap();
            System.out.println("Count of words from latest consumer
read:");
            for(String key: wordCountMap.keySet()) {
                System.out.println("\t Word: " + key + ", " + "Number of
occurrences: " + wordCountMap.get(key));
            }
        });
    }

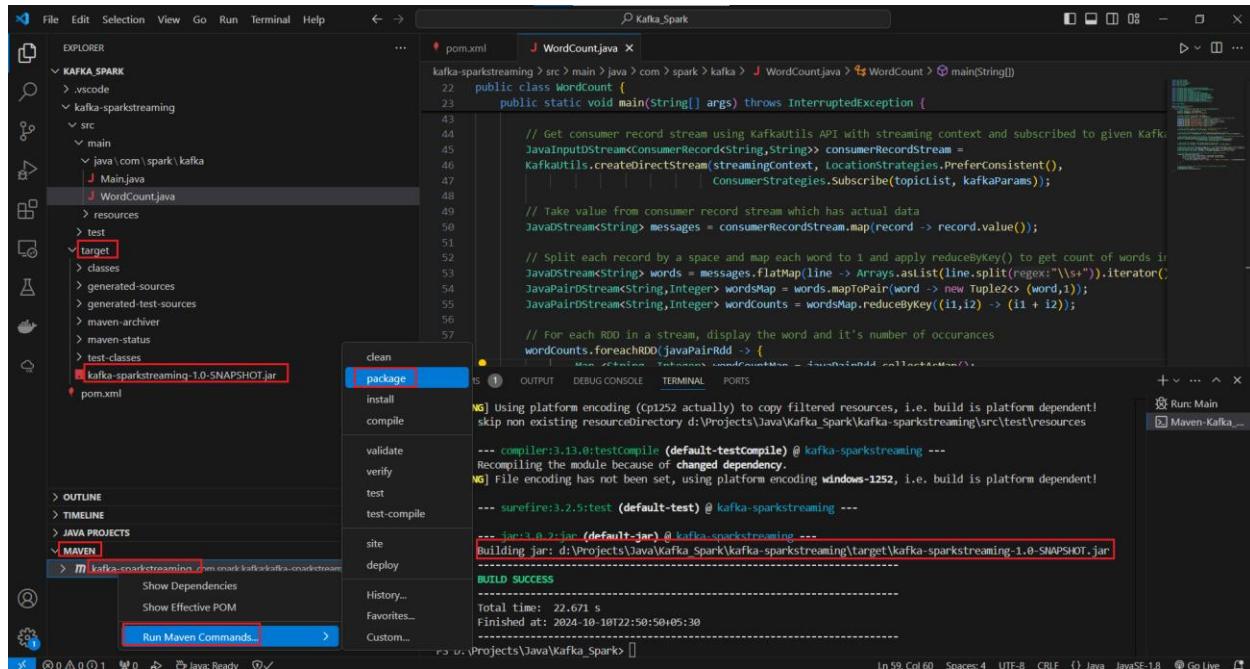
    // Start the streaming process and wait until the application is
terminated
    streamingContext.start();
    streamingContext.awaitTermination();
}
}

```

After the above code is entered, expand **Maven**, right-click on **kafka-sparkstreaming** artifact and select **Run Maven Commands** and choose **clean**. This executes mvn clean command on the terminal and displays **BUILD SUCCESS** message.



Next, right-click on `kafka-sparkstreaming` artifact under **Maven** section and select **Run Maven Commands** and choose **package**. This executes `mvn package` command on the terminal and builds a jar file called `kafka-sparkstreaming-1.0-SNAPSHOT.jar` which you can see under target folder in `kafka-sparkstreaming` folder.



7.5. Execute Spark Application:

Now that our application jar file is ready, we need to deploy our application using `spark-submit` script that comes with Spark installation.

In VS Code, click on **+** button on **Terminal** window and select **Command Prompt** where you need to run the following command to deploy the application in client mode on Spark cluster. In the below command, make sure to replace `spark://<ip_address>:7077` with your Spark Master URL that is displayed on Spark Master UI at <http://localhost:8080/>

```
%SPARK_HOME%\bin\spark-submit.cmd --master spark://<ip_address>:7077
--packages org.apache.kafka:kafka-clients:3.8.0,org.apache.spark:spark-streaming-kafka-0-10_2.12:3.5.1
--class com.spark.kafka.WordCount kafka-sparkstreaming\target\kafka-sparkstreaming-1.0-SNAPSHOT.jar
```

After executing the above command, you can see INFO logging on the console which says it is downloading necessary packages, connecting to spark cluster, integrating with Kafka cluster to

read from Kafka topic `topic-1`. Then it displays the frequency of words in the messages as shown below:

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure under `KAFKA_SPARK`, including `pom.xml`, `WordCount.java`, and `Main.java`.
- CODE EDITOR:** Displays the `WordCount.java` file with code related to reading from Kafka and printing word counts.
- TERMINAL:** Shows the command prompt where the application was run, displaying log output. A red box highlights the output of the application itself, which lists word counts from the Kafka topic.

```

24/10/10 23:42:36 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on 192.168.56.1:4330
366.3 MB
24/10/10 23:42:36 INFO MapOutputTrackerMasterEndpoint: Asked to send map output locations for shu
:4318
24/10/10 23:42:36 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 855 ms on 192.16
(/2)
24/10/10 23:42:36 INFO TaskSetManager: Finished task 1.0 in stage 1.0 (TID 2) in 858 ms on 192.16
(/2)
24/10/10 23:42:36 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, fr
24/10/10 23:42:36 INFO DAGScheduler: ResultStage 1 (collectAsMap at WordCount.java:59) finished i
24/10/10 23:42:36 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombi
24/10/10 23:42:36 INFO TaskSchedulerImpl: Killing all running tasks in stage 1: Stage finished
24/10/10 23:42:36 INFO DAGScheduler: Job 0 finished: collectAsMap at WordCount.java:59, took 6.549948 s
Count of words from latest consumer read:
Word: 2, Number of occurrences: 1
Word: Message, Number of occurrences: 10
Word: 5, Number of occurrences: 1
Word: 8, Number of occurrences: 1
Word: sent, Number of occurrences: 10
Word: 7, Number of occurrences: 1
Word: 1, Number of occurrences: 1
Word: 4, Number of occurrences: 1
Word: 6, Number of occurrences: 1
Word: 9, Number of occurrences: 1
Word: 0, Number of occurrences: 1
Word: 3, Number of occurrences: 1
24/10/10 23:42:36 INFO JobScheduler: Finished job streaming job 1728583950000 ms.0 from job set of time 1728583950000 m
s
24/10/10 23:42:36 INFO DAGScheduler: Total delay: 6.958 s for time 1728583950000 ms (execution: 6.692 s)
24/10/10 23:42:36 INFO ReceivedBlockTracker: Deleting batches:
24/10/10 23:42:36 INFO InputInfoTracker: remove old batch metadata:

```

As this is a streaming application, it keeps reading data from Kafka topic at 30 seconds batch interval (as specified in our application) and displays frequency of words for every batch until the application is terminated (*You can enter **Ctrl + C** to terminate this application manually*).

Note: By default, Spark displays **INFO** logging on the console.

Follow these steps to turn off the unnecessary **INFO** logging:

- Go to `%SPARK_HOME%\conf` folder, take the copy of `log4j2.properties.template` file and rename as `log4j2-spark.properties`.
- Open `log4j2-spark.properties` and change the `rootLogger.level` value from `info` to `warn`.
- On the VS Code terminal, execute the below `spark-submit` command. Make sure to replace `spark://<ip_address>:7077` with your Spark Master URL displayed on Spark Master UI and also put the absolute path of your `SPARK_HOME` in `configurationFile` path

```
%SPARK_HOME%\bin\spark-submit.cmd --master spark://<ip_address>:7077
--driver-java-options "
Dlog4j.configurationFile=file:///D:/ProgramFiles/Spark/spark-3.5.1-
bin-hadoop3/conf/log4j2-spark.properties" --packages
org.apache.kafka:kafka-clients:3.8.0,org.apache.spark:spark-
```

```
streaming-kafka-0-10_2.12:3.5.1 --class com.spark.kafka.WordCount
kafka-sparkstreaming\target\kafka-sparkstreaming-1.0-SNAPSHOT.jar
```

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows the project structure under "KAFKA_SPARK".
- WordCount.java:** The active file in the editor.
- PROBLEMS:** Shows 1 error.
- OUTPUT:** Displays the application logs.
- TERMINAL:** Shows the command used to run the application.
- PORTS:** Shows open ports.
- Run:** Shows "Main" selected.
- Java:** Shows "Maven-Kafka..." selected.

```
File Edit Selection View Go Run Terminal Help ← → Kafka_Spark
EXPLORER ... WordCount.java
KAFKA_SPARK
> .vscode
< kafka-sparkstreaming
  < src
    < main
      < java \ com \ spark \ kafka
        Main.java
        WordCount.java
      pom.xml
    resources
    test
  target
  classes
  generated-sources
  generated-test-sources
  maven-archiver
  maven-status
  test-classes
  kafka-sparkstreaming-1.0-SNAPSHOT.jar
pom.xml

OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
m kafka-sparkstreaming com.spark.kafka.kafka-sparkstreaming

WordCount.java
22 public class WordCount {
23     public static void main(String[] args) throws InterruptedException {
24         KafkaParams.put("enable.auto.commit", "false");
25         // Create collection object and assign list of existing Kafka topics to read
26         Collection<String> topicList = Arrays.asList(...);
27         ...
28     }
29 }

PROBLEMS 1
OUTPUT
TERMINAL
PORTS
RUN: Main
Maven-Kafka...
java

22 public class WordCount {
23     public static void main(String[] args) throws InterruptedException {
24         KafkaParams.put("enable.auto.commit", "false");
25         // Create collection object and assign list of existing Kafka topics to read
26         Collection<String> topicList = Arrays.asList(...);
27         ...
28     }
29 }

:: retrieving :: org.apache.spark#spark-submit-parent-c3539547-4680-4445-b7a1-ea70229337ee
conf: [default]
0 artifacts copied, 11 already retrieved (0kB/s)
24/10/11 00:00:45 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/10/11 00:00:50 WARN Kafk-utils: overriding enable.auto.commit to false for executor
24/10/11 00:00:50 WARN Kafk-utils: overriding auto.offset.reset to none for executor
24/10/11 00:00:50 WARN Kafk-utils: overriding executor group.id to spark-executor-word-count-streaming
24/10/11 00:00:50 WARN Kafk-utils: overriding receive.buffer.bytes to 65536 see KAFKA-3135
Count of words from latest consumer read:
Word: 2, Number of occurrences: 1
Word: Message, Number of occurrences: 10
Word: 5, Number of occurrences: 1
Word: 8, Number of occurrences: 1
Word: sent, Number of occurrences: 10
Word: 7, Number of occurrences: 1
Word: 1, Number of occurrences: 1
Word: 4, Number of occurrences: 1
Word: 6, Number of occurrences: 1
Word: 9, Number of occurrences: 1
Word: 0, Number of occurrences: 1
Word: 3, Number of occurrences: 1
Count of words from latest consumer read:
```

Here, you can see that the final output on the console without any other logging.

Congratulations!! You have successfully installed Apache Kafka with single broker and multi-broker configuration and created a streaming application by integrating Kafka with Spark in Windows operating system.