SE424: Distributed Systems	**Recitation 4c**
Semester 1 5787	**17 November 2026**
Lecturer: Michael J. May	**Kinneret College**

Kafka

In this recitation, we'll do some rudimentary experiments with Kafka, a message oriented middleware system in use across the world.

Key web sites You can find out more about Kafka at the project's website: `https://kafka.apache.org` and its documentation site: `https://kafka.apache.org/documentation/` .

The recitation that we're going to go through today is based on the Quickstart tutorial on the Kafka project's web page (`https://kafka.apache.org/quickstart`). You can look at the website as a reference if you need to.

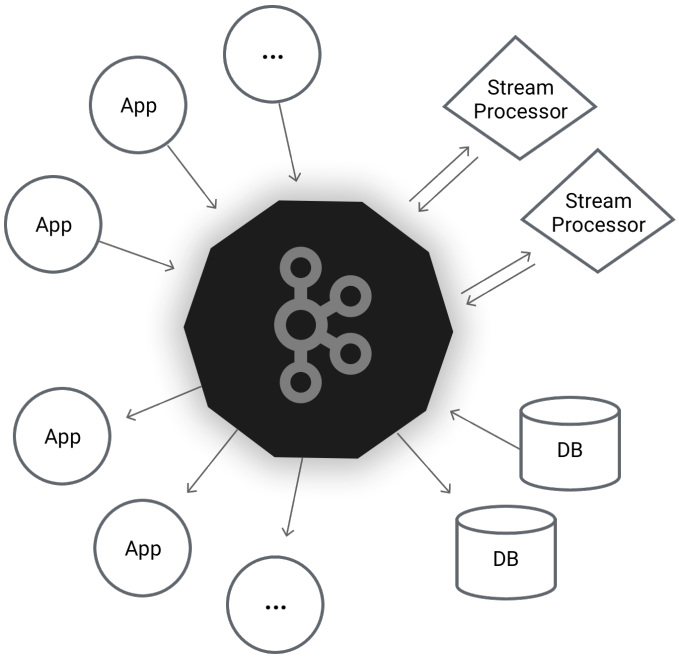


Figure 1: Kafka General Architecture

The general architecture of Kafka is shown in Figure **??**. We will write a couple of apps in recitation today that work with Kafka and use it for sending messages based on topics.

1 Downloading Kafka

We'll start by downloading the Kafka 4.4.0 package. It can be downloaded from:

`https://www.apache.org/dyn/closer.lua/kafka/4.4.0/kafka_2.13-4.4.0.tgz?action=download`

You will need to use Linux for this recitation because Kafka 4 does not work on Windows 11 well (if at all) due to differences in how memory mapped files are handled in Windows and Linux. Ideally, use a Linux computer or a virtual machine with Linux installed. You can use Windows Subsystem for Linux (WSL2) also.

Once you have the code downloaded, unpack it into a directory which is convenient. I'll put it in `/home/mjmay/kafka` which is what I'll be using for the rest of this document.

There are two important directories under the main directory:

- `kafka/bin` which has the binaries you'll need to run Kafka under Linux (`.sh` scripts)
- `kafka/config` which has the configuration files you'll need.

2 Kafka with KRaft

Kafka was reliant on Apache ZooKeeper (`https://zookeeper.apache.org/`) since its inception over 10 years ago. ZooKeeper provides coordination between Kafka brokers (servers), covering many of the basics of consensus and distributed communication. Recent versions of Kafka include the Raft distributed consensus protocol (KRaft) that enables Kafka to operate without ZooKeeper's help. The most recent version of Kafka (4.4.0) removed ZooKeeper entirely, so KRaft is the only consensus protocol supported.

3 Setting up a Kafka Cluster

The first step is to set up a Kafka cluster. We'll use the Kafka storage script to set up a random cluster id. Enter the following command into your

```
KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
```

3.1 Log Directories

Then we need to set up the log directories for the brokers. Kafka's logs are essential to its behavior since brokers use them to keep the message history. We'll use the Kafka storage script again to set up the log storage directory:

```
bin/kafka-storage.sh format --standalone -t $KAFKA_CLUSTER_ID -c config/server.properties
```

4 Starting a Kafka Broker

The next step we'll take is to start a Kafka Broker. The broker (or server) will be connected to the ZooKeeper instance. Right now the connection is rather trivial since we have only one broker. When we add more than one broker, ZooKeeper's job will start to get more interesting.

Run the following command in a new command line window in Linux. (Don't close the previous one with the ZooKeeper inside).

```
bin/kafka-server-start.sh config/server.properties
```

As before, the command must be run from the `/home/mjmay/kafka` directory. The console will start to spit out some log lines to indicate that the server is working.

4.1 Broker Configuration

Let's take a quick look at the broker's configuration file to find some important fields that control how the broker works. The line numbers and values below refer to the `config/server.properties` file as distributed with the standard Kafka distribution.

- Line 19: process.roles=broker,controller

  - This means that the broker will be ready to serve as a regular node (broker) as well as a coordinator for the KRaft cluster (controller). Multiple brokers can be ready to be controllers to ensure reliability in case of failure.

- Line 21: node.id=1
  - This line gives the unique ID for each broker. If you have multiple brokers, you'll need to ensure that each one gets a unique ID.
- Line 37: listeners=PLAINTEXT://:9092,CONTROLLER://:9093
  - The first value is the IP and port the broker listens on. Right now it's set for localhost on port 9092. If you're going to do just localhost experiments, that's fine. If you want to have multiple brokers in a cluster, you'll need to set up the IP addresses and ports for all of them.

    The second value is for the broker coordinator (the one that controls the cluster using KRaft). It's also on local host using port 9093.
- Line 44: advertised.listeners=PLAINTEXT://localhost:9092,CONTROLLER://localhost:9093
  - This field is what the node will tell others its IP address and ports for listening and controlling are. The difference between this and the previous one is that this one is what outsiders think the node's IP and ports are and the previous one is what the broker listens on at the local machine.
  - The two values will be different if the broker is sitting behind a NAT router and you use port forwarding. In that case, the external and internal IP addresses will be different.
- Line 73: log.dirs=/tmp/kraft-combined-logs
  - Where the broker dumps its logs. Kafka is very verbose and within 5 minutes you can end up with 1GB of logs. If you're running multiple brokers on a single computer for some reason, you must ensure that each broker gets its own log directory. Otherwise, bad things will happen.
- Line 78: num.partitions=1
  - Partitions allow a single topic's storage to be broken up across many different servers, perhaps to allow storage of a topic with a huge amount of storage, more than any one server can handle or that need to handle heavy parallel consumption. This flag now says that all topics by default will have one partition.
- Lines 118, 125, 129: log.retention.hours, log.segment.bytes, log.retention.check.interval.ms
  - You'll find that Kafka is very verbose, dumping out GB of logs within minutes if you don't stop it. You can use these fields and others to control how much time the log is stored for and how big each segment is.
  - Keep in mind that if you make the log too small, you'll lose some of Kafka's abilities to recover from server crashes and to provide a "from the beginning" replay of conversations.

5 Create a Test Topic

Once we have the server (broker) and ZooKeeper ready, we can start a topic. The topics we create are the "channels" over which the applications can communicate. Each message that a sender sends is put under a topic and will be routed to all of the consumers who are interested in it. Run the following command from a new command line in Linux.

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

The topic is created with the following parameters:

- The Kafka broker instance that will be in charge managing the leader for the topic (localhost:9092)
- The topic name (quickstart-events)

Once the command above is run, we'll see a success message as below:

```
Created topic quickstart-events
```

If you don't get a success message, check that the Kafka server is still running.

5.1 Checking the topic

You can check that your topic creation worked by running the following command (in Linux). You can run it in the same command prompt that you used for the topic creation command.

```
bin/kafka-topics.sh --list --bootstrap-server localhost:9092
```

You'll see a message showing that the quickstart-events topic is defined.

You can also get more information about the topic using the topic describe command from the command line:

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

You'll see an output that looks like the following:

```
Topic: quickstart-events        TopicId: mLkjWCUxRpWFmJeCs3e4og PartitionCount: 1
    ReplicationFactor: 1        Configs: min.insync.replicas=1,segment.bytes=1073741824
        Topic: quickstart-events        Partition: 0    Leader: 1       Replicas: 1     Isr:
            1       Elr:    LastKnownElr:
```

The parameters you see here give more information about the topic:

- The name of the topic (quickstart-events) and its unique ID (mLkjWCUxRpWFmJeCs3e4og)
- The number of partitions the topic has (1). Kafka can use multiple partitions to divide up a topic's events across multiple brokers.
- The replication factor (1). That refers to the number of copies of each partition. Since there is only 1 here, there is no backup in case of failure.
- The leader, which broker is in charge of partition 0. In this case it's broker with ID 1 (the only one we have). Since it's the leader, all reads and writes go through it.
- The number of in sync replicas (Isr). Since there's only one replica here, it's listed alone.
- Other eligible leader replicas (Elr). Since there are no other replicas, no one else is eligible.
- The last known eligible leader replica. Same as above.

5.2 Build a Consumer

Let's set up a consumer now that will communicate with the Kafka broker and receive messages from the topic "test". Run the following command in Linux (open a new command prompt window):

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

The fields that we're providing are as follows:

- The bootstrap server - the Kafka broker that we're connecting to
- The topic to read from (quickstart-events).
- Where to being reading from - here it's from the beginning. We also can provide other options from where to begin reading from.

Right now we won't see anything, but when we start a producer we'll start to see output appear in the consumer's window.

5.3 Build a Producer

Now that we have a consumer, let's open a producer which will send data on it. Run the following command in Linux (in a new command prompt window):

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

If the command works, we'll be able to start entering in some messages and see them appear after a short time in the consumer's window.

You can try to dump a lot of text into the producer using input redirection. There should be a file called LICENSE in the Kafka directory. Writing something like:

```
bin/kafka-console-producer.sh --topic quickstart-events --bootstrap-server localhost:9092 <
    LICENSE
```

will dump the whole text contents of the file into the producer. They should appear shortly thereafter at the consumer.

5.4 Adding a new Consumer

Let's see what happens when we open a new consumer to the existing system. Run the command for creating a new Consumer in a new CMD window. What output do you see on the screen?

Close the new consumer using Ctrl+C.

5.5 Another new Consumer

Let's start another consumer using the following command (note that I removed the last flag from the previous version):

```
bin/kafka-console-consumer.sh --topic quickstart-events --bootstrap-server localhost:9092
```

What happens?

6 Summary So Far

So far we have built a simple setup with one Kafka broker. The architecture is as shown in Figure **??**.

There are two consumers that get data from the topic called "quickstart-events" via the single Kafka node. Note that the broker is labeled with its `node.id=1` value from its configuration file (`server.properties`).

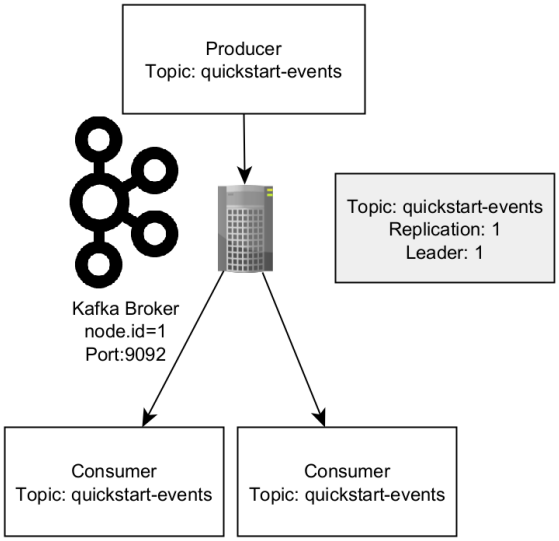


Figure 2: Single Broker Cluster

7 Building a Multibroker Cluster

We built a single broker cluster so far. Let's make our setup more interesting by adding a two more brokers and letting KRaft arrange the communication between them. Since we're going to be doing this at home, you can use localhost for all IP addresses unless you have multiple computers to work with.

First thing to do is to shut down the existing cluster so that we can mess around with the configuration files without a problem. Close the command prompt windows for all of the applications. You can use Ctrl+C to cause the nodes to shut down.

Now, let's assume that you've got three computers available with IP addresses 10.0.0.1, 10.0.0.2, and 10.0.0.3 respectively. If your IP addresses are different or you're using VirtualBox to make a closed NAT network, change the addresses accordingly. We're going to build a topology which looks like Figure **??**.

We're assuming that the original Kafka broker you started was on 10.0.0.2. If it was on a different IP address (like localhost), change the addresses below accordingly.

7.1 Configuration Files

We'll first three different `server.properties` files for the nodes. Make three copies of the configuration files and change them as follows:

7.1.1 Broker at 10.0.0.1 server-1.properties file

Note that we need to add new lines after 25 and 79, so line numbers won't necessarily match your initial file.

```
Line 22: broker.id=1
Line 25: controller.quorum.bootstrap.servers=10.0.0.1:19092,10.0.0.2:19093,10.0.0.3:19094
Line 26: controller.quorum.voters=10.0.0.1:19092,2@10.0.0.2:19093,3@10.0.0.3:19094
Line 38: listeners=PLAINTEXT://10.0.0.1:9092,CONTROLLER://10.0.0.1:19092
Line 45: advertised.listeners=PLAINTEXT://10.0.0.1:9092,CONTROLLER://10.0.0.1:19092
Line 74: log.dirs=/tmp/kraft-logs-1
Line 79: default.replication.factor=3
Line 80: num.partitions=3
Line 81: min.insync.replicas=2
```

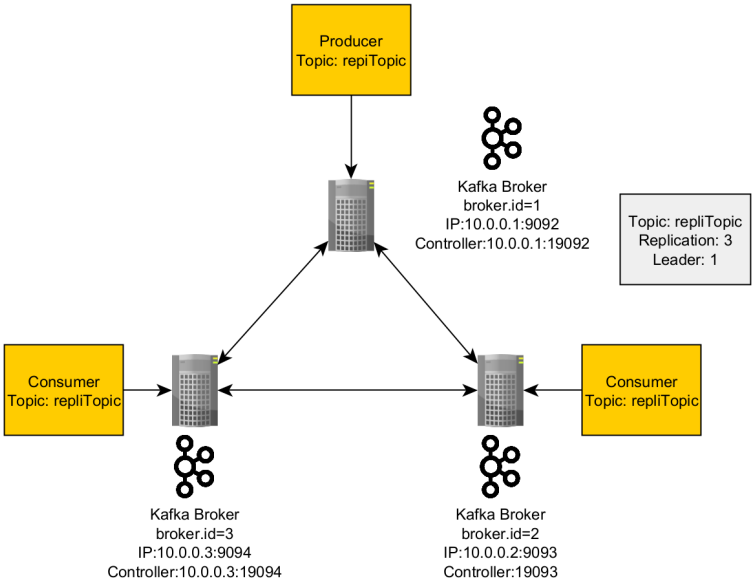


Figure 3: Multi-Broker Cluster

7.1.2 Broker at 10.0.0.2 server-2.properties file

```
Line 22: broker.id=2
Line 25: controller.quorum.bootstrap.servers=10.0.0.1:19092,10.0.0.2:19093,10.0.0.3:19094
Line 26: controller.quorum.voters=10.0.0.1:19092,2@10.0.0.2:19093,3@10.0.0.3:19094
Line 38: listeners=PLAINTEXT://10.0.0.2:9093,CONTROLLER://10.0.0.2:19093
Line 45: advertised.listeners=PLAINTEXT://10.0.0.2:9093,CONTROLLER://10.0.0.2:19093
Line 74: log.dirs=/tmp/kraft-logs-2
Line 79: default.replication.factor=3
Line 80: num.partitions=3
Line 81: min.insync.replicas=2
```

7.1.3 Broker at 10.0.0.3 server.properties file

```
Line 22: broker.id=3
Line 25: controller.quorum.bootstrap.servers=10.0.0.1:19092,10.0.0.2:19093,10.0.0.3:19094
Line 26: controller.quorum.voters=10.0.0.1:19092,2@10.0.0.2:19093,3@10.0.0.3:19094
Line 38: listeners=PLAINTEXT://10.0.0.3:9094,CONTROLLER://10.0.0.3:19094
Line 45: advertised.listeners=PLAINTEXT://10.0.0.3:9094,CONTROLLER://10.0.0.3:19094
Line 74: log.dirs=/tmp/kraft-logs-3
Line 79: default.replication.factor=3
Line 80: num.partitions=3
Line 81: min.insync.replicas=2
```

Single Computer If you're doing this all on a single computer, replace the 10.0.0.1, 10.0.0.2, 10.0.0.3 addresses above with `localhost`.

7.2 Initializing the Storage

The first thing we need to do is create a cluster ID for the broker. If you already made one before in Section **??** above, you can check its value using the command:

```
echo "Cluster ID: $KAFKA_CLUSTER_ID"
```

You should get output that looks like:

```
Cluster ID: XvcofFisS3eDkwD6W1SgJA
```

If you don't see a value for it, you'll need to follow the instructions in Section **??** above.

If you're opening the brokers on different computers (not all using localhost), then you'll need to copy the value down from the computer where you created the value and set it up on other computers using a command like:

```
KAFKA_CLUSTER_ID=XvcofFisS3eDkwD6W1SgJA
```

Once we have the cluster ID ready, we need to initialize the storage directories for the nodes. Run the following commands (assuming you put the server-1, server-2, and server-3 files in the config directory):

```
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c config/server-1.properties
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c config/server-2.properties
bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c config/server-3.properties
```

You should get success messages that look like this:

```
Formatting /tmp/kraft-logs-1 with metadata.version 4.4-IV0.
```

7.3 Starting Up the Cluster

Start up the three brokers using the configuration files you defined above. Ensure they are all up and running correctly. Run the following commands, either on different computers or in different terminals:

```
bin/kafka-server-start.sh config/server-1.properties
bin/kafka-server-start.sh config/server-2.properties
bin/kafka-server-start.sh config/server-3.properties
```

After a minute or so, you'll see that the cluster has found itself and a leader has been chosen.

7.4 Setting up a Replicated Topic

We next will set up a topic with replication factor 3. Enter the following command in a new command line window:

```
bin/kafka-topics.sh --create --topic repliTopic --partitions 3 --replication-factor 3 --
    bootstrap-server 10.0.0.1:9092
```

Remember to chance 10.0.0.2 to the actual address of a broker. If you are doing everything on one computer, use localhost.

Once you have the topic set up, you can test that it works using the following command:

```
bin/kafka-topics.sh --describe --topic repliTopic --bootstrap-server 10.0.0.1:9092
```

You should get an output that looks like:

```
Topic: repliTopic       TopicId: CA9AIIW1R22S6kuaeidqrQ PartitionCount: 3
    ReplicationFactor: 3    Configs: min.insync.replicas=2,segment.bytes=1073741824
        Topic: repliTopic       Partition: 0    Leader: 3       Replicas: 3,1,2 Isr: 3,1,2
                Elr:    LastKnownElr:
        Topic: repliTopic       Partition: 1    Leader: 1       Replicas: 1,2,3 Isr: 1,2,3
                Elr:    LastKnownElr:
        Topic: repliTopic       Partition: 2    Leader: 2       Replicas: 2,3,1 Isr: 2,3,1
                Elr:    LastKnownElr:
```

The fields above are as follows:

leader The node responsible for all reads and writes for the given partition. Each node will be the leader for a randomly selected portion of the partitions.

replicas the list of nodes that replicate the log for this partition regardless of whether they are the leader or currently alive

isr the set of "in-sync" replicas. This is the subset of the replicas list that is currently alive and caught-up to the leader. There should be 3 total here.

In this case since we have three replicas of the topic and three partitions, each broker is leader for one partition and the others are all caught up. If we try to check on our original topic (test), we would get output showing that there is only one replica held by the leader.

7.5 Setting up Producers and Consumers

Let's set up a consumer that's connected to the cluster. Note that we give it three options so it can recover if one fails:

```
bin/kafka-console-consumer.sh --topic repliTopic --from-beginning --bootstrap-server
    10.0.0.1:9092,10.0.0.2:9093,10.0.0.3:9094
```

Then setup a console producer in another window:

```
bin/kafka-console-producer.sh --topic repliTopic --bootstrap-server
    10.0.0.1:9092,10.0.0.2:9093,10.0.0.3:9094
```

Test the two by entering text into the producer. It should appear soon afterwards in the consumer.

7.6 Testing the Replication Resilience

Let's go to the window for broker 3 and close it using Ctrl+C. Give a few seconds and then try the describe command again to see the status. You should see something like this:

```
Topic: repliTopic       TopicId: CA9AIIW1R22S6kuaeidqrQ PartitionCount: 3
    ReplicationFactor: 3    Configs: min.insync.replicas=2,segment.bytes=1073741824
        Topic: repliTopic       Partition: 0    Leader: 1       Replicas: 3,1,2 Isr: 1,2
                   Elr:    LastKnownElr:
        Topic: repliTopic       Partition: 1    Leader: 1       Replicas: 1,2,3 Isr: 1,2
                   Elr:    LastKnownElr:
        Topic: repliTopic       Partition: 2    Leader: 2       Replicas: 2,3,1 Isr: 2,1
                   Elr:    LastKnownElr:
```

Note how there are only 2 in sync replicas and the leader of 3's partition is now someone else.

Continue using the consumer and producer. They should still work.

7.7 Restoring Connectivity

Now restart broker 3 using the above command. Try using the producer and consumer that you had open before. They should not be affected.

Describe the topic again using the above command. You should see something like this:

```
Topic: repliTopic       TopicId: CA9AIIW1R22S6kuaeidqrQ PartitionCount: 3
    ReplicationFactor: 3    Configs: min.insync.replicas=2,segment.bytes=1073741824
        Topic: repliTopic       Partition: 0    Leader: 1       Replicas: 3,1,2 Isr: 1,2,3
                  Elr:    LastKnownElr:
```

```
        Topic: repliTopic       Partition: 1    Leader: 1       Replicas: 1,2,3 Isr: 1,2,3
                Elr:    LastKnownElr:
        Topic: repliTopic       Partition: 2    Leader: 2       Replicas: 2,3,1 Isr: 1,2,3
                Elr:    LastKnownElr:
```

Notice how there are now 3 in sync replicas.

8 Use Kafka Connectors to Send Files

Use the connectors to send text files line by line. Use the connector consumer to see the file arrive correctly and the regular one to see it as JSON. Follow the instructions in Section 6 at: `https://kafka.apache.org/quickstart/#step-6-importexport-your-data-as-streams-of-events-with-kafka-connect`.

9 Implementing with Kafka

If you finished the above tasks, it's time to write a small Java program that uses Kafka as a communication backend. Follow the instructions at: `https://kafka.apache.org/42/streams/tutorial/`.

10 Using Docker Compose for a Multibroker Cluster

Instead of running multiple Kafka brokers on your local machine (`localhost`) or configuring multiple physical computers, you can use Docker and Docker Compose to set up a KRaft cluster. Docker isolates each broker in its own container. By explicitly defining a custom network, we isolate communication between the nodes.

10.1 The Compose File

Create a new directory for your Docker setup and create a file named `compose.yaml` inside it. Copy and paste the following configuration into the file. This file uses the official Apache Kafka image, defines an explicit `kafka-net` bridge network, and automatically uses the generated cluster ID for the storage step:

```
services:
  kafka-1:
    image: apache/kafka:4.4.0
    container_name: kafka-1
    networks:
      - kafka-net
    environment:
      KAFKA_NODE_ID: 1
      KAFKA_PROCESS_ROLES: broker,controller
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka-1:19092,2@kafka-2:19092,3@kafka-3:19092
      KAFKA_LISTENERS: PLAINTEXT://:9092,CONTROLLER://:19092
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      CLUSTER_ID: "XvcofFisS3eDkwD6W1SgJA"
    ports:
      - "9092:9092"

  kafka-2:
    image: apache/kafka:4.4.0
    container_name: kafka-2
    networks:
      - kafka-net
    environment:
      KAFKA_NODE_ID: 2
      KAFKA_PROCESS_ROLES: broker,controller
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka-1:19092,2@kafka-2:19092,3@kafka-3:19092
```

```
      KAFKA_LISTENERS: PLAINTEXT://:9092,CONTROLLER://:19092
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9093
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      CLUSTER_ID: "XvcofFisS3eDkwD6W1SgJA"
    ports:
      - "9093:9092"

  kafka-3:
    image: apache/kafka:4.4.0
    container_name: kafka-3
    networks:
      - kafka-net
    environment:
      KAFKA_NODE_ID: 3
      KAFKA_PROCESS_ROLES: broker,controller
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka-1:19092,2@kafka-2:19092,3@kafka-3:19092
      KAFKA_LISTENERS: PLAINTEXT://:9092,CONTROLLER://:19092
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9094
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      CLUSTER_ID: "XvcofFisS3eDkwD6W1SgJA"
    ports:
      - "9094:9092"

networks:
  kafka-net:
    driver: bridge
```

10.2 Running the Cluster

Open a Linux command prompt in the directory containing `compose.yaml` file and run the following Docker Compose command:

```
docker compose up -d
```

Note that we're using docker compose version 2, so there is no - between the "docker" and "compose". The older version of docker compose was written docker-compose.

The command downloads the official Kafka 4.4.0 image and starts all three brokers in the background. Because we defined the `kafka-net` network, Docker's internal DNS allows the containers to communicate using container names (`kafka-1`, `kafka-2`, `kafka-3`) over the internal controller ports (19092).

10.3 Creating Topics and Connecting Clients

Docker maps the internal port 9092 of each container to ports 9092, 9093, and 9094 on the host machine, so you can interact with the cluster just like you did in the previous sections.

To create the replicated topic, use the same command as before:

```
bin/kafka-topics.sh --create --topic repliTopic --partitions 3 --replication-factor 3 --
    bootstrap-server localhost:9092
```

We can then run the same commands as above to test consumers and producers:

```
bin/kafka-console-consumer.sh --topic repliTopic --from-beginning --bootstrap-server
    localhost:9092,localhost:9093,localhost:9094
```

Then setup a console producer in another window:

```
bin/kafka-console-producer.sh --topic repliTopic --bootstrap-server localhost:9092,localhost
    :9093,localhost:9094
```

When testing resilience by killing a broker, you can simulate a failure gracefully using Docker Compose:

```
docker compose stop kafka-3
```

After stopping the kafka-3 container, run the describe command:

```
bin/kafka-topics.sh --describe --topic repliTopic --bootstrap-server localhost:9092
```

We have three nodes and three partitions with a leader on each of the (1 in this case). There are two in sync replicas because we defined a minimum of 2 for each topic and the third one is offline:

```
Topic: repliTopic       TopicId: VHRAUrw7SJi7G5zqlxhvUQ PartitionCount: 3
    ReplicationFactor: 3    Configs: min.insync.replicas=2
        Topic: repliTopic       Partition: 0    Leader: 1       Replicas: 3,1,2 Isr: 1,2
                Elr:    LastKnownElr:
        Topic: repliTopic       Partition: 1    Leader: 1       Replicas: 1,2,3 Isr: 1,2
                Elr:    LastKnownElr:
        Topic: repliTopic       Partition: 2    Leader: 1       Replicas: 2,3,1 Isr: 1,2
                Elr:    LastKnownElr:
```

To restore connectivity and have the broker rejoin the cluster, simply start the container again:

```
docker compose start kafka-3
```

Running the describe command above shows that things are back to being in sync:

```
Topic: repliTopic       TopicId: VHRAUrw7SJi7G5zqlxhvUQ PartitionCount: 3
    ReplicationFactor: 3    Configs: min.insync.replicas=2
        Topic: repliTopic       Partition: 0    Leader: 1       Replicas: 3,1,2 Isr: 1,2,3
                Elr:    LastKnownElr:
        Topic: repliTopic       Partition: 1    Leader: 1       Replicas: 1,2,3 Isr: 1,2,3
                Elr:    LastKnownElr:
        Topic: repliTopic       Partition: 2    Leader: 2       Replicas: 2,3,1 Isr: 1,2,3
                Elr:    LastKnownElr:
```