SE 424: Distributed Systems	**Recitation 3c**
Semester 1 5785	**18 Nov 2024**
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.

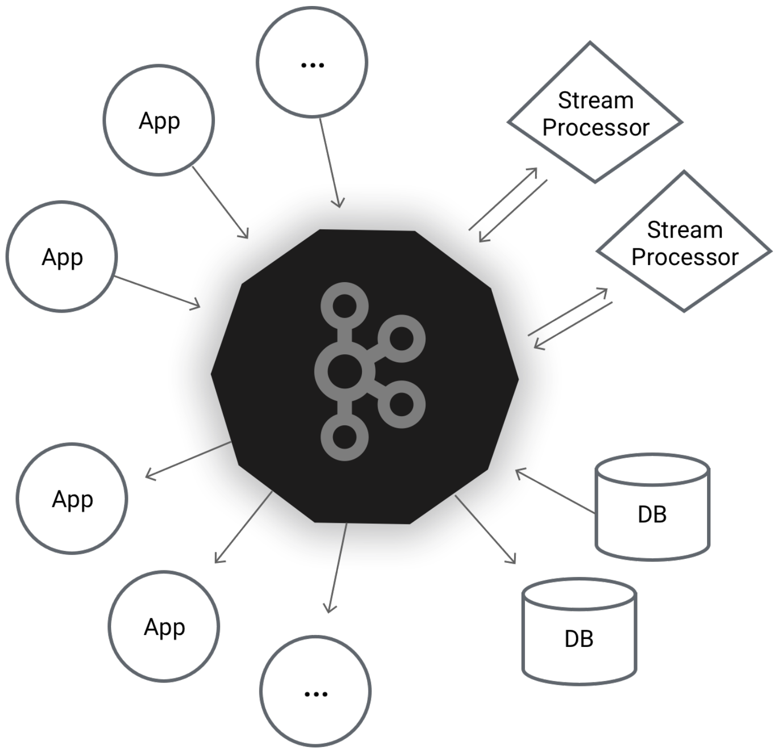

Figure 1: Kafka General Architecture

The general architecture of Kafka is shown in Figure 1. 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 3.3.1 package. It can be downloaded from:

```
https://downloads.apache.org/kafka/3.6.1/kafka_2.13-3.6.1.tgz
```

To save time, I'll also put a copy of it on the shared H: drive.

You will need to use Linux for this recitation because Kafka 3 does not work on Windows 10 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) if you want, but the installation is non-trivial.

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 Starting ZooKeeper

Apache ZooKeeper (`https://zookeeper.apache.org/`) is the server which coordinates between Kafka brokers (servers) and clients. A general figure of how ZooKeeper works is shown in Figure 2. ZooKeeper lets you manage clusters of servers and coordinates between them. It takes care of leader election and fault detection, two topics that we'll get to later in the semester.

You can read more about Zookeeper at:

`https://zookeeper.apache.org/doc/r3.9.1/index.html`

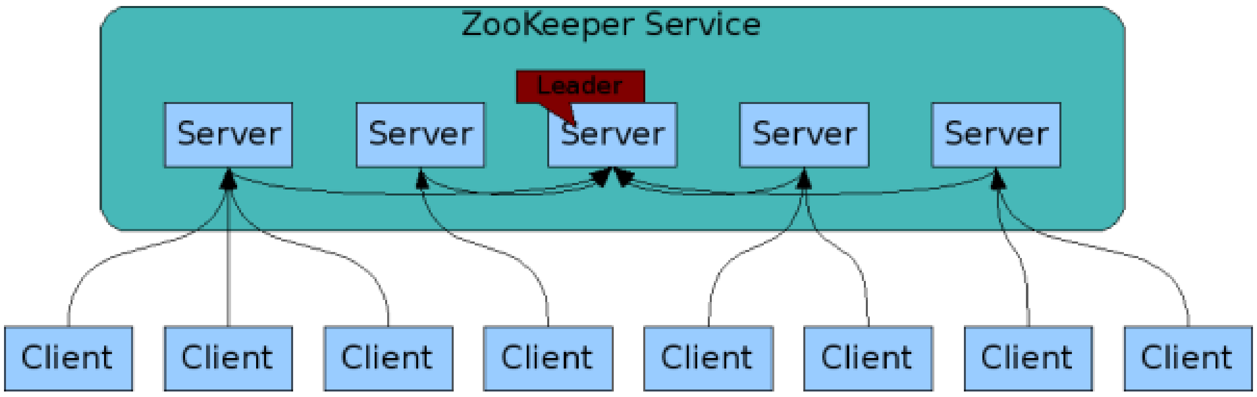

Figure 2: ZooKeeper General Architecture

We'll start up ZooKeeper using the commands shown below:

```
> bin/zookeeper-server-start.sh config/zookeeper.properties
```

If the command succeeds, you'll have a working ZooKeeper instance running which will let you connect Kafka brokers.

2.1 Kafka without ZooKeeper

Kafka has been reliant on ZooKeeper since its inception over 10 years ago. ZooKeeper provides coordination between Kafka brokers (servers), covering many of the basics of consensus and distributed communication. Newer versions of Kafka include the Raft distributed consensus protocol (KRaft) that will eventually enable Kafka to operate without ZooKeeper's help. The implementation of the new protocol is ongoing and only recently was declared to be production ready. We'll use the classic ZooKeeper syntax for this recitation.

2.2 A note about JDK versus JRE

When you try to run the above command, you might get an error warning which looks like:

```
Missing server JVM at ‘C:\Program Files\Java\jre8\bin\server\jvm.dll’.
Please install or use the JRE or JDK that contains these missing components.
```

The issue is that the Java Runtime Environment (JRE) which you usually run doesn't come with the Java server executables. They're usually distributed only with the Java Development Kit (JDK). You can solve the problem by installing a full JDK or openJDK that includes the Java server environment. Check out `https://www.oracle.com/java/technologies/downloads/` to find one that's appropriate or use the built in Linux package manager (*e.g.* , APT) to install it.

3 Running 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.

3.1 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 24: broker.id=0
  - 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 34: listeners=PLAINTEXT://:9092
  - This value is what 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.
- Line 38: advertised.listeners=PLAINTEXT://your.host.name:9092
  - This field is what the broker will tell the ZooKeeper its IP address and port are. The difference between this one and the previous one is that this one is what outsiders think the broker's IP and port 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 62: log.dirs=/tmp/kafka-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.

- Lines 105, 112, 116: 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.
- Line 125: zookeeper.connect=localhost:2181
  - This points the broker to the ZooKeeper instances it can use. ZooKeeper itself is fault tolerant, so you can have a cluster of ZooKeepers and let the broker choose from one of them.
  - If you use the KRaft option in Kafka, you will not need to connect a ZooKeeper instance.

4 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 (don't close the Kafka broker or ZooKeeper windows).

```
> bin/kafka-topics.sh --create --replication-factor 1 --partitions 1 --topic test
--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)
- How many replicas to keep. Since the value is 1, that means that only one broker will manage it. If that broker fails without warning, we may lose data.
- The number of partitions (1). Partitions refer to the number of pieces that of the log that will be stored. They allow you to have logs that are too large for any one server to hold. We'll use 1 for now since we're not going to be making huge logs.

Once the command above is run, we'll see a success message. If you don't get a success message, check that the ZooKeeper and Kafka servers are still running in the background.

4.1 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 test 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 test --bootstrap-server localhost:9092
```

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

```
Topic: test TopicId: YlFxY73lRRW4SKkZ5i9eTQ PartitionCount: 1 ReplicationFactor: 1 Configs:
 Topic: test Partition: 0 Leader: 0 Replicas: 0 Isr: 0
```

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

- The name of the topic (test) and its unique ID (YlFxY73lRRW4SKkZ5i9eTQ)
- 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 0 (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.

5 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 --bootstrap-server localhost:9092 --topic test --from-beginning
```

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 (test).
- 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.

6 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 --bootstrap-server localhost:9092 --topic test
```

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 --bootstrap-server localhost:9092 --topic test < LICENSE
```

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

6.1 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. Then start it again using the following command (note that I removed the last flag from the previous version):

```
> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test
```

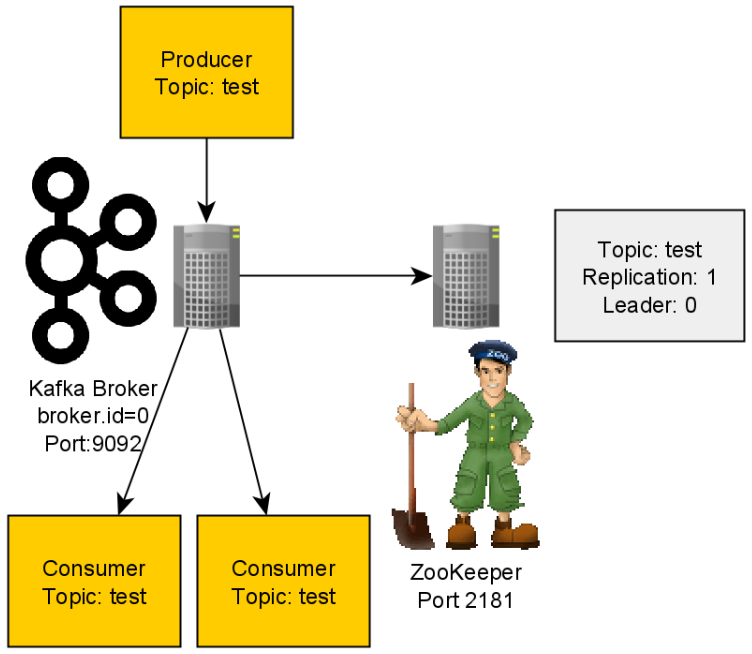

Figure 3: Single Broker Cluster

What happens?

7 Summary So Far

So far we have built a simple Kafka and ZooKeeper setup with one Kafka broker and one ZooKeeper coordinating. The architecture is as shown in Figure 3.

There are two consumers that get data from the topic called "test" via the single Kafka broker. Note that the broker is labeled with its `broker.id=0` value from its configuration file (`server.properties`).

8 Building a Multibroker Cluster

We built a single broker cluster so far. Let's make our setup more interesting by adding a few more brokers on different computers and letting ZooKeeper arrange the communication between them.

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 brokers and ZooKeeper 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 4.

We're assuming that the original Kafka broker and ZooKeeper instance that you started were on 10.0.0.2. If they were on a different IP address, change the addresses below accordingly.

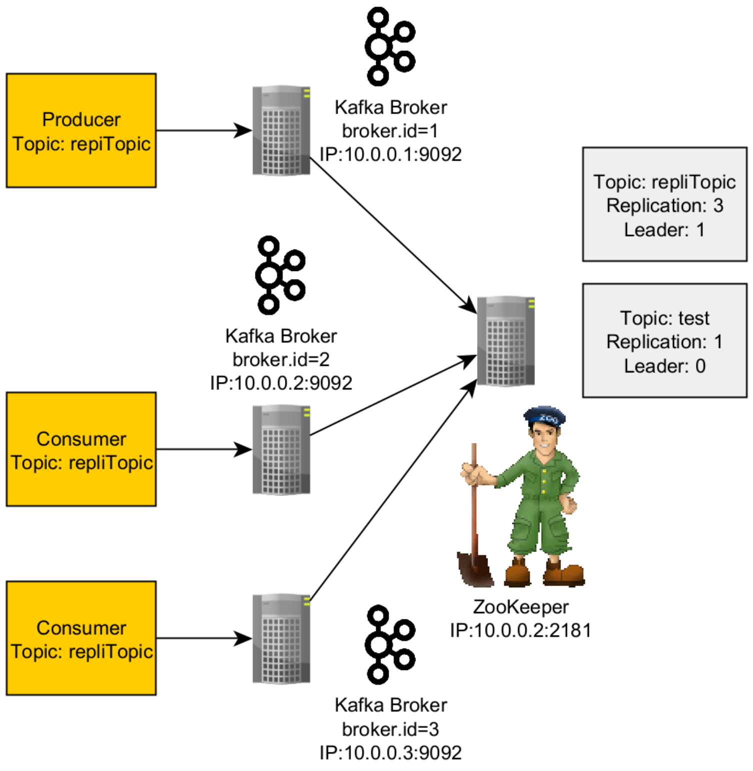

Figure 4: Multi-Broker Cluster

8.1 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:

8.1.1 Broker at 10.0.0.1 server.properties file

- `broker.id=1 (Line 24)`
- `zookeeper.connect=10.0.0.2:2181 (Line 125)`

8.1.2 Broker at 10.0.0.2 server.properties file

- `broker.id=2 (Line 24)`
- `zookeeper.connect=10.0.0.2:2181 (Line 125)`

8.1.3 Broker at 10.0.0.3 server.properties file

- `broker.id=3 (Line 24)`
- `zookeeper.connect=10.0.0.2:2181 (Line 125)`

8.2 Running multiple brokers on a single computer

In case you don't have access to multiple computers, it's possible to create a fake "cluster" on a single computer. To do that, you need to change the following elements:

1. Give the files three different file names
2. Make the brokers all have different IDs (as above)
3. Make the brokers all listen on different ports (say 9092, 9093, 9094).
4. Give separate log directories for each broker (say /tmp/kafka-logs1, /tmp/kafka-logs2, /tmp/kafka-logs3)

8.3 Starting Up the Cluster

Start up the ZooKeeper instance with the same configuration information as before on 10.0.0.2 only. Don't start any other ZooKeeper instances.

Start up the three brokers using the configuration files you defined above. Ensure they are all up and running correctly.

Once you have the cluster up, set up the test topic as above (Sections 4, 5, 6) and test that it works.

8.4 Setting up a Replicated Topic

We next will set up a topic with replication factor 3. In Linux enter the following command:

```
> bin/kafka-topics.sh --create --bootstrap-server 10.0.0.2:9092 --replication-factor 3
--partitions 1 --topic repliTopic
```

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 --bootstrap-server 10.0.0.2:9092
```

You should get an output that looks like:

```
Topic:repliTopic PartitionCount:1 ReplicationFactor:3 Configs:  Topic:  repliTopic
Partition:  0 Leader:  1 Replicas:  1,2,3 Isr:  1,2,3
```

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.

In this case since we have three replicas of the topic, one (1) is the leader 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.

8.5 Testing the Replication Resilience

Follow the commands in Sections 5 and 6 to set up two consumers and producers for repliTopic. Once they are set up, have the producers send a few messages and ensure that the data arrive correctly at the consumers.

Next, find the leader for repliTopic (using the kafka-topics.bat command above), and shut it down. You can do that by pressing Ctrl+C on the window for the leader.

After you shut down the topic, run the describe command on repliTopic again (using kafka-topics.bat) and see who the new leader is. Use the producers to send more messages and check that they still arrive at the consumers.

9 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`.

10 Kafka Streams

Follow the quickstart example at the link below to get experience with Kafka streaming processing.

- `https://kafka.apache.org/33/documentation/streams/quickstart`

11 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. You'll need to use the following classes from the Kafka client API:

1. Producer: Used by a program to send data using Kafka
2. Consumer: Used by a program to receive data using Kafka

You can install the classes from the API using Maven at the following id: org.apache.kafka:kafka-clients:3.6.1 using Intellij. If you use Maven POM files, add the code to your POM using the following XML:

```
<dependency>
 <groupId>org.apache.kafka</groupId>
 <artifactId>kafka-clients</artifactId>
 <version>3.6.1</version>
</dependency>
```

11.1 Producer

Create a class that uses a `Producer` to send the contents of a text file using a topic called "filesender." You can get the file name to send from the command line as a parameter. Send the file using the `ProducerRecord` class. You can give null keys if you wish.

Send the file line by line (using readline).

11.2 Consumer

Create a class that uses a Consumer to receive the file from the topic filesender. Print the contents of the file out to the command line.