1. Minimal Cluster Overview:

1. **Cluster Overview**
    - **3 Kafka brokers**:
        - Each broker can serve as a controller, leader, or follower as needed.
        - Roles are dynamically assigned by Kafka during runtime.
    - **Zookeeper**: Ensures controller election and metadata management.
2. **Nodes**:
    - **Controller**: Runs Zookeeper for metadata management.
    - **Leader Broker**: Handles client requests.
    - **Follower Broker**: Replicates data from the leader.
3. **Environment**:
    - 3 Ubuntu VMs on VirtualBox.
    - Each node has static IPs for communication.

2. System Requirements:

Virtual Machine Requirements:

VM Role	RAM	CPU Cores	Disk Space
Broker 1	4 GB	2 cores	75 GB
Broker 2	4 GB	2 cores	75 GB
Broker 3	4 GB	2 cores	75 GB

Operating System:

- Ubuntu (20.04 LTS or later is recommended).

Dependencies:

- Java 8 (Kafka requires Java to run).
- Zookeeper Comes with Kafka, no separate installation required.

Kafka:

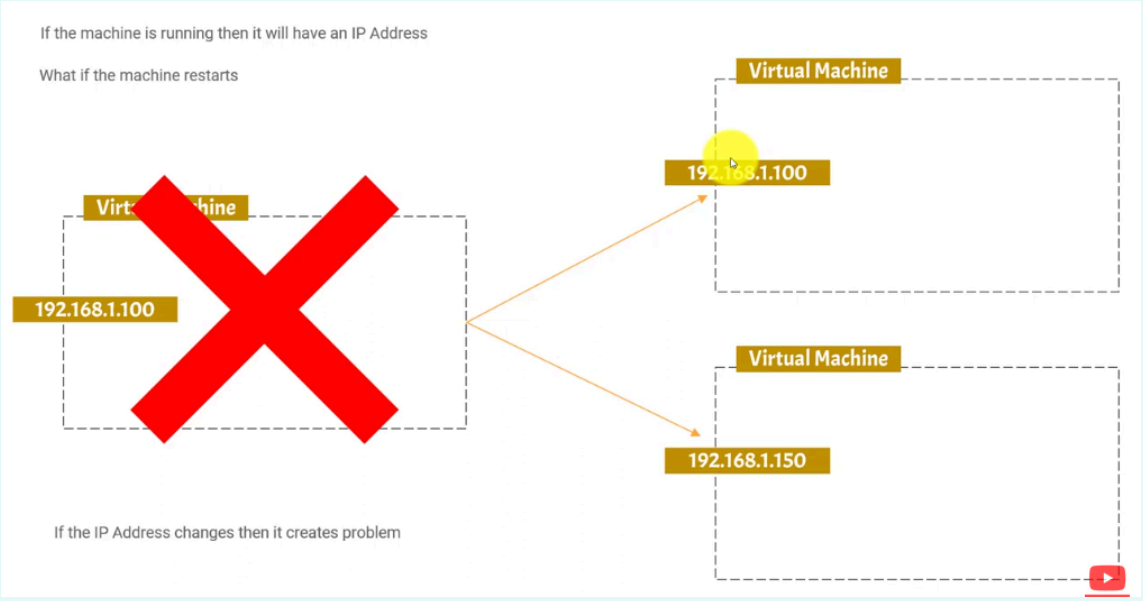
- Kafka 3.7.1

Networking Requirements:

Tip

Why Static IPs Are Essential for a Kafka Cluster

1. **Dynamic IP Problems**:
   - If IPs are assigned dynamically via DHCP, they might change after a VM restarts, disrupting communication between nodes.



2. **Solution: Static IPs**:
   - Assigning static IPs ensures predictable communication between nodes, even after restarts.
   - Static IPs make it easier for Kafka brokers and Zookeeper instances to reliably communicate using fixed addresses.

Hint

- **Private IP Range**:
  - The `192.168.x.x` range is reserved for private networks, making it suitable for internal communication between virtual machines without conflicting with public IPs.
- **Host-Only Network**:
  - In VirtualBox, the **Host-Only Network** typically defaults to the `192.168.56.x` subnet.
  - This allows VMs to communicate with each other and the host machine but not with the internet (unless explicitly configured).

- **Incremental IPs**:
  - IPs are assigned incrementally for simplicity and clarity.
  - Each VM is assigned a unique address within the same subnet (`192.168.56.0/24`) to avoid conflicts.

Configure the Network for Each VM (For instance we will configure the first VM. Then, after cloning, configure the other VMs)

Part 1: Enable NAT Adapter (Adapter 1)

1. Open **VirtualBox Manager**.
2. Select your VM and go to **Settings > Network**.
3. Enable **Adapter 1**:
   - Check **Enable Network Adapter**.
   - Set **Attached to** to **NAT**.
4. Save the changes.

> **Purpose**: Adapter 1 provides internet access to the VM but is not used for communication between cluster nodes.

Part 2: Enable Host-Only Adapter (Adapter 2)

1. Go to **File > Tools > Network Manager**.
2. Click **Create** to add a new host-only network (e.g., `vboxnet0`).
   - This creates a virtual network interface with a default IP range like `192.168.56.0/24`.
3. Go to VirtualBox settings for your VM.
4. Navigate to the **Network** tab.
5. Set **Adapter 1** to **Host-Only Adapter**.
6. Select the created Host-Only Network (e.g., `vboxnet0`) from the **Name** dropdown.

> **Purpose**: Adapter 2 is used exclusively for internal communication between VMs in your cluster.

Tip

Why Use Two Adapters?
In a typical cluster setup, we need two types of connectivity:

1. **Adapter 1: NAT (for internet access)**

- Allows each VM to access the internet (e.g., for installing updates or software).
- This is helpful for downloading Kafka, dependencies, or any tools you might need.

2. **Adapter 2: Host-Only (for internal cluster communication)**
   - Creates a private network where the VMs can communicate with each other.
   - Ensures predictable communication by assigning static IPs for each VM in this network.
   - This network does **not rely on DHCP** and remains isolated from the internet.

Part 3: Verifying the Configuration:

After starting the VMs:

1. Inside each VM, check the available network interfaces:

```
ip addr
```

- You should see two interfaces:
  - `enp0s3`: Assigned to Adapter 1 (NAT).
  - `enp0s8`: Assigned to Adapter 2 (Host-Only).
- Assign static IPs **only** to the `enp0s8` interface (Host-Only Adapter).

Part 4: Set Static IPs in Ubuntu VMs

1. Edit Network Configuration:
   - Open a terminal in each VM and edit the network configuration file:

```
sudo nano /etc/netplan/00-installer-config.yaml
```

2. Configure a static IP for the Host-Only Adapter (`enp0s8`):

```
        network:
          version: 2
          ethernets:
            enp0s8:
              addresses:
                - 192.168.56.104/24
              gateway4: 192.168.56.1
              nameservers:
                addresses:
```

```
                - 8.8.8.8
                - 8.8.4.4
```

Repeat this step for each VM, assigning a unique IP (e.g., `192.168.56.102`, `192.168.56.103`, etc.).

3. Apply the configuration:

```
sudo netplan apply
```

> **Hint**
>
> **Netplan configuration** ensures each VM can connect to others on the network.

4. Check the static IP is set correctly:

```
ip addr
```

3. Step-by-Step Guide

Step 1: Prepare the First VM (Base VM):

1. **Create a Virtual Machine**:
   - Install Ubuntu Server on a new VirtualBox VM.
2. **Install Dependencies**:
   - **Java**: Kafka requires Java to run:

```
sudo apt update
sudo apt install openjdk-8-jdk -y
```

   - check java version

```
java -version
```

> **Success**

```
openjdk version "1.8.0_432"
OpenJDK Runtime Environment (build 1.8.0_432-8u432-ga~us1-0ubuntu2~20.04-ga)
OpenJDK 64-Bit Server VM (build 25.432-bga, mixed mode)
```

- **Find the Java Installation Path**: Use the following command to locate where Java is installed:

```
sudo update-alternatives --config java
```

✓ **Check**

Note the path up to `java-8-openjdk-amd64`.

```
maryem@Kafka-Node-1:~$ sudo update-alternatives --config java
There is only one alternative in link group java (providing /usr/bin/java):
/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
Nothing to configure.
```

2. **Set JAVA_HOME in `/etc/environment`**:
    - Open the environment file for editing:

    ```
    sudo nano /etc/environment
    ```

    - Add the following line at the end of the file:

    ```
    JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
    ```

    - Append `:$JAVA_HOME/bin` to the `PATH` variable:

    ```
    PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:$JAVA_HOME/bin"
    ```

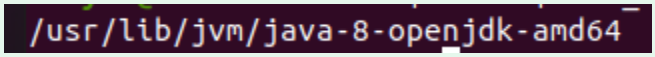
    - Reload Environment Variables:

    ```
    source /etc/environment
    ```

    - Verify JAVA_HOME:

```
echo $JAVA_HOME
```

✓ **Success**



3. **Test Java Configuration**
   - **Compile and Run a Simple Java Program**: Create a test file `HelloWorld.java`:

```
nano HelloWorld.java
```

   - Add the following code:

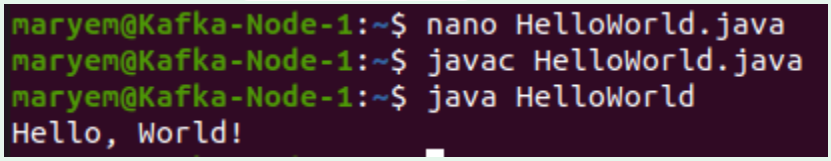
```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

   - Compile and run the program:

```
javac HelloWorld.java
java HelloWorld
```

✓ **Check**

You should see: `Hello, World!`



4. **Install Kafka:**
   - Download Kafka:

```
wget https://downloads.apache.org/kafka/3.7.1/kafka_2.12-3.7.1.tgz
```

- Extract and move Kafka:

```
sudo tar -xvzf kafka_2.12-3.7.1.tgz -C /opt/
sudo mv /opt/kafka_2.12-3.7.1 /opt/kafka
```

5. **Configure Kafka**:
   - Navigate to `kafka/config` directory

```
cd /opt/kafka/config
ls
```

Summary

This will display a list of configuration files in the `kafka/config` directory.

```
maryem@Kafka-Node-1:/opt/kafka/config$ ls /opt/kafka/config
connect-console-sink.properties    consumer.properties
connect-console-source.properties  kraft
connect-distributed.properties     log4j.properties
connect-file-sink.properties       producer.properties
connect-file-source.properties     server.properties
connect-log4j.properties           tools-log4j.properties
connect-mirror-maker.properties    trogdor.conf
connect-standalone.properties      zookeeper.properties
```

Connect Configuration Files: These files are used for Kafka Connect, a framework to integrate Kafka with external systems:

1. `connect-console-sink.properties`: Configuration for a **console sink connector**, which writes Kafka topic data to the console (standard output).
2. `connect-console-source.properties`: Configuration for a **console source connector**, which reads data from standard input and sends it to a Kafka topic.
3. `connect-distributed.properties`: Configuration for running Kafka Connect in **distributed mode**. In this mode, multiple worker nodes can run connectors to achieve scalability and fault tolerance.
4. `connect-file-sink.properties`: Configuration for a **file sink connector**, which writes data from Kafka topics to a file.
5. `connect-file-source.properties`: Configuration for a **file source connector**, which reads data from a file and sends it to a Kafka topic.
6. `connect-mirror-maker.properties`: Configuration for Kafka **MirrorMaker**, a tool to replicate data from one Kafka cluster to another.

7. `connect-standalone.properties`: Configuration for running Kafka Connect in **standalone mode**. This mode is typically used for testing or simple single-node setups.

Consumer and Producer Configuration

1. `consumer.properties`: Default configuration for Kafka consumers. It defines properties such as the group ID, deserialization settings, and broker connection details.
2. `producer.properties`: Default configuration for Kafka producers. It includes settings like serialization format, broker connection details, and batching configurations.

Core Kafka Server Configuration

1. `server.properties`:
   - The main configuration file for a Kafka **broker**.
   - Specifies properties like the broker ID, log directories, listeners (IP/port for communication), Zookeeper connection, and topic defaults.
2. `zookeeper.properties`:
   - Configuration for **Zookeeper**, which Kafka uses to manage cluster metadata (in non-KRaft setups).
   - Includes details like the data directory and port Zookeeper listens on.

Logging Configuration

1. `log4j.properties`: Configuration for **log4j**, the logging framework used by Kafka. It controls the log levels and log file locations for Kafka and Zookeeper processes.
2. `tools-log4j.properties`: A specialized log4j configuration file used by some Kafka tools for debugging or maintenance purposes.

Kafka KRaft (Kafka Without Zookeeper)

1. `kraft`: Configuration related to **KRaft mode**, Kafka's newer architecture that eliminates the need for Zookeeper and uses Kafka itself for metadata management. Typically used in clusters that don’t rely on Zookeeper.

Miscellaneous

1. `trogdor.conf`:Configuration for **Trogdor**, Kafka's workload generator tool. It’s used for testing cluster performance by simulating producer/consumer workloads.

- Edit `/opt/kafka/config/server.properties`:

```
broker.id=1
listeners=PLAINTEXT://192.168.56.104:9092
log.dirs=/var/lib/kafka/logs
zookeeper.connect=192.168.56.104:2181
```

Hint

`broker.id=1`

- **What it does**: Assigns a unique ID to this broker within the Kafka cluster.
- **Why it's important**: Each broker in a Kafka cluster must have a unique ID to distinguish itself. This ID is used by Kafka for internal cluster management (e.g., leader election, replication).

`listeners=PLAINTEXT://192.168.56.104:9092`

- **What it does**: Configures the address and port on which this broker will accept client and inter-broker connections.
- **Why it's important**: Kafka brokers need to listen for requests (e.g., from producers, consumers, and other brokers in the cluster). This property specifies the protocol, IP address, and port used for these connections.
- **Breaking it down**:
  - `PLAINTEXT`: The protocol used for communication. Kafka also supports **SASL_SSL** and **SSL** for encrypted or authenticated connections, but **PLAINTEXT** is the default.
  - 192.168.56.104: The IP address of the VM
  - `9092`: The default port Kafka brokers use for client communication.

`log.dirs=/var/lib/kafka/logs`

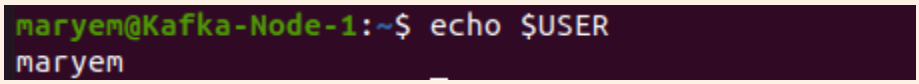
- **What it does**: stored and managed by kafka brokers to store topic data and partition logs
- **Why it's important**: Kafka logs are not for application-level logging but contain:
  - Topic partitions and their data.
  - Metadata about partitions and replication.
  - Offsets for consumer tracking.

⚠ **Caution**

- Ensure that the directory exists before starting Kafka:

```
sudo mkdir -p /var/lib/kafka/logs
sudo chown -R $USER:$USER /var/lib/kafka
```

- The path must be writable by the user running the Kafka process.



- Each broker writes only the partitions it is responsible for into its `log.dirs`.

5. **Configure Zookeeper**:
   - Kafka relies on Zookeeper for cluster management. Zookeeper is included by default when Kafka is downloaded.
   - Edit `/opt/kafka/config/zookeeper.properties`:

```
dataDir=/var/lib/zookeeper
dataLogDir=/opt/kafka/logs
```

💡 **Hint**

- **`dataDir`**: Specifies the directory where Zookeeper stores its core data, including snapshots and the `myid` file.
  `dataLogDir`: Specifies the directory where Zookeeper stores its transaction logs, which record changes to the data tree.
  - These logs capture every write operation (e.g., creating nodes, updating data) and are crucial for fault recovery. If Zookeeper crashes, it replays these logs to restore consistency.
- **`/var/lib/zookeeper`**: used by Zookeeper to store its metadata and snapshots
- **`/opt/kafka/logs`**: used for kafka's operational logs (e.g., error and event logs for troubleshooting)

⚠ **Caution**

- Ensure that the directory exists before starting Kafka:

```
sudo mkdir -p /opt/kafka/logs
sudo chown -R $USER:$USER /opt/kafka/logs
sudo mkdir -p /var/lib/zookeeper
sudo chown -R $USER:$USER /var/lib/zookeeper
```

6. **Start Zookeeper:**

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

Hint

`/opt/kafka/bin/zookeeper-server-start.sh`

- This is the script to start the Zookeeper server.
- The script handles initializing Zookeeper, setting up the server environment, and running Zookeeper in the foreground.

`/opt/kafka/config/zookeeper.properties`

- This is the configuration file for Zookeeper.
- Zookeeper uses the parameters in `zookeeper.properties` to configure itself, such as where to store data, what port to listen on, and how long to wait for client heartbeats.

Success

Zookeeper will start and listen on its default port (`2181`).

Warning

After starting Zookeeper, if the logs show that it has started successfully (as in the image you shared), you can safely leave it running. **Do not press** `Ctrl + C`, as it will stop Zookeeper.

Instead, open a **new terminal window or tab** to proceed with your Kafka setup or other commands. Zookeeper should keep running in the background in this terminal.

Tip

To use `systemctl` to manage Zookeeper, you need to create a `zookeeper.service` file. Here's how:

1. **Create the Service File**: Open a new file using your text editor:

```
sudo nano /etc/systemd/system/zookeeper.service
```

2. **Add the Following Content**: Replace `/opt/kafka` with the actual path to your Kafka installation.

```
[Unit]
Description=Apache Zookeeper Server
Documentation=http://zookeeper.apache.org
After=network.target

[Service]
Type=simple
ExecStart=/opt/kafka/bin/zookeeper-server-start.sh
/opt/kafka/config/zookeeper.properties
ExecStop=/opt/kafka/bin/zookeeper-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

3. **Reload Systemd to Register the Service**: After saving the file, reload the systemd daemon to recognize the new service:

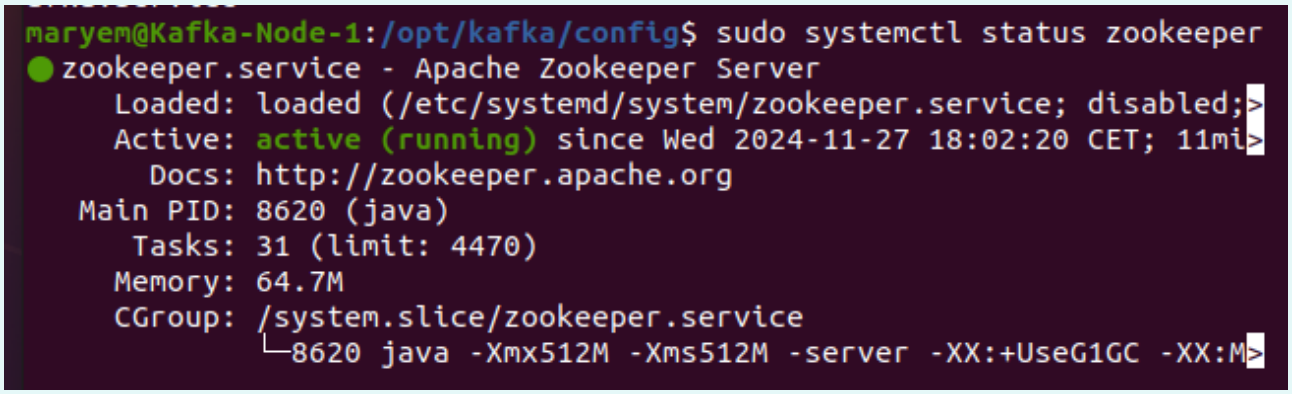
```
sudo systemctl daemon-reload
```

4. **Start the Zookeeper Service**: Now you can start Zookeeper with:

```
sudo systemctl start zookeeper
```

5. **To check if Zookeeper is running:**

```
sudo systemctl status zookeeper
```



6. **To view logs:**

```
journalctl -u zookeeper
```

7. **Start the Kafka broker to test:**

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

✓ Success

Kafka will start and listen for connections on the `PLAINTEXT` listener (default port `9092`).

Tip

To use `systemctl` to manage Zookeeper, you need to create a `kafka.service` file. Here's how:

1. **Create the Service File**: Open a new file using your text editor:

```
sudo nano /etc/systemd/system/kafka.service
```

2. **Add the Following Content**: Replace `/opt/kafka` with the actual path to your Kafka installation.

```
[Unit]
Description=Apache Kafka Server
```

```
Documentation=http://kafka.apache.org/documentation.html
After=network.target zookeeper.service

[Service]
Type=simple
User=kafka-node-1
ExecStart=/opt/kafka/bin/kafka-server-start.sh
/opt/kafka/config/server.properties
ExecStop=/opt/kafka/bin/kafka-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

3. **Reload Systemd to Register the Service**: After saving the file, reload the systemd daemon to recognize the new service:

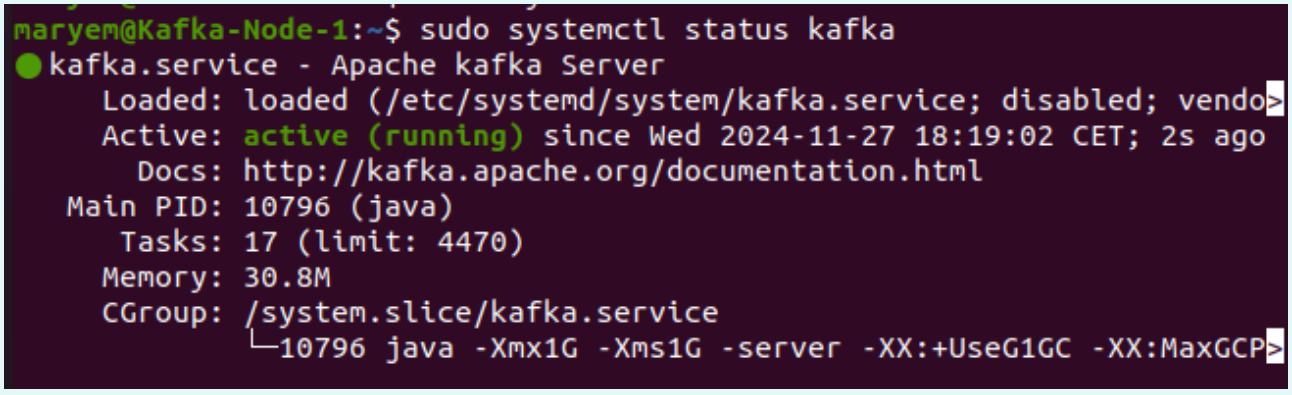
```
sudo systemctl daemon-reload
```

4. **Start the kafka Service**: Now you can start kafka with:

```
sudo systemctl start kafka
```

5. **To check if kafka is running:**

```
sudo systemctl status kafka
```



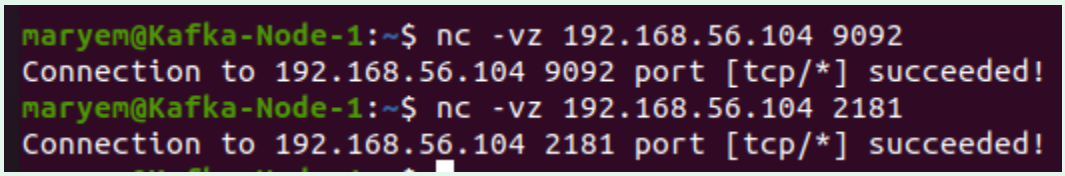
6. **To view logs:**

```
journalctl -u kafka
```

✓ **Check**

> Check connectivity using this command

```
nc -vz 192.168.56.104 9092
nc -vz 192.168.56.104 2181
```



if there's a an error regarding connectivity try using this command to ensure that the port is open:

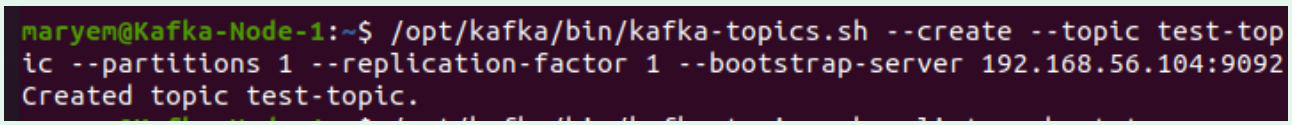
```
sudo ufw allow 9092
sudo ufw allow 2181
sudo ufw allow 2888
sudo ufw allow 3888
```

8. **Verify Kafka Works**
   - Verify Kafka works by producing and consuming test messages.
   - **Create a Test Topic**: Use the Kafka topic creation script to create a test topic:

```
/opt/kafka/bin/kafka-topics.sh --create --topic test-topic --partitions
1 --replication-factor 1 --bootstrap-server 192.168.56.104:9092
```

✓ **Success**

You should see a confirmation like:



- **List Topics**: Confirm that the topic exists:

```
/opt/kafka/bin/kafka-topics.sh --list --bootstrap-server 192.168.56.104:9092
```

✓ Success

The output should include:

```
maryem@Kafka-Node-1:~$ /opt/kafka/bin/kafka-topics.sh --list  --bootstrap-serve
r 192.168.56.104:9092
test-topic
```

Summary

Kafka Topics (`kafka-topics.sh`) Options

1. **`--create`**: Used to create a new Kafka topic.
   - Requires additional options like `--topic`, `--partitions`, and `--replication-factor`.
2. **`--delete`**: Used to delete an existing Kafka topic.
   - Requires the `--topic` option.
3. **`--list`**: Lists all the Kafka topics in the cluster.
4. **`--describe`**: Describes the details of a specific Kafka topic, including partitions, replicas, and other configurations.
   - Requires the `--topic` option.
5. **`--topic <topic_name>`**: Specifies the topic name for create, delete, or describe operations.
6. **`--partitions <number>`**: Specifies the number of partitions for a new topic (used with `--create`).
7. **`--replication-factor <factor>`**: Specifies the replication factor for a new topic (used with `--create`).
8. **`--config <config_name=value>`**: Sets a configuration for the topic (e.g., `retention.ms=1000000`).
   - Can be used with `--create` to set configurations like retention time, cleanup policy, etc.
9. **`--bootstrap-server <host:port>`**: Specifies the Kafka broker(s) to connect to for the operation.
10. **`--zookeeper <host:port>`**: (Deprecated) Specifies the Zookeeper server(s) to connect to for topic operations. It is now recommended to use `--bootstrap-server` for newer versions of Kafka.
11. **`--replica-assignment <partition>:<replica1>,<partition>:<replica2>,...`**: Used during topic creation to manually assign replicas to partitions.

- Example: `--replica-assignment 0:1,1:2`

12. `--alter`: Used to alter the configuration of an existing topic (e.g., to increase the number of partitions).

    - Can be used with `--topic` and `--partitions` to modify a topic's configuration.1>3. `--max-message-size <size>`: Defines the maximum message size that the broker will accept for this topic (used during topic creation).

14. `--validate-only`: Validates the configuration of a new or altered topic but doesn't actually apply the changes.
15. `--dry-run`: Simulates the topic creation or modification but doesn't apply the changes.
16. `--config <config_name=value>`: Allows setting configuration properties when creating or altering a topic, such as `cleanup.policy`, `retention.ms`, etc.
17. `--force`: Forces the creation or alteration of a topic even if it might cause issues like partitions with unequal replication or number of partitions mismatching the broker's configuration.
18. `--replica-placement-policy <policy>`: Defines a policy for how replicas should be placed across brokers.

> **Doc Link:** https://kafka.apache.org/documentation/#topicconfigs

- **Produce Messages**:
    - Start a producer to send messages to the topic:

```
/opt/kafka/bin/kafka-console-producer.sh --topic test-topic --bootstrap-server 192.168.56.1040:9092
```

    - Type some messages and press **Enter** after each. Example:

```
Hello, Kafka!
Welcome to the cluster!
```

✓ **Success**

```
maryem@Kafka-Node-1:~$ /opt/kafka/bin/kafka-console-producer.sh --topic test-to
pic --bootstrap-server 192.168.56.104:9092
>Hello, Kafka!
>Welcome to the cluster!
>
>maryem@Kafka-Node-1:~$
```

- Press Ctrl+D to exit

Summary

Kafka Console Producer (`kafka-console-producer.sh`) Options

1. `--topic <topic_name>`: Specifies the Kafka topic to which messages will be sent.
2. `--bootstrap-server <host:port>`: Specifies the Kafka brokers to connect to for producing messages.
3. `--producer-property <property=value>`: Allows setting producer-specific properties (like `acks`, `compression.type`, etc.).
4. `--key-serializer <serializer>`: Sets the serializer for the key (e.g., `org.apache.kafka.common.serialization.StringSerializer`).
5. `--value-serializer <serializer>`: Sets the serializer for the value (e.g., `org.apache.kafka.common.serialization.StringSerializer`).
6. `--compression-codec <codec>`: Defines the compression codec to use for messages (`gzip`, `snappy`, `lz4`, etc.).
7. `--batch-size <size>`: Specifies the size of the batch for producing records (in bytes).
8. `--linger-ms <time>`: Sets the time in milliseconds to wait before sending a batch, even if it's not full.
9. `--max-request-size <size>`: Defines the maximum size of a request (in bytes) that the producer will send to the broker.
10. `--acks <acks_value>`: Defines the acknowledgment level for the producer. Options include `0`, `1`, and `all`.
11. `--message-send-max-retries <count>`: Specifies how many retries the producer will attempt when a message send fails.
12. `--retry-backoff-ms <time>`: Specifies the backoff time in milliseconds between retry attempts.

> **Doc Link:** https://kafka.apache.org/documentation/#producerconfigs

- **Consume Messages**: Start a consumer to read messages from the topic:

```
/opt/kafka/bin/kafka-console-consumer.sh --topic test-topic --from-beginning
--bootstrap-server 192.168.56.104:9092
```

✓ **Success**

The messages produced earlier should appear:

Summary

Kafka Console Consumer (`kafka-console-consumer.sh`) Options

1. `--topic <topic_name>`: Specifies the Kafka topic to consume messages from.
2. `--bootstrap-server <host:port>`: Specifies the Kafka brokers to connect to for consuming messages.
3. `--from-beginning`: Starts consuming messages from the beginning of the topic (not just the latest).
4. `--group <group_id>`: Defines the consumer group that the consumer will belong to.
5. `--consumer-property <property=value>`: Allows setting consumer-specific properties (like `group.id`, `auto.offset.reset`, etc.).
6. `--key-deserializer <deserializer>`: Sets the deserializer for the key (e.g., `org.apache.kafka.common.serialization.StringDeserializer`).
7. `--value-deserializer <deserializer>`: Sets the deserializer for the value (e.g., `org.apache.kafka.common.serialization.StringDeserializer`).
8. `--max-messages <count>`: Limits the number of messages the consumer will read before exiting.
9. `--offset <offset>`: Defines from where to start reading (e.g., `earliest`, `latest`, or a specific offset).
10. `--property <property=value>`: Sets consumer-specific properties, like `auto.offset.reset`, `enable.auto.commit`, etc.
11. `--partition <partition>`: Specifies the partition number to consume from, if

needed.

12. `--timeout-ms <time>`: Sets a timeout in milliseconds after which the consumer will exit if no messages are available.

> **Doc Link:** https://kafka.apache.org/documentation/#producerconfigs

9. **Clean Up the VM for Cloning**:
    - Remove any machine-specific logs:

```
sudo rm -rf /tmp/kafka-logs
```

    - Shut down the VM

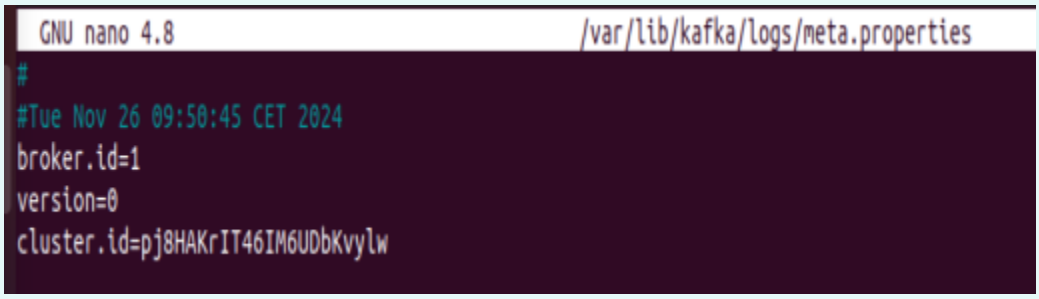
Step 2: Clone the VM:

1. **Clone the Base VM**:
    - In VirtualBox, right-click the prepared VM and select **Clone**.
    - Name the clones appropriately (e.g., `kafka-node-2`, `kafka-node-3`).
    - Choose **Generate new MAC addresses for all network adapters**
    - Choose **Full Clone** and ensure the cloned VMs are identical.
2. **Adjust VM Settings**:
    - Assign appropriate resources (RAM, CPU).
    - Set the network to **Host-Only Adapter** for each VM.
3. **Delete the `meta.properties` File** on each VM:

```
rm -f /var/lib/kafka/logs/meta.properties
```

> **Hint**
>
> 1. **What is `meta.properties`?**
>
> - The `meta.properties` file contains metadata specific to the broker, including:
>   - `broker.id`
>   - `cluster.id` (a unique ID for the Kafka cluster)
>
> 2. **Why delete or update `meta.properties` after cloning?**
>   - When you clone a VM, the `meta.properties` file from the source VM will be duplicated across all clones.

- If multiple brokers in a cluster share the same `broker.id` or `cluster.id`, Kafka will encounter conflicts, leading to errors like:
  - `java.lang.IllegalStateException: Duplicate broker id`
- Deleting or updating `meta.properties` ensures that each Kafka instance generates a **unique broker ID and cluster ID** when it starts.



Step 3: Configure the network connectivity

1. Inside each VM, check the available network interfaces:

```
ip addr
```

✓ **Success**

- You should see two interfaces:
  - **`enp0s3`**: Assigned to Adapter 1 (NAT).
  - **`enp0s8`**: Assigned to Adapter 2 (Host-Only).

2. Set Static IPs in Ubuntu VMs
   - Open a terminal in each VM and edit the network configuration file:

```
sudo nano /etc/netplan/00-installer-config.yaml
```

   - Configure a static IP for the Host-Only Adapter (`enp0s8`):

```
network:
  version: 2
  ethernets:
    enp0s8:
      addresses:
        - 192.168.56.105/24
```

```
      gateway4: 192.168.56.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 8.8.4.4
```

> ⚠ **Attention**
>
> - Assign static IPs **only** to the `enp0s8` interface (Host-Only Adapter).

3. Apply the configuration:

```
sudo netplan apply
```

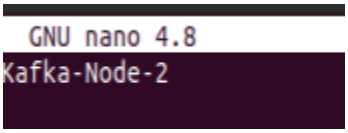
> 🔥 **Hint**
>
> **Netplan configuration** ensures each VM can connect to others on the network.

4. Check the static IP is set correctly:

```
ip addr
```

5. Change the `hostname`

```
sudo nano /etc/hostname
```



6. Update the `hosts` file:

```
sudo nano /etc/hosts
```

```
 GNU nano 4.8
127.0.0.1       localhost
192.168.56.105  Kafka-Node-2
192.168.56.106  Kafka-Node-3
192.168.56.104  Kafka-Node-1
```

7. Restart the VM
8. Check connectivity

```
maryem@Kafka-Node-2:/opt/kafka/bin$ ping Kafka-Node-2
PING Kafka-Node-2 (192.168.56.105) 56(84) bytes of data.
64 bytes from Kafka-Node-2 (192.168.56.105): icmp_seq=1 ttl=64 time=0.087 ms
64 bytes from Kafka-Node-2 (192.168.56.105): icmp_seq=2 ttl=64 time=0.094 ms
^C
--- Kafka-Node-2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.087/0.090/0.094/0.003 ms
maryem@Kafka-Node-2:/opt/kafka/bin$ ping Kafka-Node-1
PING Kafka-Node-1 (192.168.56.104) 56(84) bytes of data.
64 bytes from Kafka-Node-1 (192.168.56.104): icmp_seq=1 ttl=64 time=1.38 ms
64 bytes from Kafka-Node-1 (192.168.56.104): icmp_seq=2 ttl=64 time=1.40 ms
^C
--- Kafka-Node-1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 1.375/1.386/1.398/0.011 ms
maryem@Kafka-Node-2:/opt/kafka/bin$ ping Kafka-Node-3
PING Kafka-Node-3 (192.168.56.106) 56(84) bytes of data.
64 bytes from Kafka-Node-3 (192.168.56.106): icmp_seq=1 ttl=64 time=1.34 ms
64 bytes from Kafka-Node-3 (192.168.56.106): icmp_seq=2 ttl=64 time=1.60 ms
^C
--- Kafka-Node-3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 1.339/1.471/1.603/0.132 ms
```

Step 4: Configure SSH

Follow these steps to set up passwordless SSH access between the three Kafka nodes (VMs): `Kafka-Node-1`, `Kafka-Node-2`, and `Kafka-Node-3`.

1. On each Kafka node, install the OpenSSH server to enable SSH connections.

```
sudo apt update
sudo apt install openssh-server -y
```

2. On `Kafka-Node-1`, generate an SSH key pair

```
ssh-keygen -t rsa -P ""
```

3. Add the public key to the list of authorized keys on `Kafka-Node-1` to enable passwordless SSH for itself.

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

4. Copy the public key from `Kafka-Node-1` to `Kafka-Node-2` to allow passwordless SSH.

```
ssh-copy-id -i ~/.ssh/id_rsa.pub maryem@Kafka-Node-2
```

Alternatively, you can manually copy and append the key:

```
cat ~/.ssh/id_rsa.pub | ssh maryem@Kafka-Node-2 "mkdir -p ~/.ssh && cat >>
~/.ssh/authorized_keys"
```

5. Verify that passwordless SSH access from `Kafka-Node-1` to `Kafka-Node-2` is working.

```
ssh maryem@Kafka-Node-2
```

6. Repeat for `Kafka-Node-3` and `Kafka-Node-2`
7. Verify Connectivity Between All Nodes

> ⚠ **Caution**
>
> Ensure the hostname or IP addresses (`Kafka-Node-2`, `Kafka-Node-3`) resolve correctly. If not, update `/etc/hosts`

Step 5: Configure Kafka to Use Static IPs

1. Open the Kafka broker configuration file on each node:

```
sudo nano /opt/kafka/config/server.properties
```

2. Update the `advertised.listeners` and `listeners` properties with the static IP:

```
listeners=PLAINTEXT://192.168.56.105:9092  # Example for VM2
```

```
advertised.listeners=PLAINTEXT://192.168.56.105:9092
```

✓ Check

Check that firewall rules (if any) allow communication over ports `2181`, `2888`, and `3888`, `9092`.

⚠ Caution

In most cases, it is not strictly necessary to specify both `listeners` and `advertised.listeners` unless you have specific networking requirements, such as when Kafka brokers are behind firewalls, NAT, or when clients are connecting from different networks.

> **Doc Link:** https://kafka.apache.org/documentation/#brokerconfigs

Step 6: Update Zookeeper Configuration for Each VM

1. **Ensure Consistent Zookeeper Configuration Across All Nodes**
   - Edit the Kafka broker configuration file on each node:

```
sudo nano /opt/kafka/config/server.properties
```

   - Zookeeper connection string remains the same across all nodes:

```
zookeeper.connect=192.168.56.101:2181,192.168.56.102:2181,192.168.56.103
:2181
```

Hint

- The `zookeeper.connect` property lists all the Zookeeper servers that the Kafka broker will connect to for managing cluster metadata.
- This setup ensures redundancy and fault tolerance, allowing the broker to connect to other Zookeeper nodes if one goes down.
- The default port for Zookeeper client connections is `2181`.
- Example breakdown of the connection string:
  - **192.168.56.101:2181**: Zookeeper node on VM1.

- **192.168.56.102:2181**: Zookeeper node on VM2.
- **192.168.56.103:2181**: Zookeeper node on VM3.

2. **Configure Zookeeper on Each Node**

> **Tip**
>
> If you're running Zookeeper bundled with Kafka, you don't need `zoo.cfg`. You can directly edit or verify the configuration in `/opt/kafka/config/zookeeper.properties`.

- Edit the Zookeeper configuration file:

```
sudo nano /opt/kafka/config/zookeeper.properties
```

- Verify or add the following parameters:

```
# Directory for snapshots and logs
dataDir=/var/lib/zookeeper

#kafka's operational logs
dataLogDir=/opt/kafka/logs

# Port for client connections
clientPort=2181

# Number of ticks between heartbeats
tickTime=2000
# Number of ticks to wait for an initial connection
initLimit=10
# Number of ticks to allow for syncing between nodes
syncLimit=5
# Maximum client connections allowed per host
maxClientCnxns=60

# Define the ensemble's servers
server.1=Kafka-Node-1:2888:3888
server.2=Kafka-Node-2:2888:3888
server.3=Kafka-Node-2:2888:3888
```

> **Hint**

- `server.<id>` specifies the server's ID and its communication ports:
  - `server.1` defines **server 1** with the IP `192.168.56.104` that corresponds to the hostname `Kafka-Node_1`
  - `server.2` defines **server 2** with the IP `192.168.56.105` that corresponds to the hostname `Kafka-Node_2`.
  - `server.3` defines **server 3** with the IP `192.168.56.106` that corresponds to the hostname `Kafka-Node_3`.
  - The number after `server.` (e.g., `1`, `2`, `3`) is the **ID** of that server in the cluster.
- **2888**: Port for leader election.
- **3888**: Port for communication between Zookeeper nodes.
- The `dataDir` directory will store snapshots and the `myid` file.

2. For Zookeeper to identify each node in the ensemble, it uses the `myid` file located in the directory specified by `dataDir`
   (in this case, `/var/lib/zookeeper`). **Assign Unique IDs to Each Node**
   - Execute the following commands for each node:

```
# Command For Node 1
echo 1 > /var/lib/zookeeper/myid
# Command For Node 2
echo 2 > /var/lib/zookeeper/myid
# Command For Node 3
echo 3 > /var/lib/zookeeper/myid
```

   - Check the content of `myid` file:

```
cat /var/lib/zookeeper/myid
```

Hint

- Each Zookeeper server must have a file called `myid`.
- The `myid` file must contain a **single number** corresponding to the server's ID in the `server.<id>` configuration in `zookeeper.properties`.
- Example:
  - VM1: `myid` = `1`
  - VM2: `myid` = `2`

- VM3: `myid` = `3`

Summary

- **`dataLogDir`**:
  - Specifies a directory for transaction logs, which can improve performance if placed on a separate, fast disk. Defaults to `dataDir` if not set.
- **`tickTime`**:
  - Determines the interval (in ms) for Zookeeper's basic unit of time. It's used for heartbeats and timeouts. The default of `2000` is typically sufficient.
- **`initLimit`**:
  - Maximum number of `tickTime` intervals Zookeeper waits for a follower to connect and sync during startup.
- **`syncLimit`**:
  - Maximum number of `tickTime` intervals a leader waits for a follower to sync.
- **`maxClientCnxns`**:
  - Limits the number of client connections per host to prevent overloading.
- **`autopurge.snapRetainCount`**:
  - Specifies the number of snapshot files to retain. Older snapshots are automatically purged to save disk space.
- **`autopurge.purgeInterval`**:
  - Defines the frequency (in hours) for cleaning up old snapshots and transaction logs. Set to `0` to disable automatic purging.
- **`quorumListenOnAllIPs`**:
  - When `false`, binds the quorum ports (`2888` and `3888`) to the specific IP addresses defined in `server.<id>`. Improves security and prevents accidental exposure on all interfaces.

Step 4: Start the Cluster:

1. **Start Zookeeper**:
   - Start Zookeeper on all three VMs:

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

or

```
sudo systemctl start zookeeper
```

2. **Start Kafka Brokers**:
   - Start Kafka on all three VMs:

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

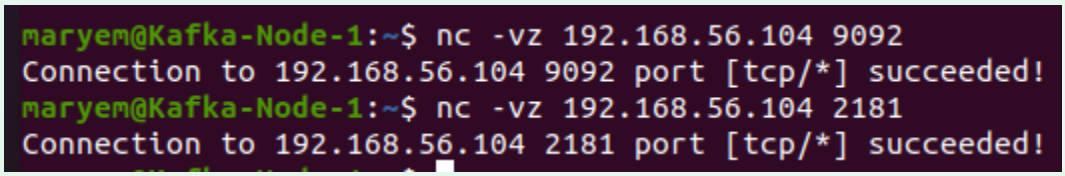
or

```
sudo systemctl start kafka
```

✓ **Check**

Check connectivity for all VMs using this command

```
nc -vz 192.168.56.105 9092 #specify the IP address of the VM
nc -vz 192.168.56.105 2181
```



if there's a an error regarding connectivity try using this command to ensure that the port is open:

```
sudo ufw allow 9092
sudo ufw allow 2181
sudo ufw allow 2888
sudo ufw allow 3888
```

Step 7: Determine the Controller Node

To identify the controller node in a Kafka cluster (the broker responsible for managing the metadata and partition leadership), follow these steps:

- **Check Controller Election**: Kafka automatically elects one broker as the controller. You can verify this:

```
sudo /opt/kafka/bin/zookeeper-shell.sh <zookeeper_address>
```

- Then run:

```
get /controller
```

✓ **Success**

All nodes must display the same controller

Here, the controller is the `Kafka-Node-1`

Step 8: Verify the cluster

Topic Creation

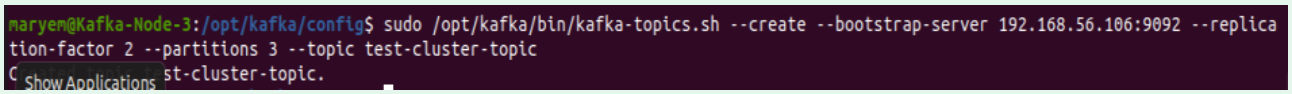
Tip

In Apache Kafka, you can create a topic from any broker in the cluster, not just a specific node. This is because all brokers in the cluster communicate with the controller node, which is responsible for managing metadata, such as topic creation and partition assignments.

1. Create a topic with replication across nodes.

```
sudo /opt/kafka/bin/kafka-topics.sh --create \
  --bootstrap-server <broker_address>:<port> \
  --replication-factor 2 \
  --partitions 3 \
  --topic test-cluster-topic
```

Success



Hint

Key Points on Topic Creation:

1. **Bootstrap Server**:
   - Use any broker's address in the cluster as the `--bootstrap-server`.
   - Example: `192.168.56.106:9092`
2. **Controller Node**:
   - The broker you specify as the bootstrap server does not need to be the **controller node**.
   - The controller node (elected automatically by Zookeeper) coordinates the topic creation process and updates metadata.
3. **Replication Across Nodes**:
   - Topic partitions are distributed across the cluster based on the replication factor.

- You don't need to explicitly choose which nodes hold replicas or leaders—that's managed by the controller.

Tip

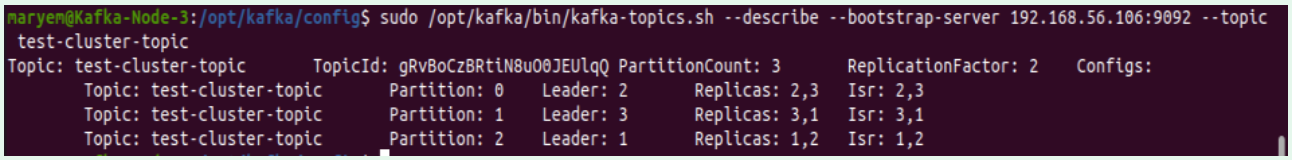
While you can connect to any broker, it's often beneficial to connect to the leader of the desired partition for optimal performance. However, Kafka's internal mechanisms will handle routing requests appropriately.

2. Verify the topic creation and observe its metadata::

```
sudo /opt/kafka/bin/kafka-topics.sh --describe \
  --bootstrap-server <broker_address>:<port> \
  --topic test-cluster-topic
```

Success

This command will output information about the topic, including the leader for each partition. The output will look something like this:



In this output:

- **Leader:** Broker ID managing read and write operations for this partition.
- **Replicas:** The list of brokers that have replicas of that partition.
- **Isr (In-Sync Replicas):** The brokers that are currently in sync with the leader (**followers**).

Hint

Leader and Follower Assignment Timing

- **Leaders and followers for partitions are determined at the time of topic creation** by the Kafka controller.

- When you create a topic, the controller decides which broker will act as the leader for each partition and assigns followers for replication based on the replication factor.
- This decision is based on Kafka's **partition assignment strategy**, which aims to balance partitions and replicas across the cluster.

You can find the leader and follower information **immediately after creating the topic** by describing the topic.

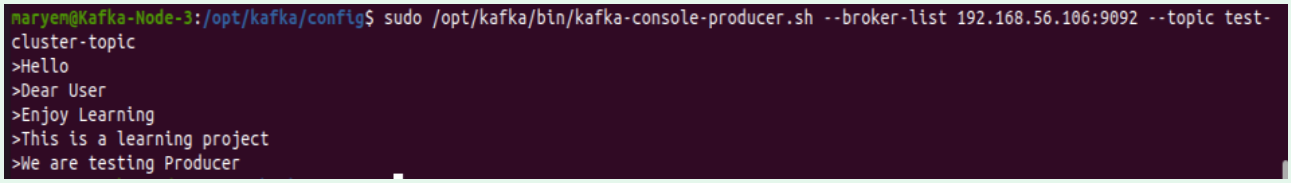
Publish Messages

1. Use the producer CLI to send messages to the topic:

```
sudo /opt/kafka/bin/kafka-console-producer.sh \
--broker-list <broker_address>:<port> \
--topic test-cluster-topic
```

✓ Success

Enter messages line by line, pressing Enter after each.



These messages are sent to the leader, which replicates them to followers.

ⓘ Info

Producers always interact with the **leader broker** for a partition. Publish messages to the topic:

Consume Messages

2. Consume the messages from the topic using the consumer CLI:

```
sudo /opt/kafka/bin/kafka-console-consumer.sh \
  --bootstrap-server <broker_address>:<port> \
  --topic test-cluster-topic \
```

```
  --partition 0
  --from-beginning
```

> ✓ **Success**
>
> This will display all messages from the beginning of the partition (fetched directly from the leader by default)

> ⓘ **Info**
>
> Consumers can fetch messages from either the **leader** or a **follower** (depending on configuration).

Observe Replication and Leader-Follower Behavior

1. Inspect the topic's replication and synchronization status:

```
sudo /opt/kafka/bin/kafka-topics.sh --describe \
  --bootstrap-server <broker_address>:<port> \
  --topic test-cluster-topic
```

> **Summary**
>
> **Summary of Command:**
>
> | Command | Description |
> | --- | --- |
> | `bin/kafka-topics.sh --create ...` | Create a new Kafka topic. |
> | `bin/kafka-console-producer.sh ...` | Publish messages to a Kafka topic. |
> | `bin/kafka-console-consumer.sh ...` | Consume messages from a Kafka topic. |
> | `bin/kafka-topics.sh --describe ...` | Describe a Kafka topic, showing leaders and replicas. |

Step 9: Simulate a Broker Failure

To observe fault tolerance and leader election:

1. Identify the leader broker for the partition.
2. Stop the leader broker
3. Recheck the topic details to observe the new leader election:

```
sudo /opt/kafka/bin/kafka-topics.sh --describe \
  --bootstrap-server <any_active_broker>:<port> \
  --topic test-cluster-topic
```

✓ **Success**

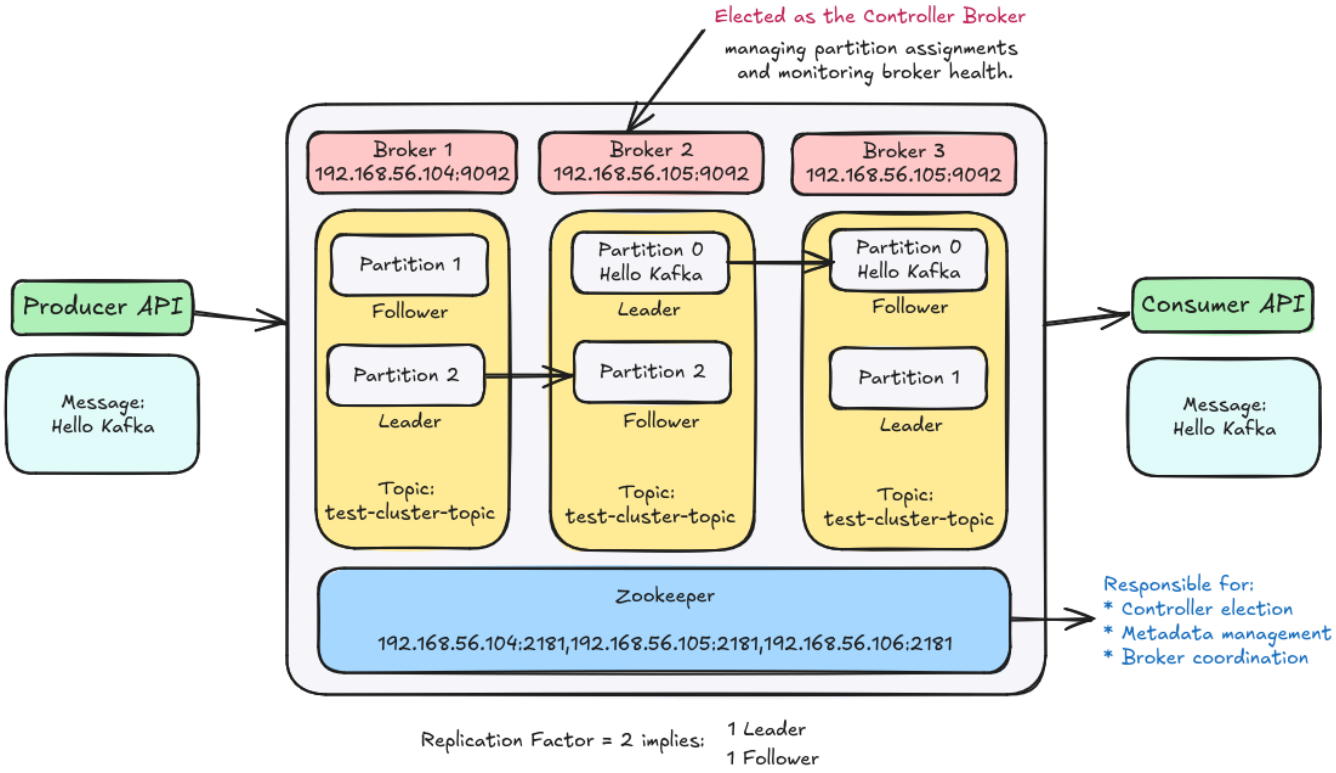
- A new leader from the ISR list should be elected.
- Messages will continue to be replicated and served by the new leader.

4. Bring the stopped broker back online
5. Recheck the ISR list to confirm that the restarted broker rejoins as a follower:

```
sudo /opt/kafka/bin/kafka-topics.sh --describe \
  --bootstrap-server <any_active_broker>:<port> \
  --topic test-cluster-topic
```

Check also the controller election when stopping one broker

3. Final Architecture


Elected as the Controller Broker
managing partition assignments and monitoring broker health.
Broker 1
192.168.56.104:9092
Broker 2
192.168.56.105:9092
Broker 3
192.168.56.105:9092
Partition 1
Follower
Partition 2
Leader
Topic:
test-cluster-topic
Partition 0
Hello Kafka
Leader
Partition 2
Follower
Topic:
test-cluster-topic
Partition 0
Hello Kafka
Follower
Partition 1
Leader
Topic:
test-cluster-topic
Producer API
Message:
Hello Kafka
Consumer API
Message:
Hello Kafka
Zookeeper
192.168.56.104:2181,192.168.56.105:2181,192.168.56.106:2181
Responsible for:
* Controller election
* Metadata management
* Broker coordination
Replication Factor = 2 implies:
1 Leader
1 Follower