Kafka Workflow: Creating a Topic and sending messages in Kafka

The purpose of this Standard Operating Procedure (SOP) is to provide:
A server that stores data and serves clients.

Why are we adding or managing brokers?

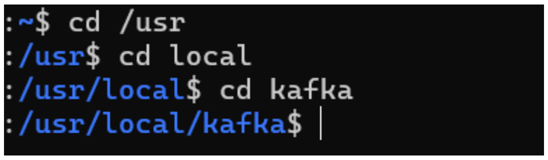
1. Adding a broker to a cluster makes the message-handling system faster and more reliable by spreading out the workload and providing backup if something goes wrong.
2. **Capacity**: More brokers can handle more messages at the same time, speeding up communication.
3. **Resilience**: If one broker has a problem, the others can still work, so the whole system is less likely to fail.

Step 1. Setting up your first broker

- **The command below Command Configures the Kafka server properties file. The file is located**

```
# Start the ZooKeeper service
bin/zookeeper-server-start.sh config/zookeeper.properties

# Start the Kafka broker
bin/kafka-server-start.sh config/server.properties
```



```
:~$ cd /usr
:/usr$ cd local
:/usr/local$ cd kafka
:/usr/local/kafka$
```

- **In the "config" directory of your Kafka installation. As the first Broker, you can use default configuration.**
- **This will start a single Kafka broker along with ZooKeeper, which manages broker coordination.**

Step 2. Adding a New Broker to the Cluster:

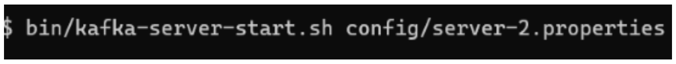
- **You will need to create a new "server.properties" file for the new broker to configure to run the following commands:**

```
broker.id=2
listeners=PLAINTEXT://:9093
log.dirs=/tmp/kafka-logs-2
```

- **The "broker.id" is unique within the cluster.**
- **You need to set a separate "`log.dirs`" path and ensure that the "`listeners`" port does not conflict with existing brokers.**

Start the new broker by running:

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

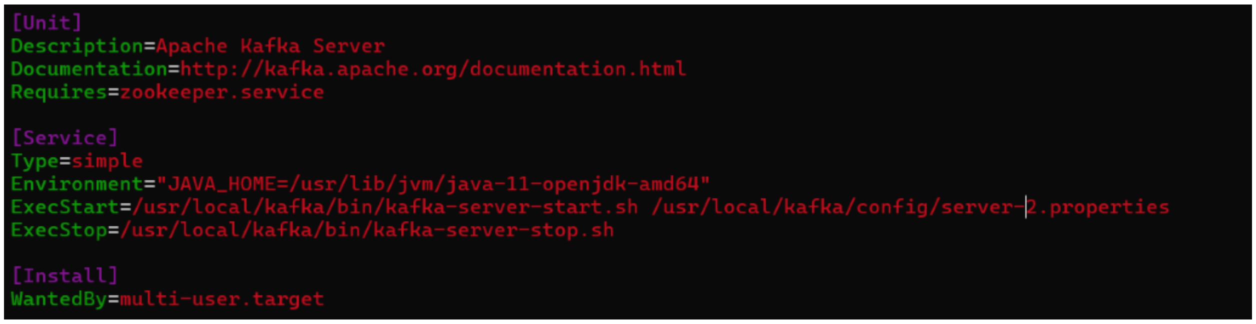


```
$ bin/kafka-server-start.sh config/server-2.properties
```

Step-2.5: adding new broker to boot:

- Run (note this example adds kafka-2 as the new broker):
  `Sudo vim /etc/systemd/system/kafka-2.service`

- Add the follow



```
[Unit]
Description=Apache Kafka Server
Documentation=http://kafka.apache.org/documentation.html
Requires=zookeeper.service

[Service]
Type=simple
Environment="JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64"
ExecStart=/usr/local/kafka/bin/kafka-server-start.sh /usr/local/kafka/config/server-2.properties
ExecStop=/usr/local/kafka/bin/kafka-server-stop.sh

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

- Run:
  ```
  Sudo systemctl enable kafka-2.service
  Sudo systemctl start kafka-2.service
  Sudo systemctl status kafka-2.service
  ```
  (should be started)

Step 3. Expanding the Cluster:

- **Expanding your Kafka cluster involves adding more brokers.**
- **You can repeat the above process for each new broker you wish to add**

Step 4. Monitoring Kafka Brokers:

- **Use the following command to enable JMX when starting a broker:**

```
KAFKA_JMX_OPTS="-
Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.authenticate=fa
lse -Dcom.sun.management.jmxremote.ssl=false"
bin/kafka-server-start.sh
config/server.properties
```

- **Dcom.sun.management.jmxremote.authenticate=false"**:

  -Disables authentication for JMX connections. This means that anyone who can access the JMX port can monitor and manage the broker without providing credentials.
- **Dcom.sun.management.jmxremote.ssl=false:**

  -Disables SSL encryption for JMX connections. This simplifies setup but makes JMX communication insecure over the network.

Step 5. How to remove Brokers from the Cluster:

- **As with any dynamic system, sometimes you may need to remove brokers from the cluster. This could be for maintenance, scaling down, or decommissioning a server.**
  - **To remove a broker, you need to ensure that no partitions use the broker as their leader, which requires reassigning partitions through Kafka's partition reassignment tool.**

```
bin/kafka-reassign-partitions.sh --bootstrap-
server localhost:9092
 --reassignment-json-file reassign.json --
execute
```

- **This command allows administrators to dynamically adjust how Kafka partitions are distributed across the cluster. It helps in balancing load, improving performance, and ensuring fault tolerance by redistributing data across different brokers as needed.**