Multi-Broker Apache Kafka cluster in container:

- Run the zookeeper

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

- Open a new tab and run the docker

```
docker run --rm \
-e BROKER_ID=0 \
-e ZOOKEEPER_CONNECT=host.docker.internal:2181 \
-e KAFKA_LISTENERS=PLAINTEXT://:9092 \
-e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://host.docker.internal:9092 \
-p 9092:9092 \
debezium/kafka:2.5
```

Part	Description
`docker run`	Starts a new Docker container.
`--rm`	Automatically removes the container after it stops. Useful for temporary containers.
`-e BROKER_ID=0`	Sets the Kafka broker ID to 0. Each broker in a cluster needs a unique ID.
`-e ZOOKEEPER_CONNECT=host.docker.internal:2181`	Tells Kafka where to find Zookeeper. Uses host machine's Zookeeper running on port 2181.

`-e KAFKA_LISTENERS=PLAINTEXT://:9092`	Kafka listens on port 9092 on all network interfaces using PLAINTEXT protocol.
`-e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://host.docker.internal:9092`	Kafka tells clients to connect using this advertised address (important for accessing Kafka from the host).
`-p 9092:9092`	Maps container's Kafka port 9092 to host's port 9092, allowing external access.
`debezium/kafka:2.5`	Specifies the Kafka Docker image from Debezium, version 2.5. (If you use `:latest` and it doesn't exist, it throws an error.)

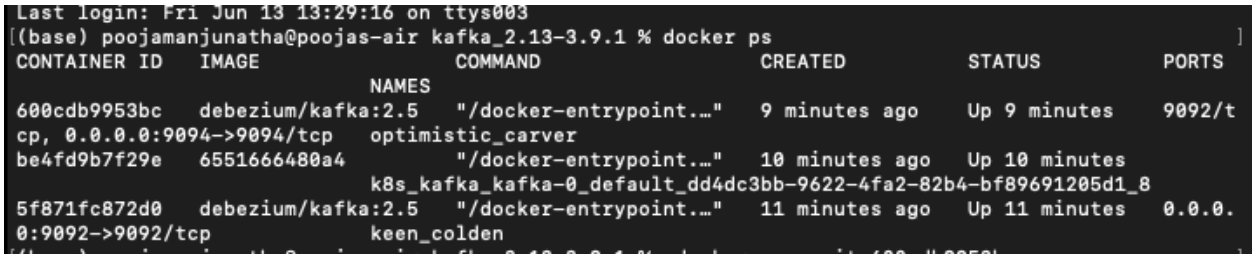
But with newer versions of KAFKA, it prefers mentioning NODE_ID over broker ID
Create 3 brokers:

```
docker run --rm \
  -e NODE_ID=0 \
  -e ZOOKEEPER_CONNECT=host.docker.internal:2181 \
  -e KAFKA_LISTENERS=PLAINTEXT://:9092 \
  -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://host.docker.internal:9092 \
  -p 9092:9092 \
  debezium/kafka:2.5

docker run --rm \
  -e NODE_ID=1 \
  -e ZOOKEEPER_CONNECT=host.docker.internal:2181 \
  -e KAFKA_LISTENERS=PLAINTEXT://:9093 \
  -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://host.docker.internal:9093 \
  -p 9093:9093 \
  debezium/kafka:2.5

docker run --rm \
  -e NODE_ID=2 \
  -e ZOOKEEPER_CONNECT=host.docker.internal:2181 \
  -e KAFKA_LISTENERS=PLAINTEXT://:9094 \
  -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://host.docker.internal:9094 \
  -p 9094:9094 \
  debezium/kafka:2.5
```

Check if the three brokers are running by using the command: *docker ps*



Next step is to run commands inside the container. The command is:

docker exec -it 600cdb9953bc /bin/bash

It gives you **interactive shell access** inside the container with ID `600cdb9953bc`. This is just like SSH'ing into a VM.

You'll be dropped into the container's terminal, where you can:

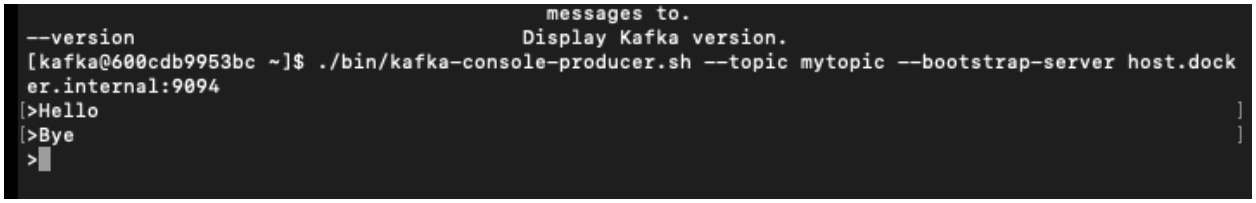
- Explore the filesystem
- Check logs
- Run commands (e.g., `kafka-topics.sh`)
- Troubleshoot issues

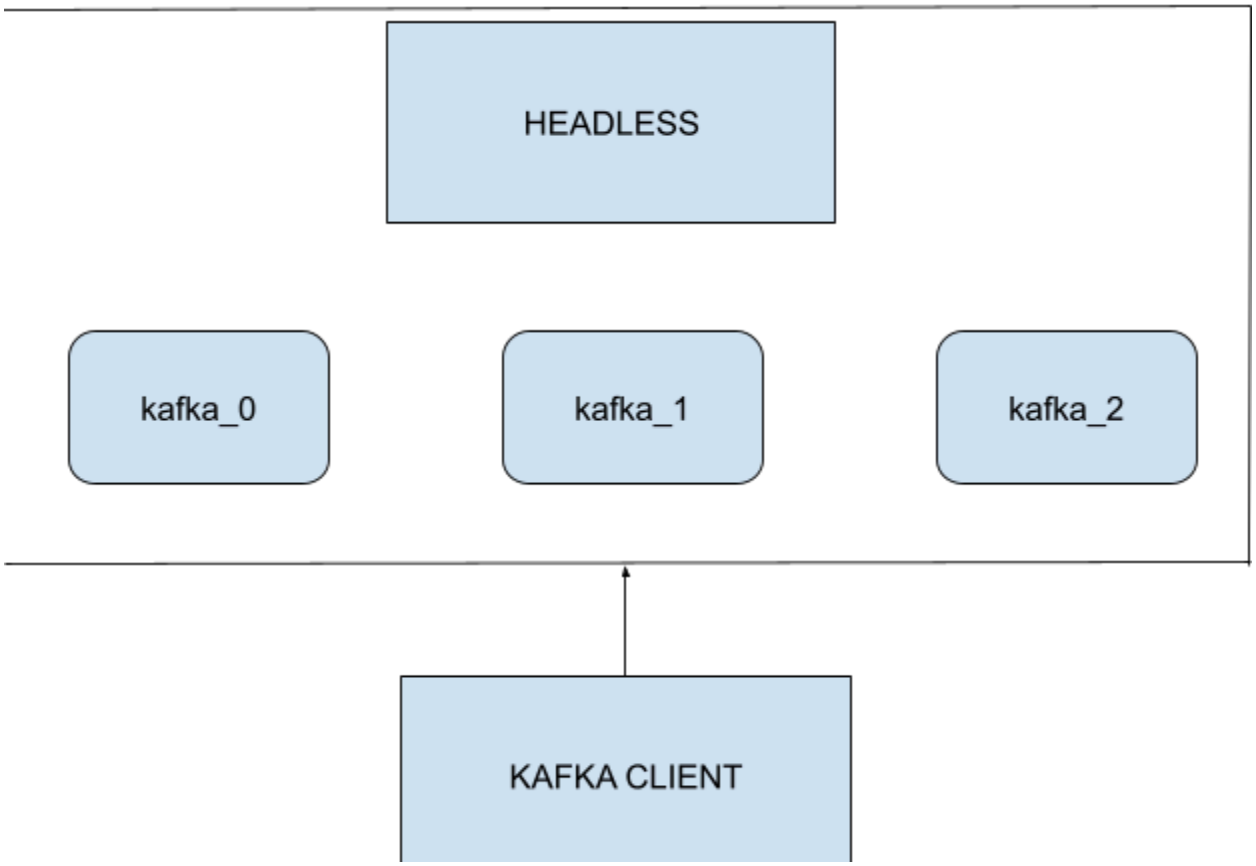
Create a topic called mytopic in the broker with the replication factor of 3 as there are 3 brokers.

./bin/kafka-topics.sh --create --topic mytopic --bootstrap-server host.docker.internal:9094 --partitions 1 --replication-factor 3

Produce messages in this topic using the command:

./bin/kafka-console-producer.sh --topic mytopic --bootstrap-server host.docker.internal:9094





These are **Kafka broker pods** managed as part of a **StatefulSet** in Kubernetes. They are named sequentially:

Pod Name	Broker ID
`kafka-0`	0
`kafka-1`	1
`kafka-2`	2

Each pod runs a Kafka broker and:

- Stores and serves partitions.
- Works with ZooKeeper (or Kraft mode in newer versions).
- Maintains state (thanks to StatefulSet: each pod gets a persistent identity and volume).

A **headless service** in Kubernetes is a service **without a cluster IP** (`clusterIP: None`). It is used for **direct pod-to-pod communication**.

Why use a headless service for Kafka?

- Brokers need to **know and directly communicate** with each other.
- Clients need to resolve **each broker individually** (e.g., `kafka-0.kafka-headless.default.svc.cluster.local`).
- This is **essential** for high availability and partition leadership.

So `kafka-headless` enables:

he **Kafka client** (producer or consumer) is usually:

- A separate **Kubernetes pod** or microservice.
- Connects to the Kafka cluster using the broker DNS names exposed by the **headless service**.
- Example bootstrap server config:

What are node ports?

Until now, he have accessed the broker from inside the container. But how do we access it from the outside? With the help of node ports: **NodePort** is a type of Kubernetes **Service** that exposes a pod **to the outside world (external clients)** by opening a specific port on **every node in the cluster**

Concept	Description
NodePort	A port (usually in the range `30000–32767`) exposed on every node in your cluster.
Access	You can access the service using `NodeIP:NodePort`.

Target	It forwards traffic to the **ClusterIP service**, which then routes it to the appropriate pod(s).

To access the docker from outside:

1. Make sure zookeeper is running
2. Docker and Kubernetes must be running

Run the commands:

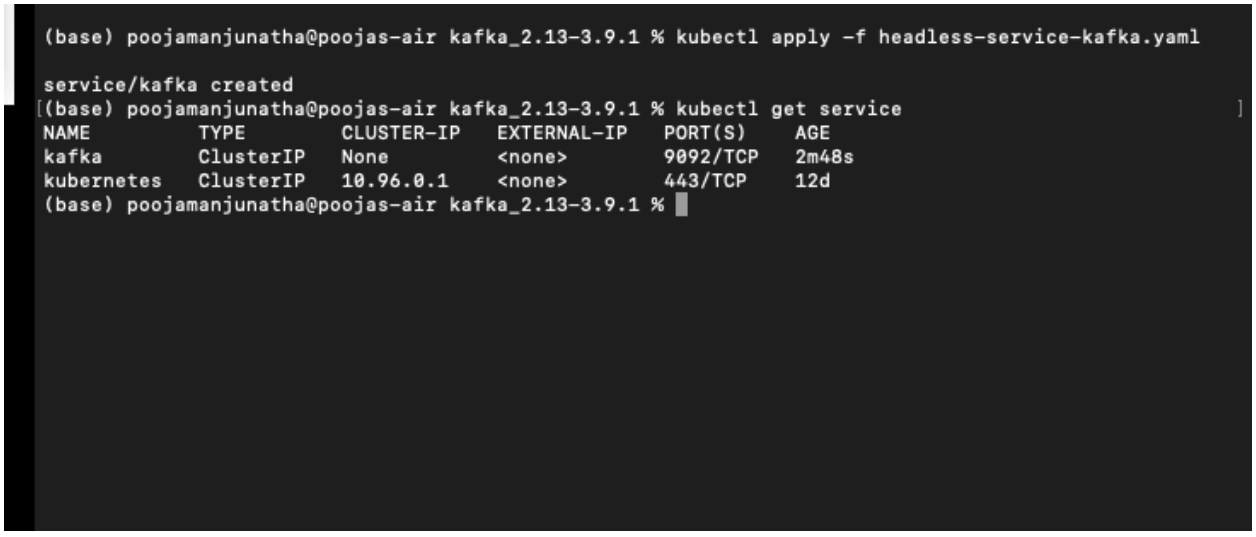
kubectl get nodes

```
NAME             STATUS   ROLES           AGE   VERSION
docker-desktop   Ready    control-plane   12d   v1.32.2
```

Create a file called headless-service-kafka.yaml file and paste the content and then run the command:

kubectl apply -f headless-service-kafka.yaml

To check if it has been created:
Run the command
Kubectl get service



Create a file for bootstrap-service-kafka as well and apply it .

Create 3 node ports file :

```
EOF
(base) poojamanjunatha@poojas-air kafka_2.13-3.9.1 % kubectl apply -f kafka-0-service.yaml -f kafka-
1-service.yaml -f kafka-2-service.yaml

service/kafka-0 created
service/kafka-1 created
service/kafka-2 created
(base) poojamanjunatha@poojas-air kafka_2.13-3.9.1 %
```

```
base) poojamanjunatha@poojas-air kafka_2.13-3.9.1 % kubectl get service
AME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
afka             ClusterIP   None            <none>        9092/TCP         30m
afka-0           NodePort    10.101.193.52   <none>        9093:30000/TCP   4m29s
afka-1           NodePort    10.108.109.51   <none>        9093:30001/TCP   4m29s
afka-2           NodePort    10.100.249.21   <none>        9093:30002/TCP   4m29s
afka-bootstrap   NodePort    10.110.186.14   <none>        9092:30003/TCP   14m
ubernetes        ClusterIP   10.96.0.1       <none>        443/TCP          12d
base) poojamanjunatha@poojas-air kafka_2.13-3.9.1 %
```

Create and apply another file called statefulset-multi-broker.yaml and apply it

Run: kubectl gte pods -w and this should be seen
Produce and cosume messages