Two Producers and two Consumers KAFKA using zookeeper

1. Create the Topic

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

Verify:

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

2. Start Two Producers (in separate terminals)

Producer 1

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

Producer 2

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

3. Start Two Consumers (in separate terminals)

Option A: Same Consumer Group (messages load-balanced)

Consumer 1

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

Consumer 2

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

Messages will be split between them.

Option B: Different Consumer Groups (both get all messages)

Consumer 1

```
/opt/kafka/bin/kafka-console-consumer.sh --topic test --
bootstrap-server localhost:9092 --group group1
```

Consumer 2

```
/opt/kafka/bin/kafka-console-consumer.sh --topic test --
bootstrap-server localhost:9092 --group group2
```

Both will see **every message** produced.

Ip addr show

My vm ip 192.168.114.128

If you want to delete a topic

sudo /opt/kafka/bin/kafka-topics.sh --describe --topic test-topic1 --bootstrap-server localhost:9092

Configure Multiple Brokers

We'll create **3 copies** of `server.properties`.

`sudo cat /opt/kafka/config/server-1.properties`

```
broker.id=1
listeners=PLAINTEXT://:9092
log.dirs=/tmp/kafka-logs-1
zookeeper.connect=localhost:2181
```

sudo cat /opt/kafka/config/server-2.properties

```
broker.id=2
listeners=PLAINTEXT://:9093
log.dirs=/tmp/kafka-logs-2
zookeeper.connect=localhost:2181
```

sudo cat /opt/kafka/config/server-3.properties

```
broker.id=3
listeners=PLAINTEXT://:9094
log.dirs=/tmp/kafka-logs-3
zookeeper.connect=localhost:2181
```

With these configs:

- Broker **1** runs on port **9092**.
- Broker **2** runs on port **9093**.
- Broker **3** runs on port **9094**.
- All connect to **Zookeeper at localhost:2181**.
- Each broker writes its logs to a separate directory.

Step 4: Start the Brokers

In separate terminals (or append `&`):

```
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 &
```

Now you have **3 Kafka brokers running** on ports `9092, 9093, 9094`.

Step 5: Verify Cluster

List brokers via metadata:

```
bin/zookeeper-shell.sh localhost:2181 ls /brokers/ids
```

Should show:

```
[1, 2, 3]
```

Step 6: Create a Topic with Replication

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

Step 7: Check Topic Details

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

Example output:

```
Topic: test-topic  PartitionCount:3  ReplicationFactor:3
    Partition: 0  Leader: 1  Replicas: 1,2,3  Isr: 1,2,3
    Partition: 1  Leader: 2  Replicas: 2,3,1  Isr: 2,3,1
    Partition: 2  Leader: 3  Replicas: 3,1,2  Isr: 3,1,2
```

- **Leader** = which broker is handling writes for that partition.
- **Replicas** = copies stored across brokers.
- **ISR (In-Sync Replicas)** = healthy replicas.

Let's break it down **line by line**

Topic Summary

```
Topic: test-topic  PartitionCount:3  ReplicationFactor:3
```

- **Topic: test-topic** → The name of your topic.
- **PartitionCount:3** → The topic has **3 partitions** (Kafka splits topic data across partitions for parallelism).
- **ReplicationFactor:3** → Each partition has **3 copies** across your 3 brokers.

Partition 0

```
Partition: 0  Leader: 1  Replicas: 1,2,3  Isr: 1,2,3
```

- **Partition: 0** → First partition of the topic.
- **Leader: 1** → Broker **1** is responsible for handling all **reads/writes** for this partition.
- **Replicas: 1,2,3** → This partition is stored on **all 3 brokers**.
- **ISR (In-Sync Replicas): 1,2,3** → All replicas (1,2,3) are currently up-to-date with the leader.

Partition 1

```
Partition: 1  Leader: 2  Replicas: 2,3,1  Isr: 2,3,1
```

- **Leader: 2** → Broker **2** is handling reads/writes for Partition 1.
- **Replicas: 2,3,1** → Partition 1 data exists on brokers 2, 3, and 1.
- **ISR: 2,3,1** → All replicas are synced and healthy.

Partition 2

```
Partition: 2  Leader: 3  Replicas: 3,1,2  Isr: 3,1,2
```

- **Leader: 3** → Broker **3** handles Partition 2.
- **Replicas: 3,1,2** → Partition 2 data exists on brokers 3, 1, and 2.
- **ISR: 3,1,2** → All replicas are in sync.

What this means

- The topic is split into **3 partitions**.
- Each partition has **3 replicas** (one on each broker).
- Leadership is **balanced across brokers**:
    - Broker 1 leads Partition 0
    - Broker 2 leads Partition 1
    - Broker 3 leads Partition 2

- **All replicas are in sync (ISR = 3/3)** → the cluster is healthy.

Why this matters

- **High availability**: If broker 1 crashes, partitions 1 & 2 still have leaders (2 & 3). Partition 0 will elect a new leader from ISR (2 or 3).
- **Scalability**: Producers and consumers can work in parallel across partitions.
- **Fault tolerance**: Since replication factor = 3, losing one broker does not lose data.

Step 8: Test Producer and Consumer

Producer (send messages):

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

Consumer (read messages):

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

Messages should flow across the cluster.

At this point, you have a **3-broker Kafka cluster** running locally on RHEL.
If one broker goes down, messages still remain available via replicas.

Stop broker 1

```
sudo pkill -f "kafka.Kafka config/server-1.properties"
```

(or if using systemd)

```
sudo systemctl stop kafka1
```

Step 3: Verify new cluster state

Run:

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

Now you'll see something like:

```
Topic: test-topic  PartitionCount:3  ReplicationFactor:3
    Partition: 0  Leader: 2  Replicas: 1,2,3  Isr: 2,3
    Partition: 1  Leader: 2  Replicas: 2,3,1  Isr: 2,3
    Partition: 2  Leader: 3  Replicas: 3,1,2  Isr: 3,2
```

What changed?

- **Partition 0**: Broker 1 was the leader, but since it failed → Broker 2 is elected as new leader (from ISR).
- **ISR shrinks**: Broker 1 is gone, so `Isr: 2,3` instead of `1,2,3`.
- **Partitions 1 & 2**: Still led by 2 and 3, but ISR no longer includes 1.

Step 4: Why Kafka still works

- Producers & consumers can continue sending/reading messages.
- No data loss → because replication factor = 3 (copies existed on 2 and 3).
- Kafka automatically rebalances leadership using **ISR (In-Sync Replicas)**.

Step 5: Restart broker 1

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

After it catches up, ISR will expand back to `1,2,3`.

Key Takeaways

- Kafka elects a new leader from ISR when a broker fails.
- ISR ensures only **fully up-to-date replicas** can become leaders (avoiding stale data).

- Cluster remains available unless **all replicas of a partition fail**.

Produce messages with keys

Run the **console producer** with `--property parse.key=true` and a **key.separator** (e.g., `:`):

```
bin/kafka-console-producer.sh \
  --topic test-topic \
  --bootstrap-server localhost:9092 \
  --property parse.key=true \
  --property key.separator=:
```

Now, when you type messages, use the format:

```
key1:Hello from key1
key2:Hello from key2
key3:Hello from key3
```

- `key1` is the **message key**
- `Hello from key1` is the **message value**

Kafka's default partitioner will take the key, apply a hash function, and **route the message to a partition deterministically**.
Same key = always same partition.

Step 3: Consume messages with keys

To see which key went where:

```
bin/kafka-console-consumer.sh \
  --topic test-topic \
  --from-beginning \
  --bootstrap-server localhost:9092 \
  --property print.key=true \
  --property print.partition=true
```

Example output:

```
key1 Hello from key1    Partition:0
key2 Hello from key2    Partition:1
key3 Hello from key3    Partition:2
```

How partitioning works

- If **key = null**, Kafka distributes messages round-robin across partitions.
- If **key is provided**, Kafka always sends the message to the **same partition for that key** (hash-based).
- This guarantees **ordering per key**.

Kafka Connect with Postgres Connector

Step 1 — Install PostgreSQL on RHEL

```
# Update system packages
sudo dnf update -y

# Install PostgreSQL server & contrib
sudo dnf install -y postgresql-server postgresql-contrib

# Initialize PostgreSQL database
sudo postgresql-setup --initdb

# Start and enable service
sudo systemctl enable postgresql
sudo systemctl start postgresql
```

Step 2 — Configure PostgreSQL for Password Auth

```
sudo vi /var/lib/pgsql/data/pg_hba.conf
```

Change lines like this:

```
local   all    all                                    md5
```

```
host    all    all             127.0.0.1/32            md5
host    all    all             ::1/128                 md5
```

Restart PostgreSQL:

```
sudo systemctl restart postgresql
```

Step 3 — Create Database, User, and Table

```
# Login as postgres superuser
sudo -u postgres psql
```

Inside psql:

```
CREATE USER kafkauser WITH PASSWORD 'kafkapass';
CREATE DATABASE kafkadb OWNER kafkauser;

\c kafkadb

CREATE TABLE customer (
    id SERIAL PRIMARY KEY,
    firstname TEXT,
    lastname  TEXT,
    email     TEXT
);

INSERT INTO customer (firstname, lastname, email)
VALUES ('Alice','Wonder','alice@example.com'),
       ('Bob','Marley','bob@example.com');

GRANT CONNECT ON DATABASE kafkadb TO kafkauser;
GRANT USAGE ON SCHEMA public TO kafkauser;
GRANT SELECT ON customer TO kafkauser;

\q
```

Test connection:

```
PGPASSWORD=kafkapass psql -U kafkauser -d kafkadb -h
localhost -c "SELECT * FROM customer;"
```

Step 4 — Download & Install Kafka

```
# Download Kafka (3.8.0 as example)
curl -O https://downloads.apache.org/kafka/3.8.0/
kafka_2.13-3.8.0.tgz

# Extract
tar -xvzf kafka_2.13-3.8.0.tgz
sudo mv kafka_2.13-3.8.0 /opt/kafka
```

Step 5 — Start Zookeeper & Kafka Broker

```
# Start Zookeeper
/opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/kafka/
config/zookeeper.properties

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

Step 6 — Create Kafka Topic for Postgres Table

```
/opt/kafka/bin/kafka-topics.sh \
  --create \
  --topic test_customer \
  --bootstrap-server localhost:9092 \
  --partitions 1 \
  --replication-factor 1
```

Check topic:

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

At this point you have:
PostgreSQL installed and ready
Database + user + `customer` table
Kafka installed and running
Kafka topic `test_customer` created

Step 7 — Install JDBC Driver & Connector

1. Create plugin folder:

```
sudo mkdir -p /opt/kafka/plugins/jdbc
```

2. Download PostgreSQL JDBC driver

```
curl -o /tmp/postgresql-42.6.0.jar https://
jdbc.postgresql.org/download/postgresql-42.6.0.jar
sudo mv /tmp/postgresql-42.6.0.jar /opt/kafka/plugins/jdbc/
```

3. Download JDBC Connector (Confluent):

```
curl -L -o /tmp/kafka-connect-jdbc.zip https://
d1i4a15mxbxib1.cloudfront.net/api/plugins/confluentinc/kafka-
connect-jdbc/latest/confluentinc-kafka-connect-jdbc-
latest.zip

sudo unzip -o /tmp/kafka-connect-jdbc.zip -d /opt/kafka/
plugins/jdbc
```

OR
Download Manually

Move file from downloads to jdbc

sudo mv confluentinc-kafka-connect-jdbc-10.8.4 /opt/kafka/plugins/jdbc

4. Add plugin path in Kafka Connect config:

```
echo "plugin.path=/opt/kafka/plugins" | sudo tee -a /opt/
kafka/config/connect-distributed.properties
```

Step 8 — Start Kafka Connect Worker

```
/opt/kafka/bin/connect-distributed.sh /opt/kafka/config/
connect-distributed.properties &
```

Check REST API:

```
curl http://localhost:8083/
```

Step 9 — Create JDBC Source Connector Config

```
Sudo -c 'cat > /opt/kafka/config/jdbc-source.json <<EOF
{
  "name": "jdbc-source-connector",
  "config": {
    "connector.class":
"io.confluent.connect.jdbc.JdbcSourceConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:postgresql://localhost:5432/
kafkadb",
    "connection.user": "kafkauser",
    "connection.password": "kafkapass",
    "table.whitelist": "customer",
    "mode": "incrementing",
    "incrementing.column.name": "id",
    "topic.prefix": "test_",
    "poll.interval.ms": "1000"
  }
}
EOF'
```

Deploy connector:

```
curl -X POST -H "Content-Type: application/json" \
    --data @/opt/kafka/config/jdbc-source.json \
    http://localhost:8083/connectors
```

Check status:

```
curl http://localhost:8083/connectors/jdbc-source-connector/
status
```

Step 10 — Consume Data from Kafka

```
/opt/kafka/bin/kafka-console-consumer.sh \
    --bootstrap-server localhost:9092 \
    --topic test_customer \
    --from-beginning
```

You should now see rows from the Postgres `customer` table inside Kafka.

Step 11 — Prepare PostgreSQL Sink Table

We'll create a table to store Kafka messages coming **from a Kafka topic**.

```
sudo -u postgres psql -d kafkadb
```

Inside psql:

```
CREATE TABLE customer_sink (
    id SERIAL PRIMARY KEY,
    firstname TEXT,
    lastname  TEXT,
    email     TEXT
);

GRANT INSERT, UPDATE, SELECT ON customer_sink TO kafkauser;

\q
```

Step 12 — Produce a Test Message into Kafka

We'll manually put some messages into a new Kafka topic `test_customer_sink`.

```
/opt/kafka/bin/kafka-topics.sh \
  --create \
  --topic test_customer_sink \
  --bootstrap-server localhost:9092 \
  --partitions 1 \
  --replication-factor 1
```

Start a producer:

```
/opt/kafka/bin/kafka-console-producer.sh \
    --broker-list localhost:9092 \
    --topic test_customer_sink
```

Type a JSON message (then press Enter):

```
{"firstname":"Charlie","lastname":"Brown","email":"charlie@ex
ample.com"}
```

(Leave the producer running for now or exit with `Ctrl+C`.)

Step 13 — Create JDBC Sink Connector Config

```
sudo tee /opt/kafka/config/jdbc-sink.json > /dev/null <<EOF
{
  "name": "jdbc-sink-connector",
  "config": {
    "connector.class":
"io.confluent.connect.jdbc.JdbcSinkConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:postgresql://localhost:5432/
kafkadb",
    "connection.user": "kafkauser",
    "connection.password": "kafkapass",
    "topics": "test_customer_sink",
    "auto.create": "false",
    "auto.evolve": "false",
    "insert.mode": "insert",
    "table.name.format": "customer_sink"
  }
}
EOF
```

This will save the file at:
`/opt/kafka/config/jdbc-sink.json`

You can verify with:

```
sudo cat /opt/kafka/config/jdbc-sink.json
```

Deploy connector:

```
curl -X POST -H "Content-Type: application/json" \
    --data @~/jdbc-sink.json \
    http://localhost:8083/connectors
```

Check status:

```
curl http://localhost:8083/connectors/jdbc-sink-connector/
status
```

Step 14 — Verify Data in PostgreSQL

Now check if Kafka messages landed in PostgreSQL:

```
PGPASSWORD=kafkapass psql -U kafkauser -d kafkadb -h
localhost -c "SELECT * FROM customer_sink;"
```

You should see:

```
 id | firstname | lastname |        email
----+-----------+----------+---------------------
  1 | Charlie   | Brown    | charlie@example.com
```

KAFKA MONITORING USING Prometheus And Grafana

2. start zookeeper

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

leave this running in one terminal (or run it as a systemd service if you want it permanent).

3. start kafka broker (with jmx exporter)

1. download jmx exporter:

```
cd /opt
wget https://repo1.maven.org/maven2/io/prometheus/jmx/
jmx_prometheus_javaagent/0.21.0/
jmx_prometheus_javaagent-0.21.0.jar
```

2. create a jmx config:

```
cat <<EOF > /opt/kafka-jmx.yml
rules:
  - pattern: "kafka.server<type=(.+), name=(.+)><>Value"
    name: "kafka_\$1_\$2"
    type: GAUGE
EOF
```

3. start kafka with jmx agent (port **7071**):

```
cd /opt/kafka
export KAFKA_OPTS="-javaagent:/opt/
jmx_prometheus_javaagent-0.21.0.jar=7071:/opt/kafka-jmx.yml"
bin/kafka-server-start.sh config/server.properties
```

4. install prometheus

```
cd /opt
wget https://github.com/prometheus/prometheus/releases/
download/v2.52.0/prometheus-2.52.0.linux-amd64.tar.gz
tar -xvzf prometheus-2.52.0.linux-amd64.tar.gz
mv prometheus-2.52.0.linux-amd64 prometheus
```

create prometheus config:

```
sudo cat <<EOF > /opt/prometheus/prometheus.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'kafka'
    static_configs:
      - targets: ['localhost:7071']
EOF
```

run prometheus:

```
cd /opt/prometheus
./prometheus –config.file=prometheus.yml
```

open http://localhost:9090

Check If 9090 is Occupied

ss -lntp | grep 9090. or

netstat -tulnp | grep 9090

Run on port 9095

```
cd /opt/prometheus
./prometheus --config.file=prometheus.yml --web.listen-address="0.0.0.0:9095"
```

curl -v http://localhost:9095

5. install grafana

sudo wget https://dl.grafana.com/oss/release/grafana-12.1.1.linux-arm64.tar.gz

sudo tar -xvzf grafana-12.1.1.linux-arm64.tar.gz

sudo mv grafana-12.1.1 grafana

cd /opt/grafana

sudo ./bin/grafana-server

http://localhost:3000

login: **admin / admin**

6. connect grafana to prometheus

1. go to **grafana → settings → data sources → add data source → prometheus**
2. set url: `http://localhost:9095`
3. save & test

7. import kafka dashboards

- go to **dashboards → import**
- use **id 7587** (kafka overview)
- use **id 10466** (consumer lag)

summary of services

- **zookeeper** → `bin/zookeeper-server-start.sh config/zookeeper.properties`
- **kafka** → `bin/kafka-server-start.sh config/server.properties` (with jmx agent)
- **prometheus** → `/opt/prometheus/prometheus --config.file=prometheus.yml`
- **grafana** → `systemctl start grafana-server`

Step-by-Step Setup: Prometheus + Postgres Exporter(Optional)

1. install postgres exporter

```
cd /opt
wget https://github.com/prometheus-community/postgres_exporter/releases/download/v0.15.0/postgres_exporter-0.15.0.linux-amd64.tar.gz
tar -xvzf postgres_exporter-0.15.0.linux-amd64.tar.gz
mv postgres_exporter-0.15.0.linux-amd64 postgres_exporter
```

2. create monitoring user in postgres

switch to postgres and create a dedicated user:

sudo -i -u postgres psql

inside psql:

-- create db and user

CREATE DATABASE kafkadb;

CREATE USER kafkauser WITH PASSWORD 'kafkapass';

-- grant privileges

GRANT ALL PRIVILEGES ON DATABASE kafkadb TO kafkauser;

-- give monitoring role (needed for exporter)

ALTER USER kafkauser WITH SUPERUSER;

exit with \q.

2. configure postgres exporter

go to exporter folder:

cd /opt/postgres_exporter

set connection string (DSN) for your kafka DB:

export DATA_SOURCE_NAME="postgresql://kafkauser:kafkapass@localhost:5432/kafkadb?sslmode=disable"

run exporter on port 9187:

/opt/postgres_exporter/postgres_exporter

./postgres_exporter —web.listen-address="0.0.0.0:9187"

test:

curl http://localhost:9187/metrics | head

3. add scrape job in prometheus

sudo tee /opt/prometheus/prometheus.yml > /dev/null <<'EOF'

global:

  scrape_interval: 15s

scrape_configs:

  - job_name: 'kafka'

    static_configs:

      - targets: ['localhost:7071']

  - job_name: 'postgres-kafka'

    static_configs:

      - targets: ['localhost:9187']

EOF

restart prometheus:

```
sudo systemctl restart prometheus
sudo systemctl status prometheus
```

4. grafana dashboard

- go to **Grafana → Dashboards → Import**
- import dashboard ID **9628** (Postgres Exporter)
- set datasource = Prometheus
- you'll see stats for `kafkadb`.

UNINSTALL KAFKA

Stop Kafka and Zookeeper

```
sudo systemctl stop kafka
sudo systemctl stop zookeeper
sudo systemctl disable kafka
sudo systemctl disable zookeeper
```

If you didn't use systemd services, kill processes manually:

```
sudo pkill -f kafka
sudo pkill -f zookeeper
```

Remove Kafka package (if installed via YUM/RPM)

```
sudo yum remove kafka -y
```

Or, on RHEL 8+:

```
sudo dnf remove kafka -y
```

Remove Kafka directories, logs, and configs

```
sudo rm -rf /opt/kafka          # If installed manually
sudo rm -rf /var/lib/kafka      # Data directory
sudo rm -rf /var/log/kafka      # Logs
sudo rm -rf /etc/kafka          # Configs
sudo rm -rf /tmp/kafka-logs     # Default log dir
sudo rm -rf /tmp/zookeeper      # Zookeeper data/logs
```

Verify removal

```
which kafka-server-start.sh
# Should return nothing

sudo systemctl status kafka
# Should show inactive or not-found
```

If you're on **RHEL 8+**, replace `yum remove -y kafka` with:

```
sudo dnf remove -y kafka
```