Exercise: monitoring stack for **Confluent Kafka** using **Prometheus** (for metric collection) and **Grafana** (for visualization).

Objective

By the end of this exercise, you will have:

1. A running Confluent Kafka Cluster (Single Broker).
2. A JMX Exporter attached to Kafka to expose metrics.
3. Prometheus scraping the metrics.
4. A Grafana Dashboard visualizing real-time throughput, lag, and broker health.

Prerequisites

- **Docker** and **Docker Compose** installed on your machine.
- Basic familiarity with command-line operations.
- Internet access (to download Docker images and the JMX agent jar).

To install Docker and Docker Compose on Ubuntu, follow these steps:

1. Update System Packages:

sudo apt update
sudo apt upgrade -y

2. Install Prerequisite Packages:

sudo apt install -y ca-certificates curl gnupg lsb-release

3. Add Docker's Official GPG Key:

sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

4. Set up the Docker Repository:

**echo **
**"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu **
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

5. Install Docker Engine and Docker Compose Plugin:

sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

6. Verify Docker Installation:

sudo docker run hello-world

This command should download a test image and run a container, printing a "Hello from Docker!" message if successful.

7. Add Your User to the docker Group (Optional, but Recommended):

To run Docker commands without sudo, add your user to the docker group:

sudo usermod -aG docker ${USER}

You will need to log out and log back in for this change to take effect.

8. Verify Docker Compose Installation:

docker compose version

This command should display the installed version of Docker Compose.

Step 1: Project Workspace Setup

Create a directory for your project and the necessary subfolders to hold configurations.

mkdir kafka-monitoring-lab

cd kafka-monitoring-lab

mkdir -p config/prometheus jars

Step 2: Download JMX Prometheus Agent

The JMX Exporter acts as a "bridge" between Kafka's internal Java metrics (JMX) and Prometheus. We need to download the Java agent .jar file.

1. Download the jar into your jars folder:

curl -o jars/jmx_prometheus_javaagent.jar \

https://repo1.maven.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/0.19.0/jmx_prometheus_javaagent-0.19.0.jar

---

Step 3: Create Configuration Files

A. JMX Exporter Config (config/kafka_jmx_config.yml)

Create this file to tell the agent which metrics to export and how to format them.

File: kafka-monitoring-lab/config/kafka_jmx_config.yml

```
lowercaseOutputName: true
lowercaseOutputLabelNames: true
rules:
  # Broker Topic Metrics (BytesIn, BytesOut, MessagesIn, etc.)
  - pattern: kafka.server<type=BrokerTopicMetrics, name=(.+), topic=(.+)><>Count
    name: kafka_server_brokertopicmetrics_$1_total
    type: COUNTER
    labels:
      topic: "$2"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=(.+)><>Count
    name: kafka_server_brokertopicmetrics_$1_total
    type: COUNTER

  # Request Handling (Produce, Fetch, etc.)
  - pattern: kafka.network<type=RequestMetrics, name=RequestsPerSec,
request=(.+)><>Count
    name: kafka_network_requestmetrics_requests_total
    type: COUNTER
    labels:
      request: "$1"

  # JVM Metrics (Memory, GC, Threads)
  - pattern: "java.lang<type=Memory><HeapMemoryUsage>used"
```

```
    name: jvm_memory_bytes_used
    type: GAUGE
  - pattern: "java.lang<type=GarbageCollector, name=(.+)><CollectionCount>Count"
    name: jvm_gc_collection_seconds_count
    type: COUNTER
    labels:
      gc: "$1"
```

B. Prometheus Config (config/prometheus.yml)

Create this file to tell Prometheus where to find the Kafka metrics.

File: kafka-monitoring-lab/config/prometheus.yml

```
global:
  scrape_interval: 10s

scrape_configs:
  - job_name: 'kafka-broker'
    static_configs:
      - targets: ['kafka:7071']
        labels:
          env: 'dev'
```

Step 4: Create Docker Compose File

This file defines our entire stack. We are injecting the KAFKA_OPTS environment variable to attach the JMX agent to the Kafka process.

File: kafka-monitoring-lab/docker-compose.yml

```
version: '3.8'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
```

```
    hostname: zookeeper
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    image: confluentinc/cp-server:7.5.0
    hostname: kafka
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
      - "7071:7071" # Exposing JMX Exporter port
    volumes:
      - ./jars:/usr/share/jmx_exporter/
      - ./config/kafka_jmx_config.yml:/usr/share/jmx_exporter/config.yml
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_CONFLUENT_LICENSE_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_CONFLUENT_BALANCER_TOPIC_REPLICATION_FACTOR: 1
```

```
    # CRITICAL: Attach the Java Agent here
    KAFKA_OPTS: "-javaagent:/usr/share/jmx_exporter/jmx_prometheus_javaagent.jar=7071:/usr/share/jmx_exporter/config.yml"

  prometheus:
    image: prom/prometheus
    container_name: prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./config/prometheus.yml:/etc/prometheus/prometheus.yml
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    depends_on:
      - prometheus
```

Step 5: Launch the Stack

1. Start the services:

```
docker-compose up -d
```

2. Verify everything is running:

docker-compose ps

(Ensure kafka, zookeeper, prometheus, and grafana are all "Up")

3. **Verify Metrics Export**: Open your browser to http://localhost:7071.
    - You should see a long page of text metrics. If you see this, the JMX Exporter is working!
4. **Verify Prometheus Scrape**: Open http://localhost:9090/targets.
    - The state for kafka-broker should be **UP** (green).

Step 6: Generate Kafka Traffic

To see interesting charts, we need data moving through the system.

1. Enter the Kafka container:

docker exec -it kafka /bin/bash

2. Create a topic:

kafka-topics --bootstrap-server localhost:9092 --create --topic test-topic --partitions 3 --replication-factor 1

3. Run the performance test producer (generates load):

kafka-producer-perf-test \

--topic test-topic \

--num-records 500000 \

--record-size 1000 \

--throughput 5000 \

--producer-props bootstrap.servers=localhost:9092

(Let this run in the background or open a new terminal tab to keep it running)

Step 7: Configure Grafana

1. **Login**:
    - Go to http://localhost:3000
    - User: admin, Password: admin (Skip password change if asked)
2. **Add Data Source**:

- Go to **Connections (or Configuration)** > **Data Sources** > **Add data source**.
- Select **Prometheus**.
- Set URL to: http://prometheus:9090 (Note: we use the container name, not localhost).
- Click **Save & Test**. You should see "Data source is working".

3. **Import Dashboard**:
   - Go to **Dashboards** > **New** > **Import**.
   - In "Import via grafana.com", enter ID: **721** (A popular Kafka Overview dashboard) or **18276**.
   - Click **Load**.
   - Select your Prometheus data source in the dropdown.
   - Click **Import**.

Step 8: Analyze the Results

Look at your new Dashboard. You should now see:

- **Incoming Message Rate**: Spiking up to ~5k ops/sec (matches your perf-test).
- **Bytes In/Out**: Showing the network throughput.
- **Active Controller Count**: Should be 1.
- **Offline Partitions**: Should be 0.

Troubleshooting

- **Kafka container exits immediately**: Usually a memory issue. Increase Docker memory limit to at least 4GB.
- **Target DOWN in Prometheus**: Check http://localhost:7071. If that fails, check docker logs kafka. Look for "javaagent" errors.
- **No Data in Grafana**: Ensure the time range (top right) is set to "Last 5 minutes" to see the immediate traffic you generated.