## ! STOP!

## You are required to watch the walkthrough video for this week's assignment. In the video, I guide you through each command, showing both the input and the expected output. Since you are working with new technology for the first time, it’s important not to just run a command

## Week 7 Assignment: Diving into Apache Kafka

### Objective: Mastering the Fundamentals of Apache Kafka

In this assignment, you will gain hands-on experience with **Apache Kafka**, a distributed streaming platform designed for building real-time data pipelines and streaming applications. Kafka excels at handling high-throughput, low-latency data streams, making it a critical tool for large-scale, event-driven architectures. Through a series of exercises, you will learn how to create Kafka topics, produce and consume messages, and run performance tests to measure Kafka's throughput and scalability.

By the end of this assignment, you will:

* Understand how to create Kafka topics and manage message flows between producers and consumers.
* Gain practical experience in producing and consuming real-time data streams.
* Learn how to conduct performance tests to evaluate Kafka’s efficiency and scalability.
* Scale a Kafka instance and observe how partitioning and replication affect performance.

#### **1. Environment Initialization**

* Navigate to the Kafka directory and start the Docker containers:  
    
  cd dsc650-infra/bellevue-bigdatakafka
* Start Kafka

docker-compose up -d

* Open **two** terminal sessions and in each, access the Kafka container:

docker exec -it kafka\_kafka\_1 bash

* If you can’t access the Kafka container, it could be due to a container name change. In this cause use:

docker exec -it kafka-kafka-1 bash

#### **2. Topic Creation and Verification in Kafka (On One Terminal Only)**

In this section, you will create a Kafka topic named my-topic and verify its creation. Kafka topics act as message channels, where producers send data and consumers retrieve it.

**Exercise 1:** Create a Kafka topic named ‘my-topic’.

/opt/kafka\_2.13-2.8.1/bin/kafka-topics.sh --create --topic my-topic --bootstrap-server localhost:9092

**Exercise 2:** List the topics to verify that ‘my-topic’ has been successfully created.

/opt/kafka\_2.13-2.8.1/bin/kafka-topics.sh --list --bootstrap-server localhost:9092

**Deliverable:** Screenshot showing the ‘my-topic’ listed amongst the topics.

#### **3. Producing and Consuming Messages in Kafka**

You will now simulate a real-time data pipeline by starting a Kafka **producer** to send messages and a Kafka **consumer** to receive those messages. This demonstrates how Kafka brokers act as intermediaries between producers and consumers.

**Exercise 3:** In the first terminal, start a Kafka consumer:

/opt/kafka\_2.13-2.8.1/bin/kafka-console-consumer.sh --topic my-topic --from-beginning --bootstrap-server localhost:9092

**Exercise 4:** In the second terminal, start a Kafka producer:

/opt/kafka\_2.13-2.8.1/bin/kafka-console-producer.sh --topic my-topic --bootstrap-server localhost:9092

* Type some text into the producer and press ‘Enter’. Note that the text appears on the consumer terminal.

**Deliverable:** Screenshots of the producer terminal with your entered text and the consumer terminal showing the received message.

To exit the console and producer shells type CTRL + C.

Close your second terminal.

#### **4. Kafka Performance Tests**

In this section, you will perform **producer and consumer performance tests** on the my-topic topic to measure Kafka’s throughput. Performance testing helps you understand Kafka’s efficiency in handling large volumes of messages.

**Exercise 5:** Run a performance test on the producer using the Kafka producer performance test script with provided arguments:

/opt/kafka\_2.13-2.8.1/bin/kafka-producer-perf-test.sh --topic my-topic --num-records 50000 --record-size 100 --throughput 1000 --producer-props bootstrap.servers=localhost:9092 key.serializer=org.apache.kafka.common.serialization.StringSerializer value.serializer=org.apache.kafka.common.serialization.StringSerializer

**Exercise 6:** Following the producer test, run a consumer performance test on ‘my-topic’:

/opt/kafka\_2.13-2.8.1/bin/kafka-consumer-perf-test.sh --broker-list localhost:9092 --topic my-topic --messages 50000

**Deliverable:**

* Screenshots of both the producer and consumer performance test results.
* Discuss the meaning of the results.

#### **5. Expanding Kafka and Running Additional Performance Tests**

You will now scale Kafka to 3 instances and test how Kafka performs with **partitioned and replicated topics**. Partitioning and replication improve Kafka's fault tolerance and scalability, allowing more efficient distribution of messages across brokers.

* Exit the Kafka container and scale Kafka instances to 3:

exit

docker-compose scale kafka=3

* Re-enter the kafka\_kafka\_1 container:

docker exec -it kafka\_kafka\_1 bash

* If you can’t access the Kafka container, it could be due to a container name change. In this cause use:

docker exec -it kafka-kafka-1 bash

**Exercise 7:** Create a topic, this time partitioned and replicated across all three Kafka instances:

/opt/kafka\_2.13-2.8.1/bin/kafka-topics.sh --create --topic my-partitioned-topic --replication-factor 3 --partitions 3 --bootstrap-server localhost:9092

**Exercise 8:** Conduct the producer and consumer performance tests on the new topic, observing differences:

/opt/kafka\_2.13-2.8.1/bin/kafka-producer-perf-test.sh --topic my-partitioned-topic --num-records 50000 --record-size 100 --throughput 1000 --producer-props bootstrap.servers=localhost:9092 key.serializer=org.apache.kafka.common.serialization.StringSerializer value.serializer=org.apache.kafka.common.serialization.StringSerializer

Followed by:

/opt/kafka\_2.13-2.8.1/bin/kafka-consumer-perf-test.sh --broker-list localhost:9092 --topic my-partitioned-topic --messages 50000

**Deliverable:**

* Screenshots of the performance tests on the partitioned topic.
* Include your observations on performance variations between a single Kafka instance and the scaled setup.

When scaling up Kafka, you might expect improved performance due to increased parallelism and distribution of data across brokers. However, you may not see a significant performance increase in this test. Why is that? Consider the role of **replication** and **partitioning**: how did these change between the original test and the scaled setup? What impact do they have on performance, especially in terms of message distribution and redundancy?

Additionally, think about the fact that all brokers are using **shared storage**. How might this affect IOPS (input/output operations per second), and what could be the potential bottlenecks related to storage speed and resource contention? Analyze how these factors play a role in your test results.

## Shutting Down

Ensure all Docker containers are turned off with docker-compose down for each directory. If you’re using google cloud, please shut down your virtual machine to preserve cloud costs.