**Kafka Integration with Prometheus and Grafana**

In this task, we set up a complete Kafka environment using Docker Compose, enabling efficient topic creation, message production, and consumption. We created multiple Kafka topics with varying partitions to demonstrate scalability and parallel processing. Using Kafka console tools, we simulated real-time message flow between producers and consumers. For monitoring, we integrated Prometheus and Grafana, allowing us to visualize broker and topic metrics in real-time. By importing a pre-built Kafka dashboard in Grafana, we gained valuable insights into Kafka cluster health and performance. This setup demonstrates a robust and observable Kafka architecture suitable for production-like environments.

**Architecture Diagram :**

A diagram of a company

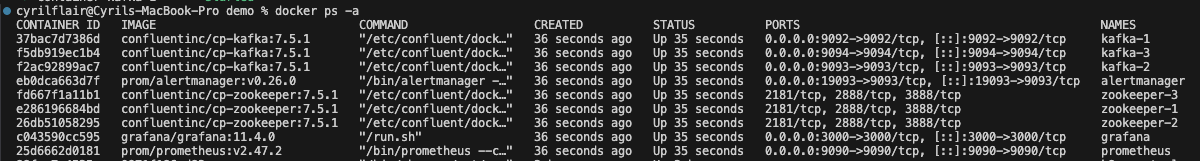
AI-generated content may be incorrect.

#**docker-compose up -d** : Starts all services defined in docker-compose.yml in detached (background) mode

A black screen with white text

AI-generated content may be incorrect.

#**docker ps -a** : Lists all containers (running, stopped, and exited) on your system



#**kafka-topics --bootstrap-server localhost:9092 --create --topic testtopic1 --partitions 2 --replication-factor 1** : Creates a Kafka topic named testtopic1 with 2 partitions and replication factor of 1 using the broker at localhost:9092



# **kafka-topics --bootstrap-server localhost:9092 --create --topic testtopic2 --partitions 3 --replication-factor 1** : Creates a Kafka topic named testtopic2 with 3 partitions and a replication factor of 1 using the broker at localhost:9092



# **kafka-topics --bootstrap-server localhost:9092 --list** : Lists all Kafka topics available on the broker at localhost:9092



# **kafka-console-producer --bootstrap-server localhost:9092 --topic testtopic1** : Starts a Kafka producer to send messages to the topic testtopic1 on the broker at localhost:9092



# **kafka-console-consumer --bootstrap-server localhost:9092 --topic testtopic1 --from-beginning** : Starts a Kafka console consumer that reads all messages from the beginning of the testtopic1 topic on localhost:9092



**Prometheus :**

For Prometheus URL: <http://localhost:9090/targets?search=>

The Prometheus **Targets** page, accessible at <http://localhost:9090/targets>, provides a real-time view of all the active scrape targets currently being monitored. Each Kafka broker (running with a JMX exporter) is listed with:

* **Endpoint**: The exposed metrics URL
* **Status**: Health status of the target
* **Labels**: Metadata for identification, including instance and job labels
* **Last Scrape**: The time since Prometheus last successfully scraped the target
* **Scrape Duration**: How long the last scrape took
* **Error**: Any issues encountered during scraping

This dashboard is essential for verifying that Prometheus is correctly scraping metrics from all Kafka components and ensures visibility into system health and performance

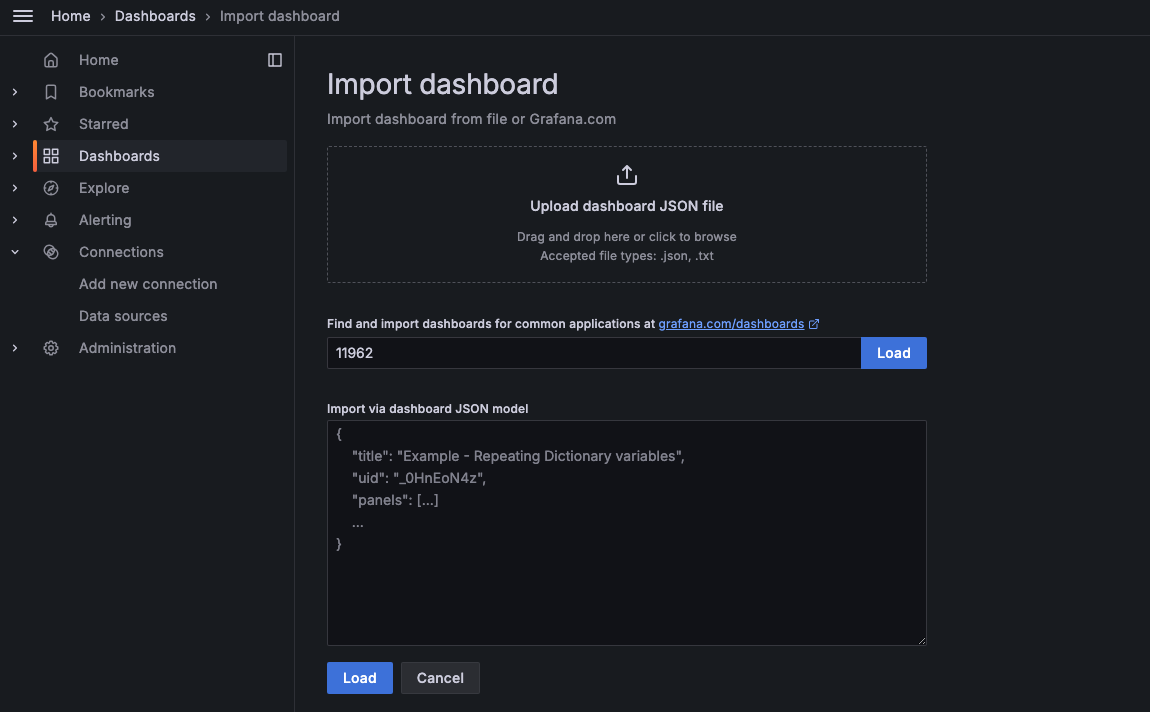
A screenshot of a computer

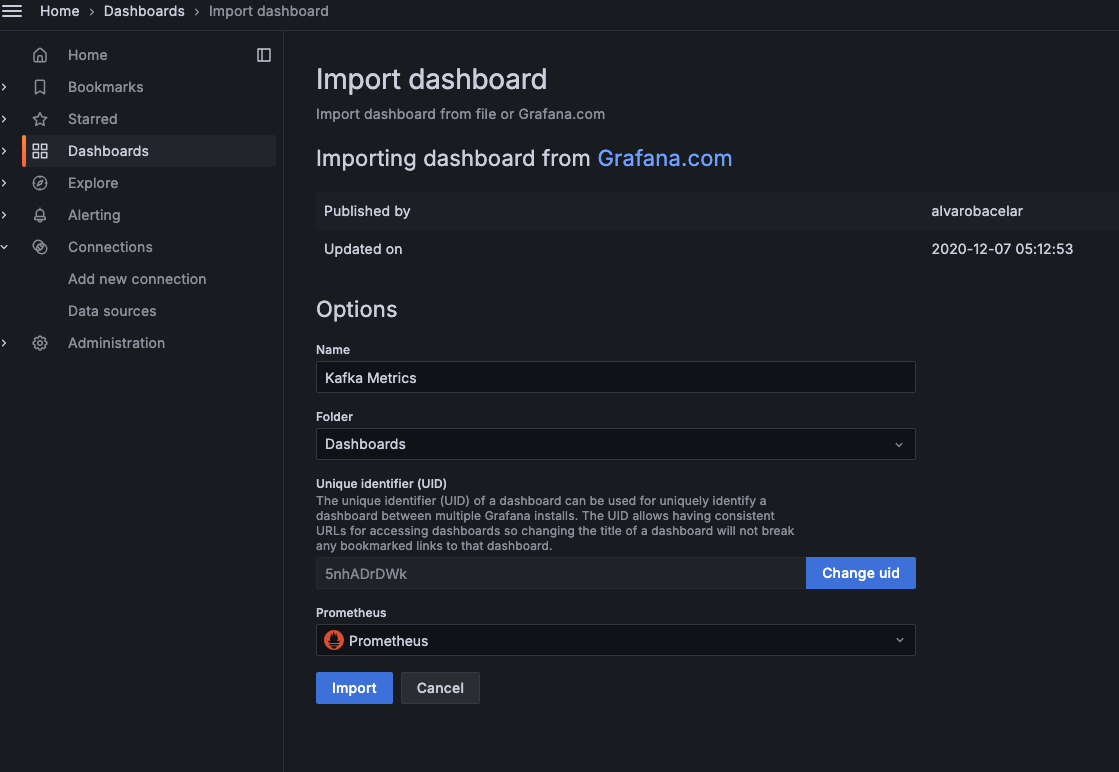
AI-generated content may be incorrect.

Grafana:

**It’s a data visualization and monitoring tool. It lets you create beautiful, real-time dashboards**

Click Dashboards and import 11962





#11962

A screenshot of a computer

AI-generated content may be incorrect.

This is a **Kafka monitoring dashboard in Grafana**, powered by **Prometheus** as the data source. It visualizes the **health and performance of a Kafka cluster**.

**Cluster Health**

Shows high-level Kafka cluster statistics:

| **Metric** | **Meaning** |
| --- | --- |
| **Brokers Count (3)** | Number of active Kafka brokers. |
| **Active Controllers (1)** | One broker is acting as the Kafka controller (expected = 1). |
| **Total # Topics (5)** | Total number of topics in the cluster. |
| **Under Replicated Partitions (0)** | All partitions have full replication — healthy. |
| **Offline Partitions (0)** | No partitions are offline — healthy. |
| **Partitions (173)** | Total number of partitions across all topics. |

**Kafka Integration with Mirror Maker**

In this task, we simulate real-time Kafka topic replication using MirrorMaker 2. We begin by launching our Kafka Source and Target clusters using Docker Compose. Once the environment is up, we create a topic (test-topic) in the source Kafka cluster and use a console producer to send sample messages. MirrorMaker handles the replication of this topic and its messages to the target cluster. We validate the success of this replication by listing topics and consuming messages from the target cluster. Throughout the process, MirrorMaker logs are monitored to observe the replication activity.

**Architecture Diagram :**

A diagram of a cluster

AI-generated content may be incorrect.

# **docker exec -it kafka-source kafka-topics.sh --create --topic test-topic \**

**--bootstrap-server kafka-source:9092 --partitions 1 --replication-factor 1**

# Creates a Kafka topic named test-topic with 1 partition and a replication factor of 1 inside the kafka-source container using its internal broker address kafka-source:9092



# **docker exec -it kafka-source kafka-console-producer.sh \**

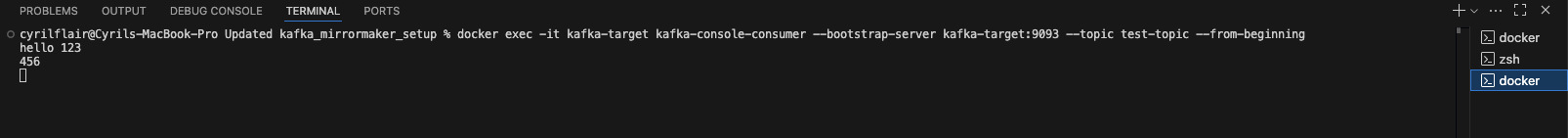
**--broker-list kafka-source:9092 --topic test-topic**

# Starts a Kafka console producer inside the kafka-source container to send messages to the test-topic using the broker at kafka-source:9092



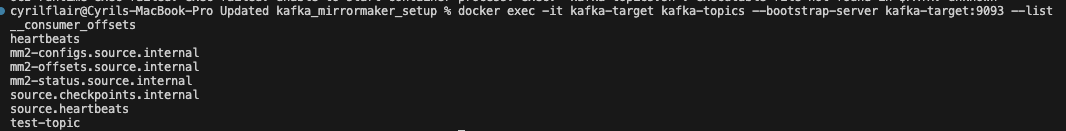
# **docker exec -it kafka-target kafka-console-consumer --bootstrap-server kafka-target:9093 --topic test-topic --from-beginning**

# Starts a Kafka console producer inside the kafka-source container to send messages to the test-topic using the broker at kafka-source:9092



# **docker exec -it kafka-target kafka-topics --bootstrap-server kafka-target:9093 --list**

# Lists all Kafka topics in the kafka-target container using the broker at kafka-target:9093



# **docker logs -f mirror-maker**

# Tails the real-time logs of the mirror-maker container, showing its output

A screenshot of a computer screen

AI-generated content may be incorrect.