**Task 2 Documentation**

**Task 2: Spin off Kafka server using Docker**

* Kafka and Zookeeper Containers Running: Verify that both Kafka and Zookeeper containers are running without errors.
* Kafka Listening on Port 9092: Check that Kafka is accessible on port 9092.
* Zookeeper Listening on Port 2181: Ensure Zookeeper is accessible on port 2181.
* Kafka Logs: Check Kafka logs for any errors or issues during startup.

**Subtask 1: Create source connectors to read from Postgresql**

* Kafka Connect Container Running: Verify that the Kafka Connect container is running without errors.
* Source Connector Deployment: Confirm that the source connector is deployed successfully.
* Connector Status: Check the status of the source connector to ensure it is running and not in a failed state.
* Data Flowing to Kafka: Verify that data is being read from the Postgres and published to the designated Kafka topic(s).

**Subtask 2: Create sink connector to dump into Postgresql**

* Sink Connector Deployment: Confirm that the sink connector is deployed successfully.
* Connector Status: Check the status of the sink connector to ensure it is running and not in a failed state.
* Data Written to Postgres: Verify that data from the Kafka topic(s) is being written to the Postgres tables as expected.
* No Errors in Logs: Ensure there are no errors in the Kafka Connect logs related to the sink connector.

What is Kafka?

**Kafka is open source, distributed message streaming platform that uses publish and subscribe mechanism to stream the records.**

* **Messaging system:**

1. It is responsible for transferring data from one application to another so the application can focus on data without getting bogged down on data transferring only.
2. point to point messaging system
3. a particular message can be received by max only one receiver only.

* point to point messaging system

1. A particular message can be received by max only one receiver only.
2. When receiver receives the message, it will send acknowledgement back to the sender.
3. Publish-subscribe messaging system

A particular message can be received by more than one receiver.

when subscriber receives the message, it doesn’t send the acknowledgment to the publisher.

**Topics**: The stream of messages belonging to a particular category.

unique name should be there for each topic.

**Partition**s:

Topics are splits in partition

All messages in partitions are ordered and immutable.

Each message withing partitions has a unique id known as OFFSET.

**Replica and Replications**:

replica is backup of partitions.

Replicas are never read or write data.

They are used to prevent data loss. (Fault Tolerance)

**Producers**:

Applications which write/publish data to topics within cluster using producing APIs.

Producers can write data either on topic level or specific partitions of the topic.

**Consumers**:

Are the application who consume data from topic.

consumers can read data either on topic level or specific partitions of the topic.

consumers are always associated with exactly one consumer group.

**Broker**:

It is simple software who maintains and manage published message.

Also known as Kafka servers.

A set of brokers who are communicating with each other to perform management and maintenance task are collectively known as **Kafka clusters**.

**Zookeepers**:

It is used to monitor Kafka cluster and coordinate with each broker.

Keeps the metadata information related to Kafka cluster in the form of key-value pair.

Metadata includes:

1.Configurations information

2. Health status of each broke

**Features of Kafka**:

1.Scalable

2.Fault tolerance

3.Durable (Kafka uses "Durable commit log" which means messages persists on disk asap.)

**Kafka produces 5 APIs**;

Producers APIs

Consumer APIs

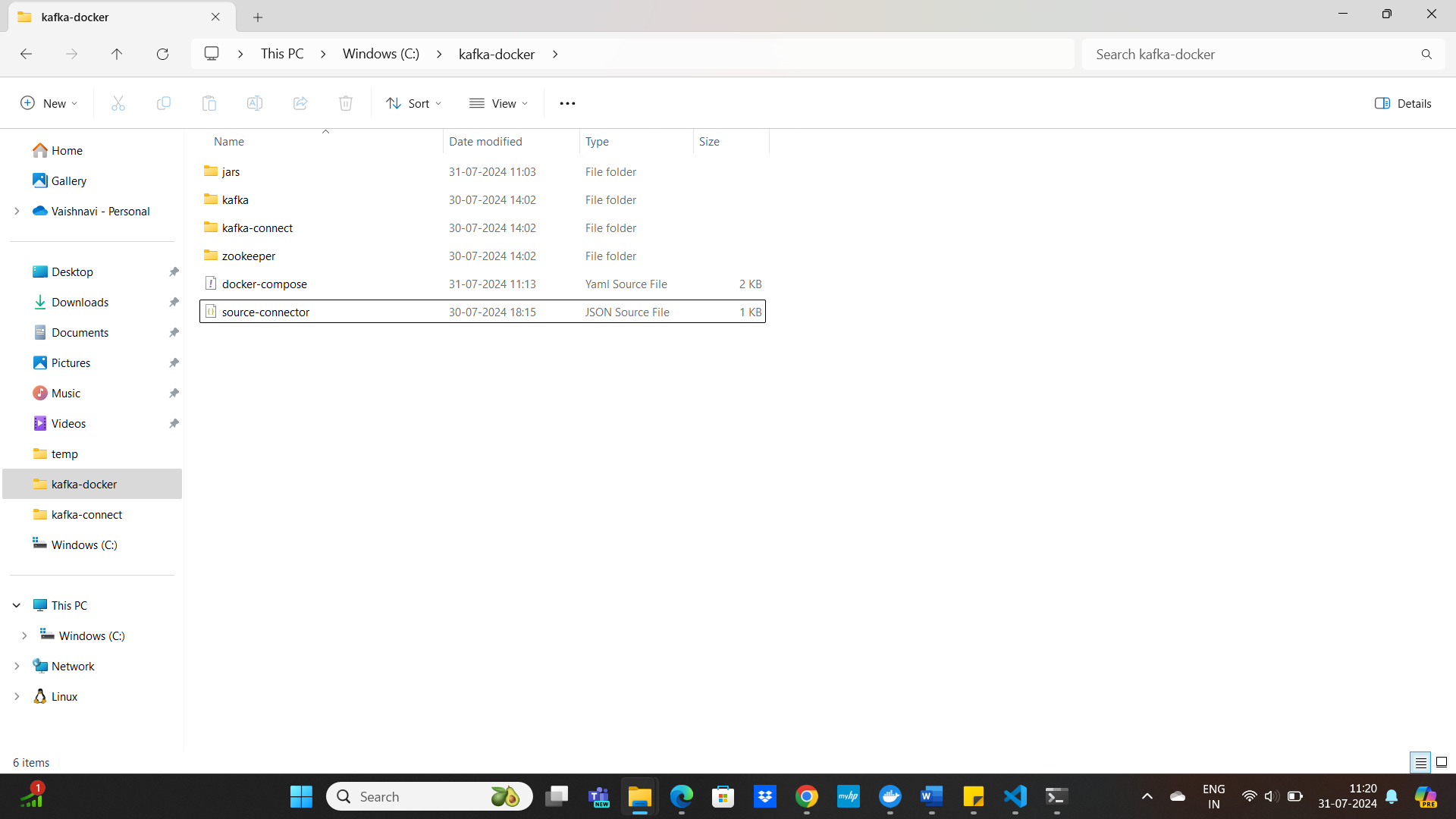
Streams APIs

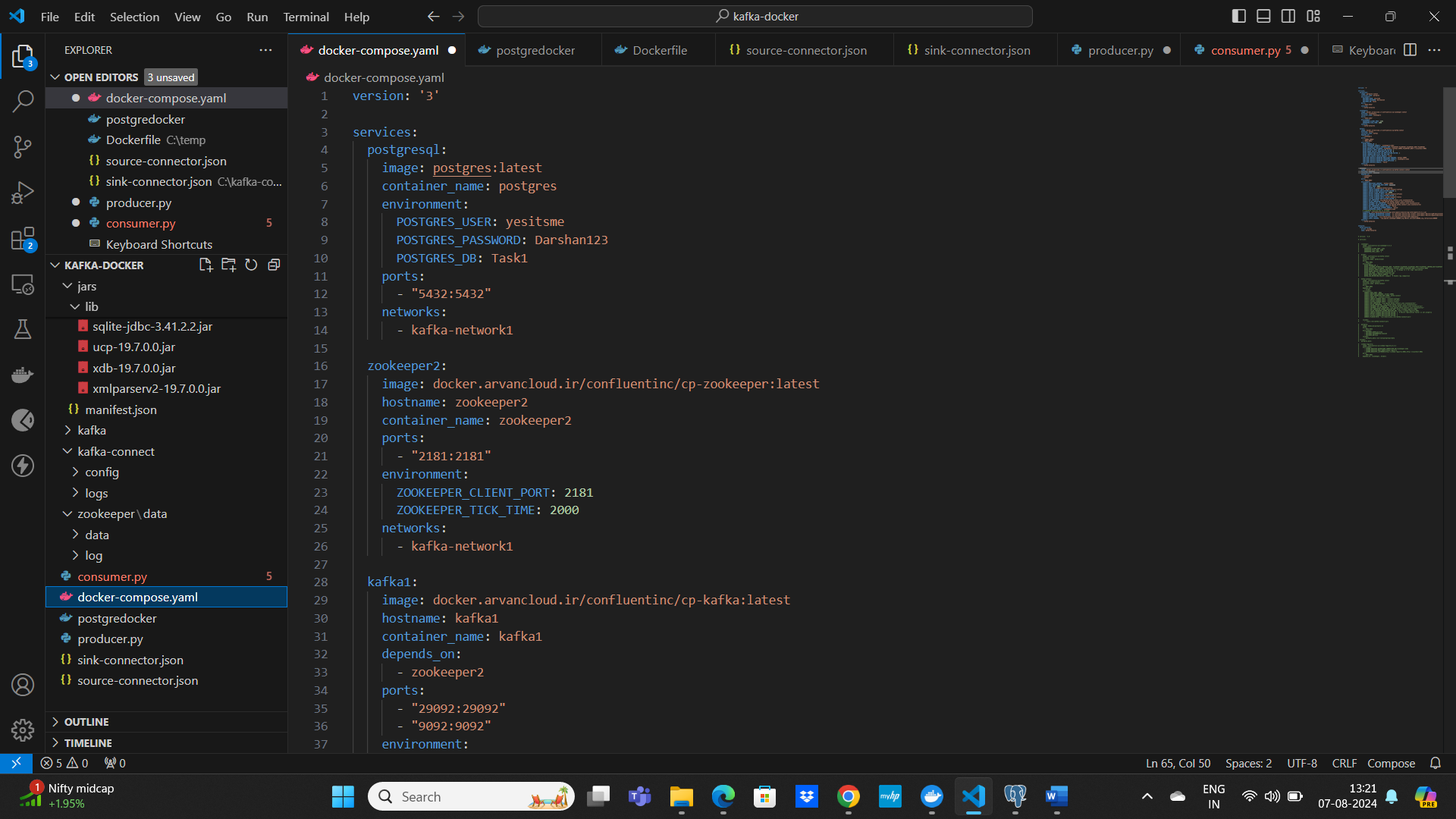
Connector APIs

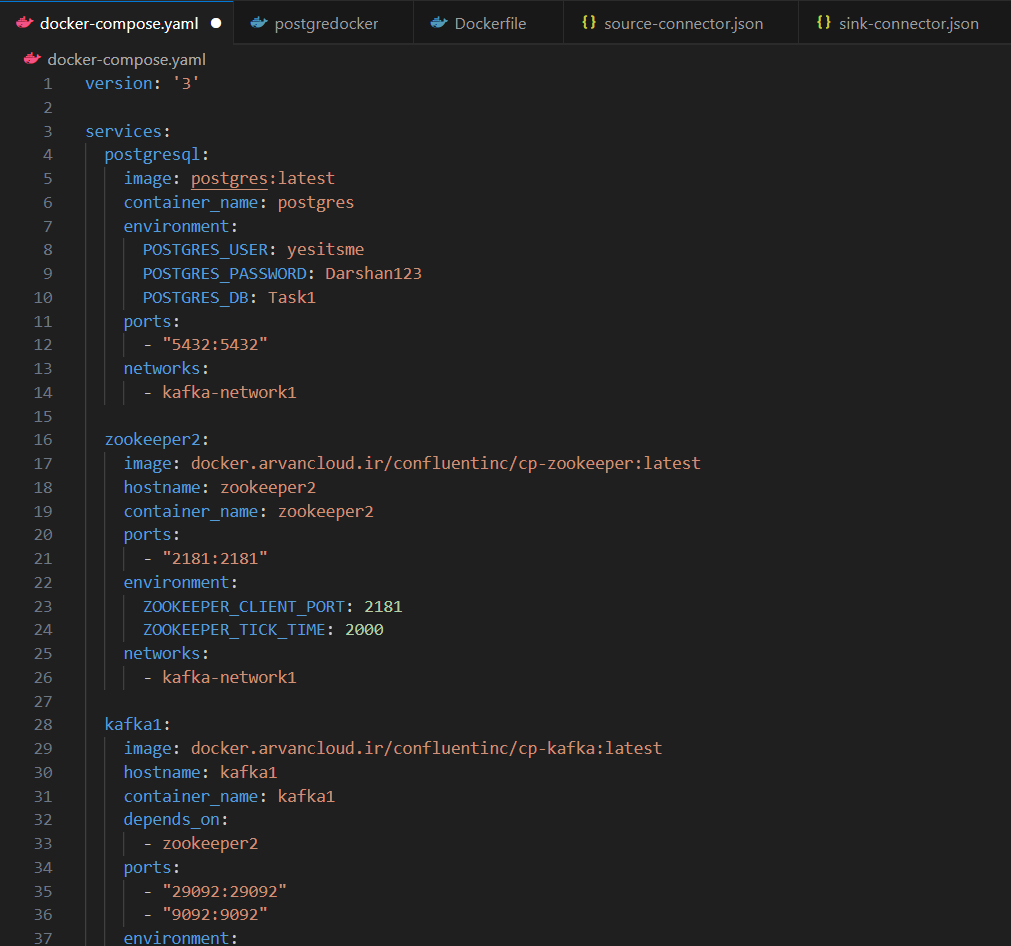
Admin APIs

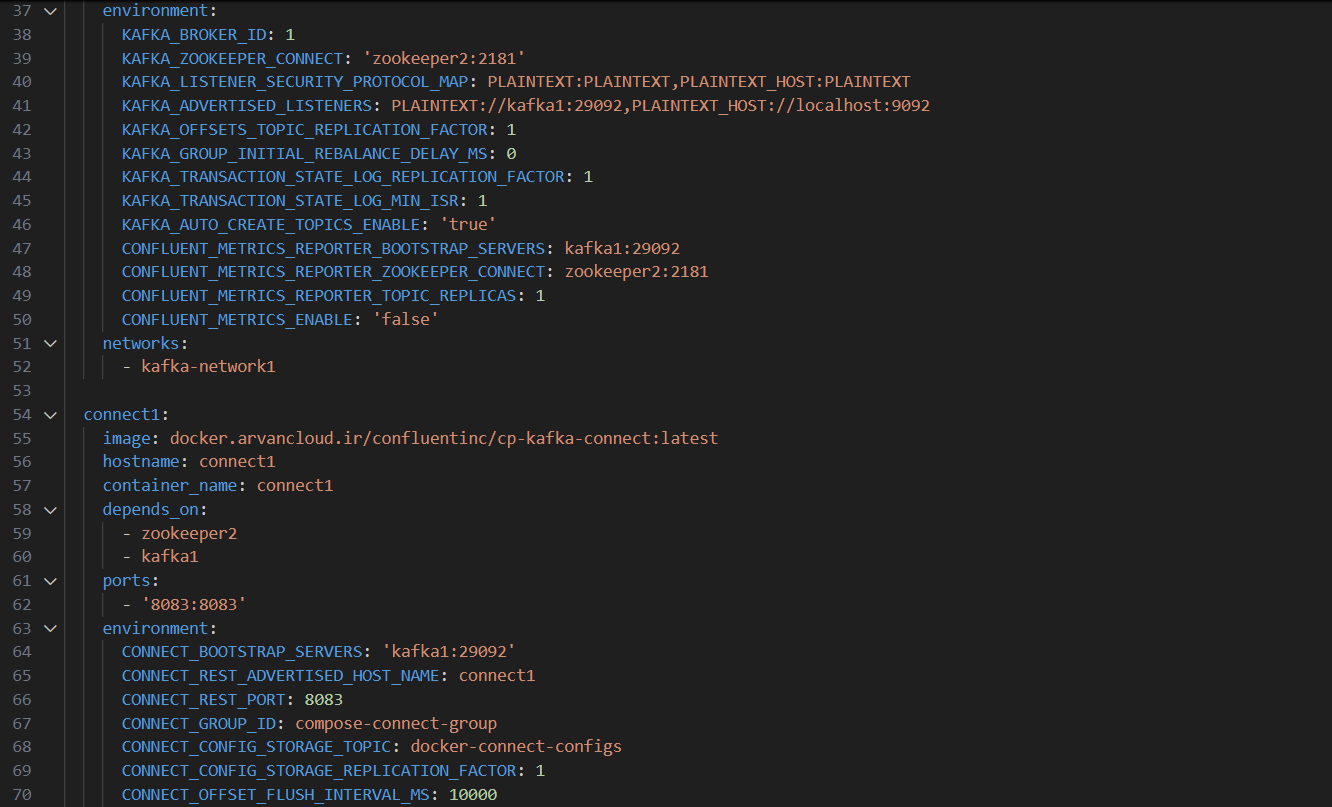
**Step 1:**

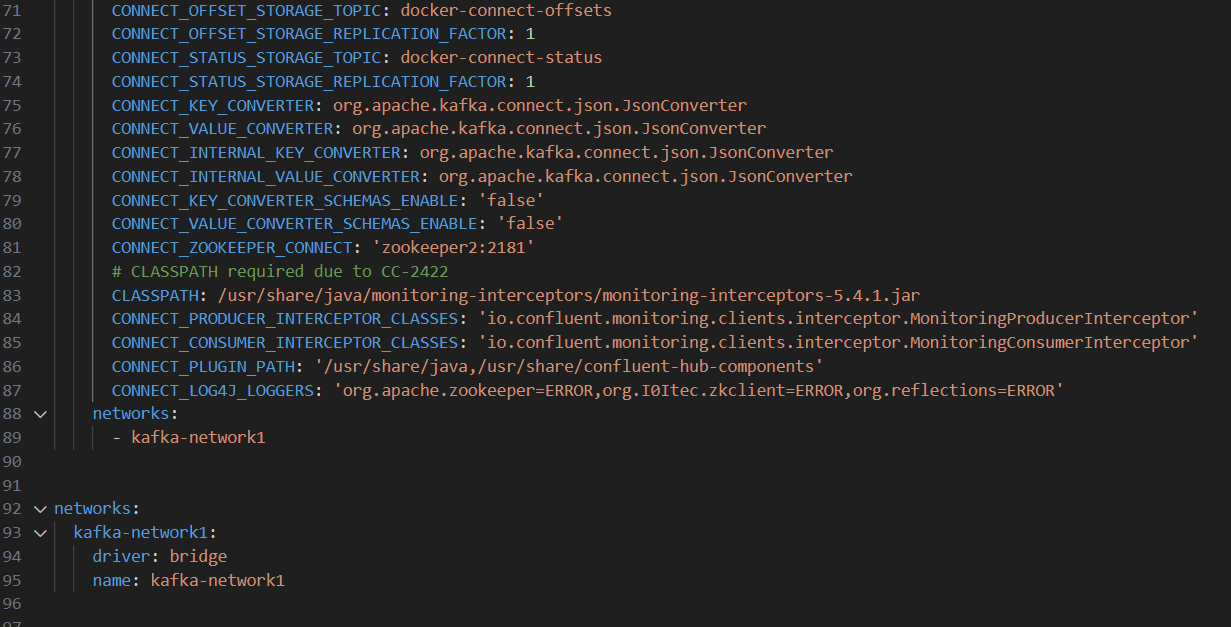
I have added all the essential files in one folder and then implemented in vs code.











Created a Directory for Docker Compose Project

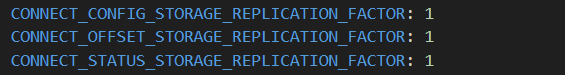
Used the following commands in terminal to create a new directory for your Kafka and Zookeeper setup:

Saved file in ‘yaml’ format

**cd C:\kafka-dockerr**: went into directory where my docker file yml is saved.

**Docker-compose up**: The docker compose up command aggregates the output of each container

* **Issues I got here in Step 1:**
  1. **At the time of Kafka connect I got replication factor error as, there was replication factor 3 and available broker count was 1.  
       
     so I debug that in docker-compose file**



* 1. **In Kafka:**

**org.apache.kafka.common.errors.InvalidReplicationFactorException: Replication factor: 3 larger than available brokers: 1.**

Reason:

**The error message you're encountering, InvalidReplicationFactorException, indicates that Kafka is trying to create a topic with a replication factor of 3, but there is only 1 available broker. In Kafka, the replication factor defines how many copies of each partition of a topic should be stored across different brokers for fault tolerance.**

* 1. **Connector was getting disconnected due to runtime error so I debug it with use of**



* 1. There was an issue in Kafka-connect as serialization and deserialization for that purpose

**I add CONNECT\_KEY\_CONVERTER: and CONNECT\_VALUE\_CONVERTER: - Specify how key/value data should be serialized/deserialized (using JSON format).**

**Docker-Compose File: Which combines the configuration of multiple containers.**

**services: Defines the different services (containers) that will be created and managed by Docker Compose.**

**###Postgres connection:**

* **Image: Uses the latest PostgreSQL Docker image.**
* **Container Name: Names the container "postgres".**
* **Environment Variables: Sets up the PostgreSQL database with a user (yesitsme), password (Darshan123), and database name (Task1).**
* **Ports: Maps port 5432 of the container to port 5432 of the host machine (accessible from the outside).**
* **Networks: Connects to a user-defined network, kafka-network1, for inter-service communication.**

**###Zookeeper Service**

* **zookeeper: -** Defines the service name as Zookeeper**.**
* **image: docker.arvancloud.ir/confluentinc/cp-zookeeper:latest**
* **-** Uses the latest version of the Zookeeper image from Confluent's Docker repository.
* **ports: -** Maps port 2181 of the container to port 2181 on the host, allowing external clients to connect to Zookeeper.
  + **"2181:2181" -** The default port Zookeeper uses; it must be accessible since Kafka depends on Zookeeper for managing its cluster state.

**environment:**- Environment variables used to configure Zookeeper.

* **ZOOKEEPER\_CLIENT\_PORT: 2181 -** Sets the port that clients connect to.
* **ZOOKEEPER\_TICK\_TIME: 2000** - Defines the frequency of heartbeats in milliseconds (2 seconds). This affects session timeouts and is crucial for maintaining the health of nodes.
* **ZOOKEEPER\_SYNC\_LIMIT: 2** - The maximum number of ticks that can pass between sending a request and getting an acknowledgment, impacting performance and latency.

**###Broker**

* **broker:** - Defines the service name as Kafka Broker.
* **image: docker.arvancloud.ir/confluentinc/cp-kafka:latest**

- Uses the latest version of the Kafka image from Confluent's Docker repository.

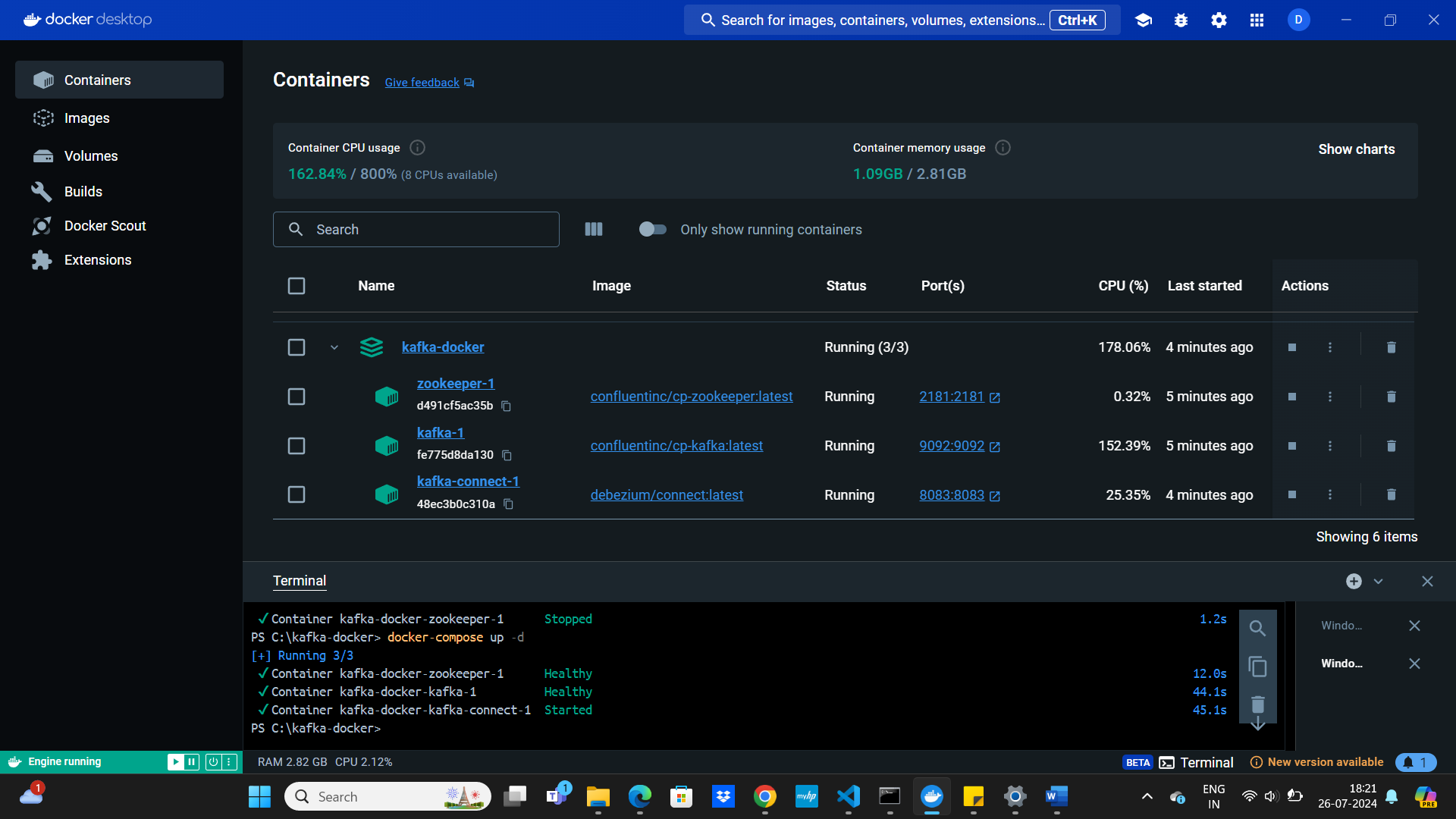
* **hostname: broker** - Sets the hostname for the Kafka broker within the Docker network.
* **container\_name: kafka-broker** - Assigns a specific name to the container for easy reference.
* **ports:** - Maps container port 9092 to host port 9092, enabling external access to Kafka.
* **environment:** - Contains environment variables for configuration:
  + **KAFKA\_BROKER\_ID: 1** - Unique identifier for the broker in the Kafka cluster.
  + **KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP:** - Maps listener names to their security protocols.
  + **KAFKA\_ADVERTISED\_LISTENERS:** - Adverts the addresses for clients to connect to this broker.
  + **KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1** - Sets the replication factor for the internal offsets topic; 1 means no redundancy.
  + **KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS: 0** - Specifies the delay before a rebalance of consumer groups occurs.
  + **KAFKA\_TRANSACTION\_STATE\_LOG\_MIN\_ISR: 1** - Minimum in-sync replicas for transaction logs, ensuring durability.
  + **KAFKA\_TRANSACTION\_STATE\_LOG\_REPLICATION\_FACTOR: 1** - Replication factor for transaction logs.
  + **KAFKA\_PROCESS\_ROLES: broker,controller** - The roles this broker will play in the cluster.
  + **KAFKA\_NODE\_ID: 1** - The ID of the node (set to the same as KAFKA\_BROKER\_ID).
  + **KAFKA\_CONTROLLER\_QUORUM\_VOTERS:** - Defines the list of nodes that can become controller (for high availability).
  + **KAFKA\_LISTENERS:** - Configures listeners for the broker, defining how it accepts connections (bind addresses).
  + **KAFKA\_INTER\_BROKER\_LISTENER\_NAME:** - Specifies the listener used for inter-broker communication.
  + **KAFKA\_CONTROLLER\_LISTENER\_NAMES:** - Defines which listener will be used for controller communications.
  + **KAFKA\_AUTO\_CREATE\_TOPICS\_ENABLE: 'true'** - Determines if Kafka should create topics automatically.
  + **KAFKA\_LOG\_DIRS: /tmp/kraft-combined-logs** - Directory where Kafka will store logs.
  + **CLUSTER\_ID: MkU3OEVBNTcwNTJENDM2Qk** - Identifier for the Kafka cluster (often set automatically in production).

**###Kafka-connect**

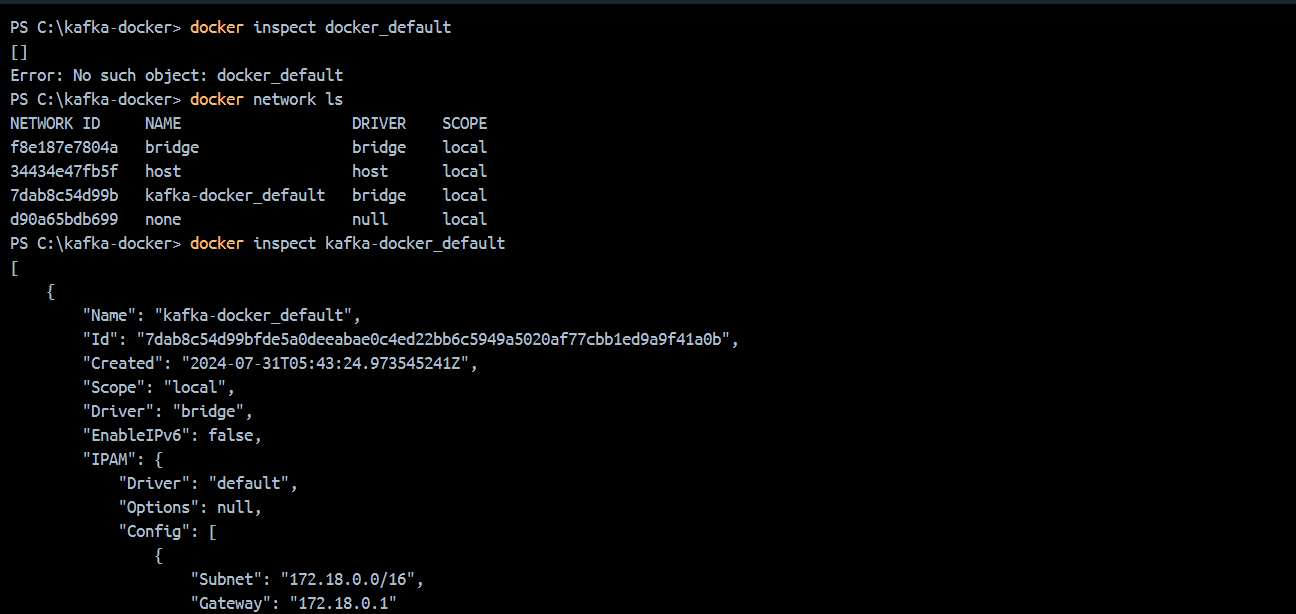
* **kafka-connect: - Defines the service name for Kafka Connect.**
* **image: confluentinc/cp-kafka-connect:latest - Uses the latest Kafka Connect image from Confluent's Docker repository.**
* **hostname: kafka-connect - Sets the hostname for the Kafka Connect container.**
* **container\_name: kafka-connect - Assigns a specific name to the container.**
* **ports: - Maps port 8083 of the container to port 8083 on the host, allowing access to the Kafka Connect REST API.**
* **depends\_on: - Ensures that Kafka Connect starts only after Kafka Broker is running.**
* **environment: - Consists of environment variables for configuring Kafka Connect:**
  + **CONNECT\_REST\_PORT: 8083 - Defines the port for the REST API where connectors will be managed.**
  + **CONNECT\_BOOTSTRAP\_SERVERS: broker:29092 - Points to the Kafka broker for establishing connections and sending/receiving messages.**
  + **CONNECT\_REST\_ADVERTISED\_HOST\_NAME: kafka-connect - Makes the Kafka Connect’s REST endpoint address resolvable to clients.**
  + **CONNECT\_GROUP\_ID: "connect-cluster" - Sets the identifier for the Connect worker group (used for coordinating distributed workers).**
  + **CONNECT\_CONFIG\_STORAGE\_TOPIC: "connect-configs" - Topic where connector configurations are stored.**
  + **CONNECT\_OFFSET\_STORAGE\_TOPIC: "connect-offsets" - Topic for storing offsets of messages processed by connectors.**
  + **CONNECT\_STATUS\_STORAGE\_TOPIC: "connect-status" - Topic for storing status updates of connectors.**
  + **CONNECT\_KEY\_CONVERTER: and CONNECT\_VALUE\_CONVERTER: - Specify how key/value data should be serialized/deserialized (using JSON format).**
  + **CONNECT\_INTERNAL\_KEY\_CONVERTER: and CONNECT\_INTERNAL\_VALUE\_CONVERTER: - Define the serialization format for internal messages.**
  + **CONNECT\_CONFIG\_STORAGE\_REPLICATION\_FACTOR: 1 - Sets the replication factor for the configuration topic.**
  + **CONNECT\_OFFSET\_STORAGE\_REPLICATION\_FACTOR: 1 - Sets the replication factor for the offset storage topic.**
  + **CONNECT\_STATUS\_STORAGE\_REPLICATION\_FACTOR: 1 - Sets the replication factor for the status topic.**
  + **CONNECT\_PLUGIN\_PATH: - Indicates where plugins (custom connectors) can be found.**
* networks:
* kafka-network1:
* driver: bridge
* name: kafka-network1
  + **It Connects to kafka.**
* **volumes: - Mounts a local directory (./jars) to a directory in the container (/etc/kafka-connect/jars) for storing custom connector JAR files.**
* **Conclusion**
* This configuration allows you to set up a basic Kafka ecosystem with Zookeeper, Kafka broker, and Kafka Connect, enabling you to handle real-time data streaming, connect various data sources, and manage data pipelines effectively. Each component plays a crucial role in the distributed messaging architecture, and the configurations ensure the services can communicate and work together seamlessly.

**Step 2:**

**Configured the successful connection of whole cluster:**



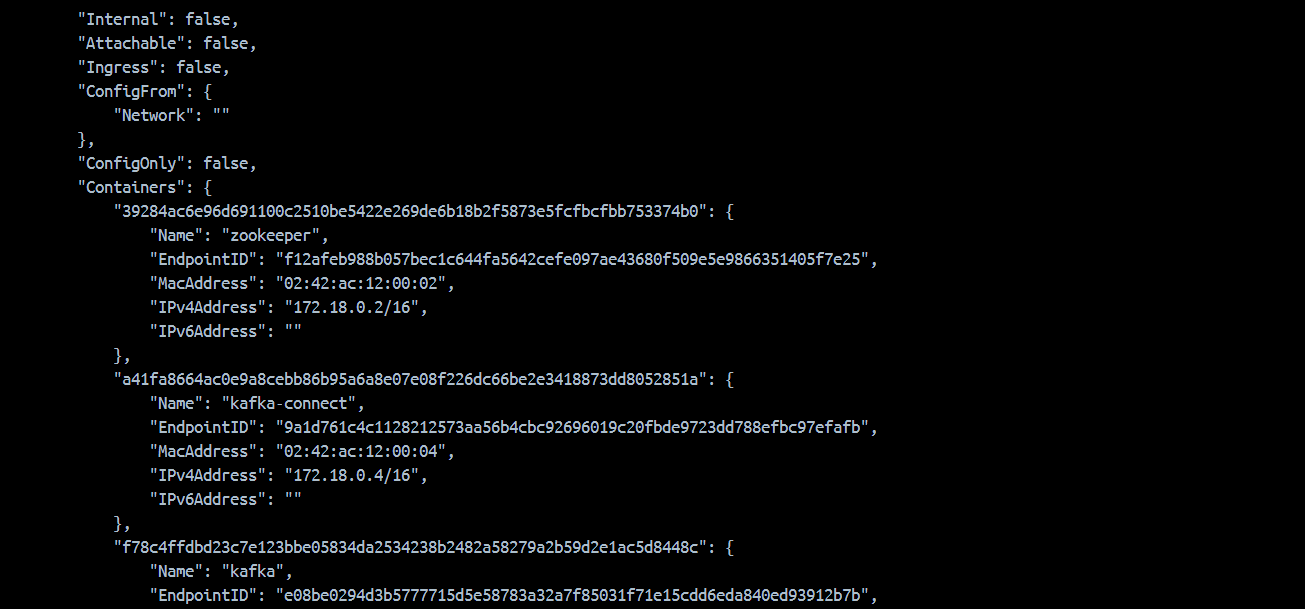
**Then I inspect the network connection as I have to add my-mssql-container which is database connection I made earlier in task 1**

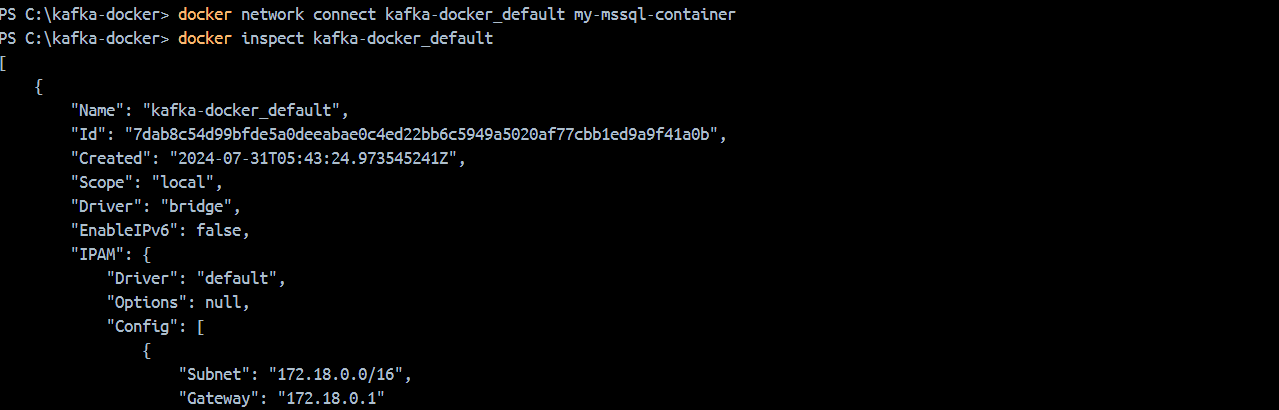
****

**Command:**

**docker inspect kafka-docker\_default**

**Result:**





**docker network connect kafka-docker\_default postgres-container :**

This command used to connect a running Docker container (postgres-) to an existing Docker network (kafka-docker\_default)

* **docker network connect**: This is the Docker command used to connect a container to a network.
* **kafka-docker\_default**: This is the name of the network you want to connect the container to.
* **my-mssql-container**: This is the name of the container you want to connect to the network.

**docker inspect kafka-docker\_default:**

This command inspects the total running container connection.

**After all see, where your curl is located:**



**curl is a command-line tool and library used to transfer data to or from a server using various network protocols.**

**Issues I got in Step 2:**

* **As I haven’t made a network connection of my-mssql-container (already created database) so it wasn’t able to connector source and sink**
* **Then I inspect the network connection as I have to add my-mssql-container which is database connection I made earlier in task 1.**

**Step 3:**

**curl http://localhost:8083/connector-plugins**

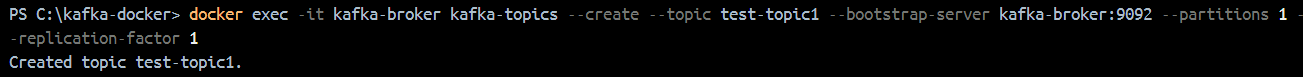
**When you run this command, you typically receive a JSON response listing the connector plugins.**

****

**Step 4: Creation of Kafka—Topic**

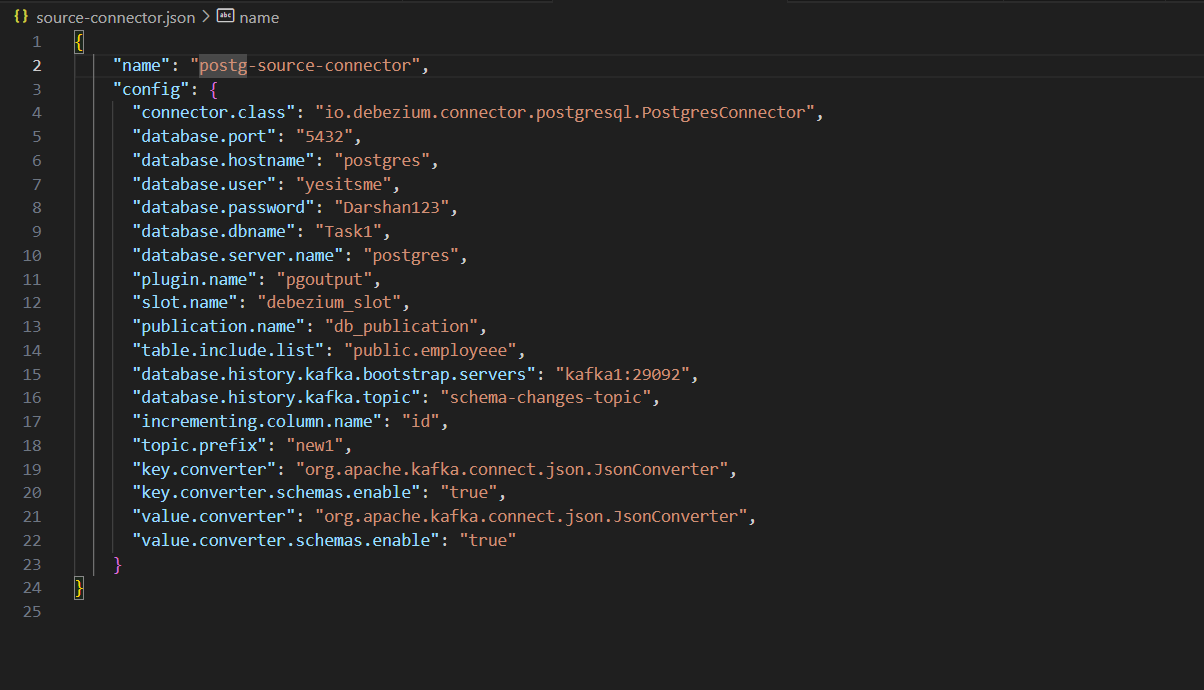
1. **The Kafka command-line tool will connect to the Kafka broker running in the specified container.**
2. **It will create a new topic named test-topic with:**
   * **1 partition**
   * **A replication factor of 1**

**docker exec -it kafka1 kafka-topics --create --topic new1 --bootstrap-server kafka1:9092 --partitions 1 --replication-factor 1**

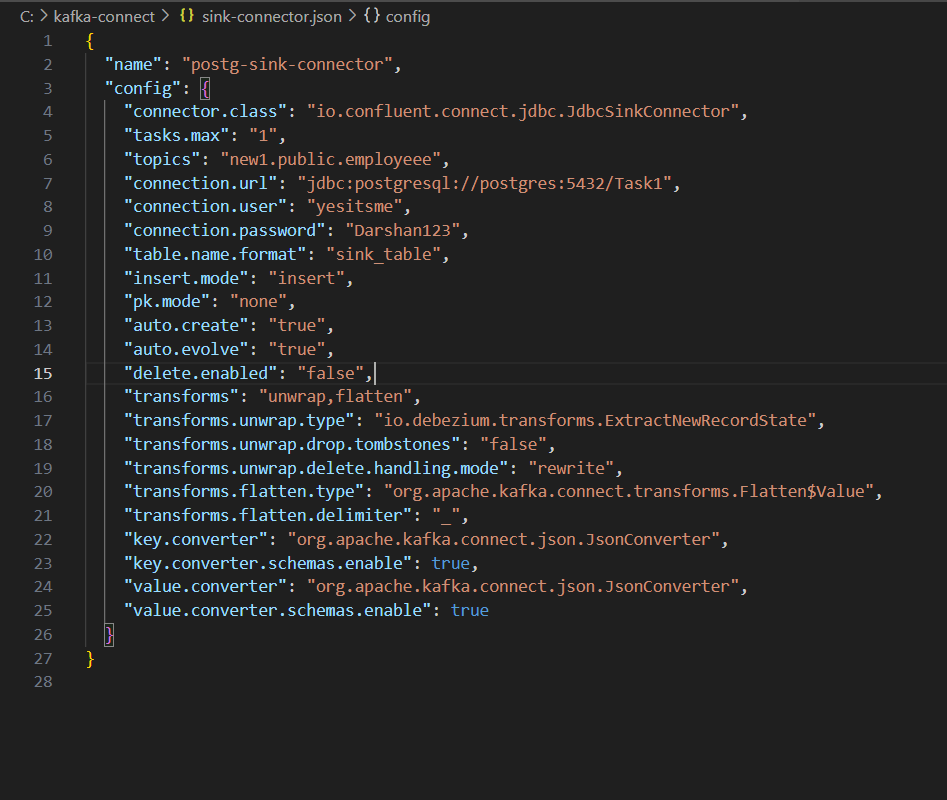
****

**Step 5: Made a execution of source and sink connector using curl as we have created a Kafka topic**

**Source-connector:**

****

**Sink-connector:**

****

**Then went to shell inside the directory for executing curl command with the json of source and sink which I created on my configurations:**

**docker exec -it <connector\_container> /bin/bash**

****

**Then run curl for source:**

curl -X POST \

http://localhost:8083/connectors \

-H "Content-Type: application/json" \

-d '{

"name": "postg-sink-connector",

"config": {

"connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",

"tasks.max": "1",

"topics": "new1.public.employeee",

"connection.url": "jdbc:postgresql://postgres:5432/Task1",

"connection.user": "yesitsme",

"connection.password": "Darshan123",

"table.name.format": "sink\_table",

"insert.mode": "insert",

"pk.mode": "none",

"auto.create": "true",

"auto.evolve": "true",

"delete.enabled": "false",

"transforms": "unwrap,flatten",

"transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",

"transforms.unwrap.drop.tombstones": "false",

"transforms.unwrap.delete.handling.mode": "rewrite",

"transforms.flatten.type": "org.apache.kafka.connect.transforms.Flatten$Value",

"transforms.flatten.delimiter": "\_",

"key.converter": "org.apache.kafka.connect.json.JsonConverter",

"key.converter.schemas.enable": true,

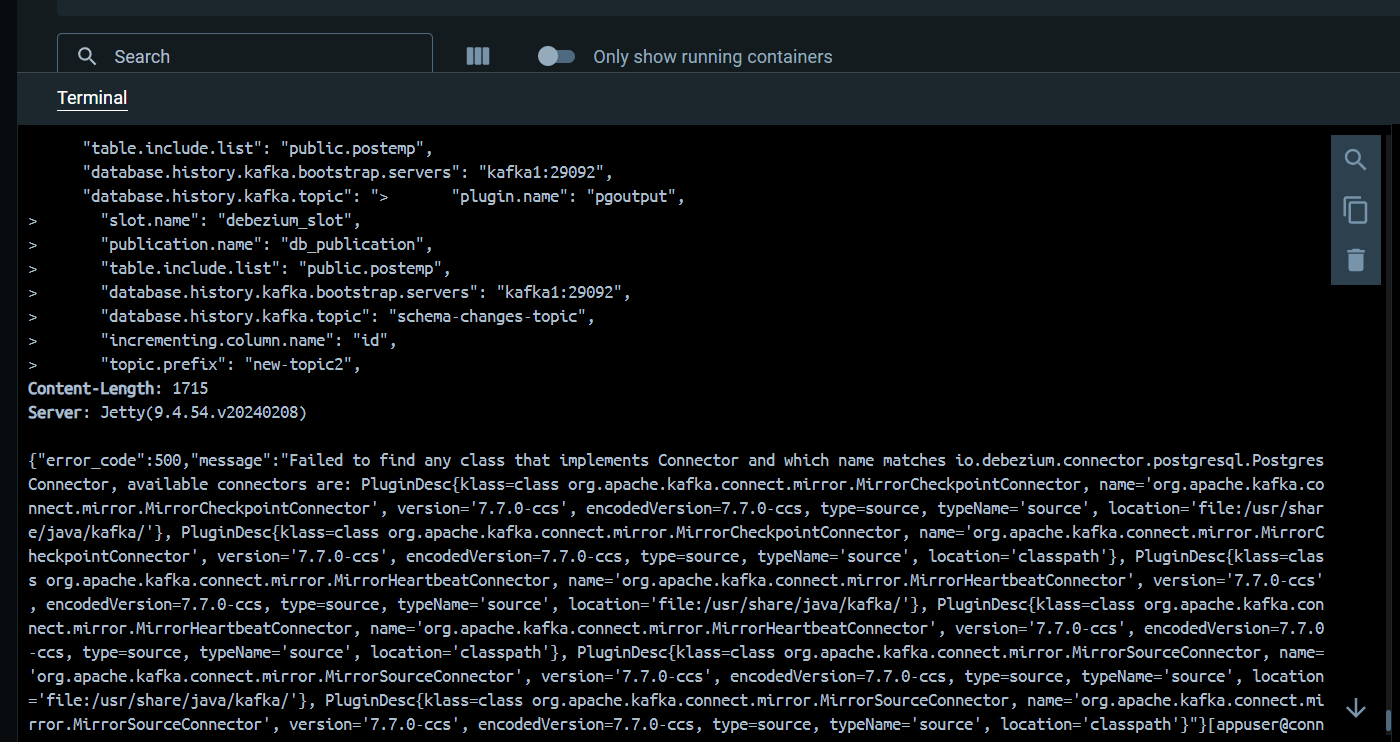
"value.converter": "org.apache.kafka.connect.json.JsonConverter",

"value.converter.schemas.enable": true

}

}'

**Issues Got here:**

****

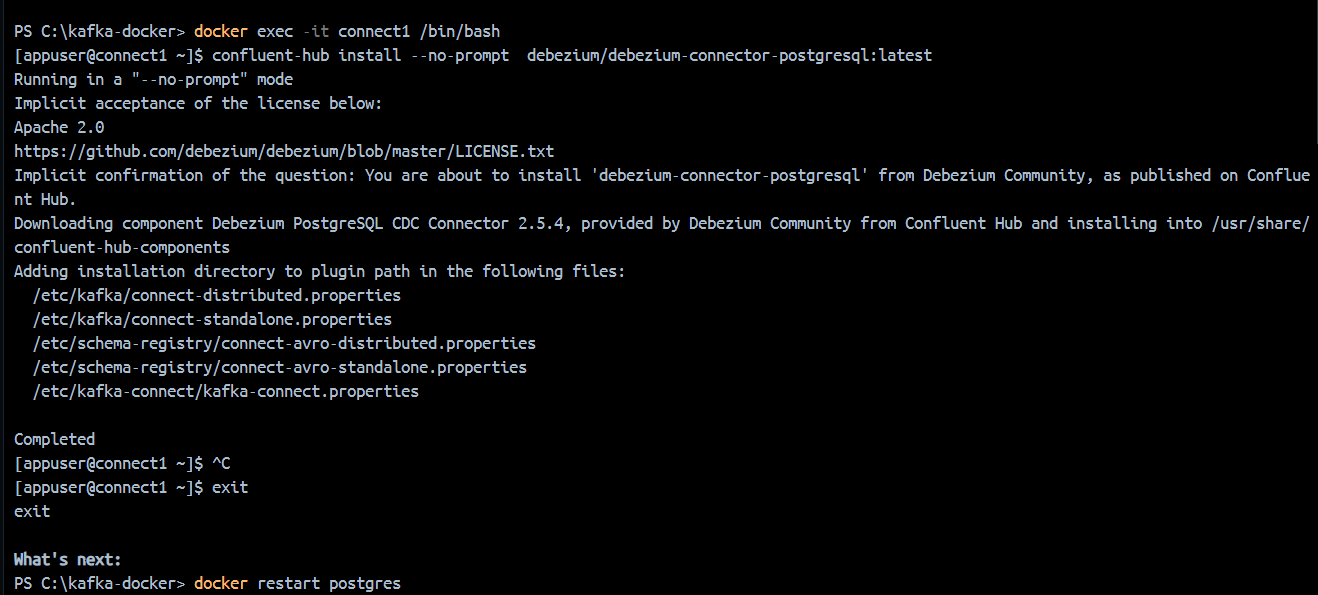
**Plugin issue of debezium connector not available in specified path so, need to install it.**

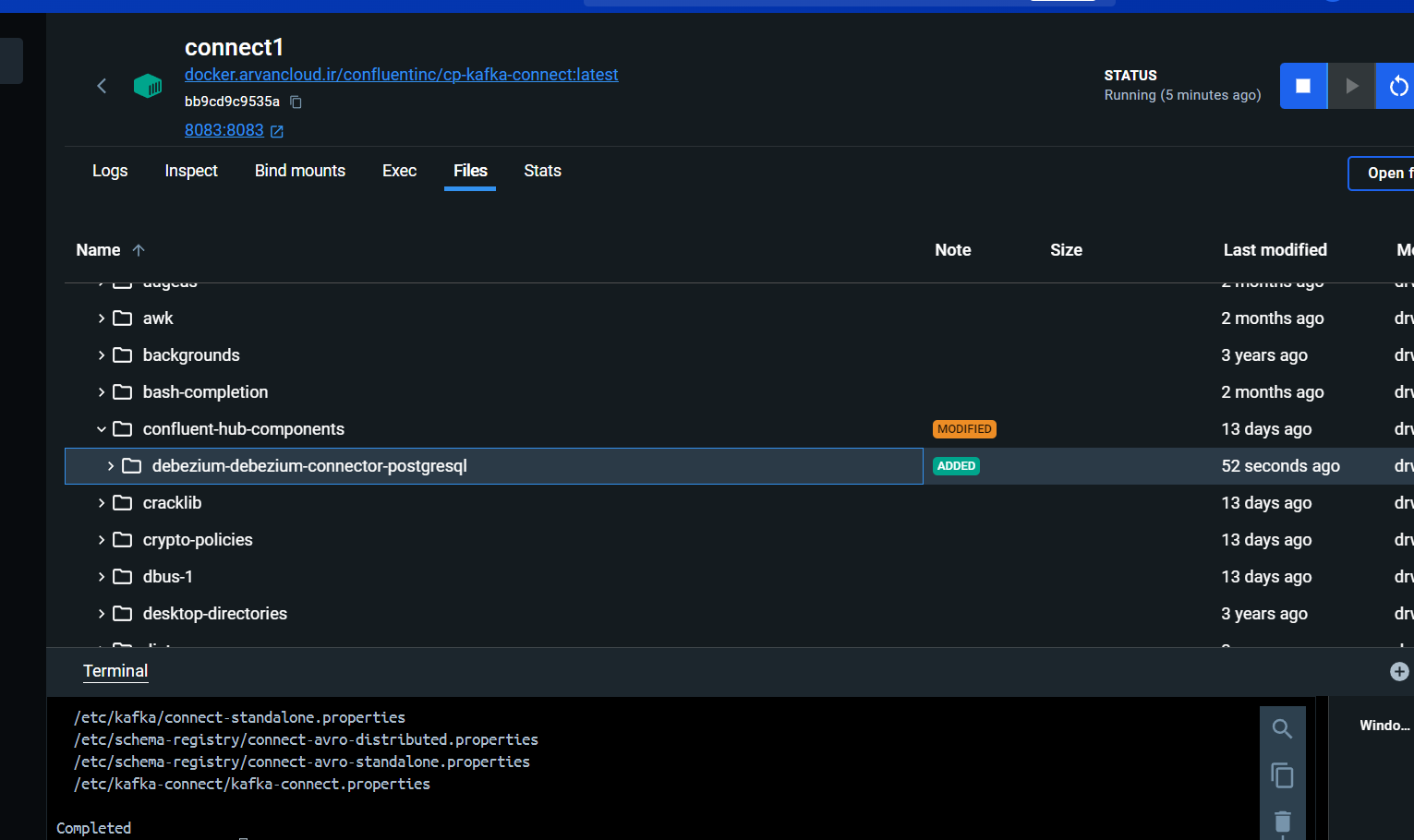
**Solution:**

****

**Then restart postgres container,**

**Docker restart postgres**

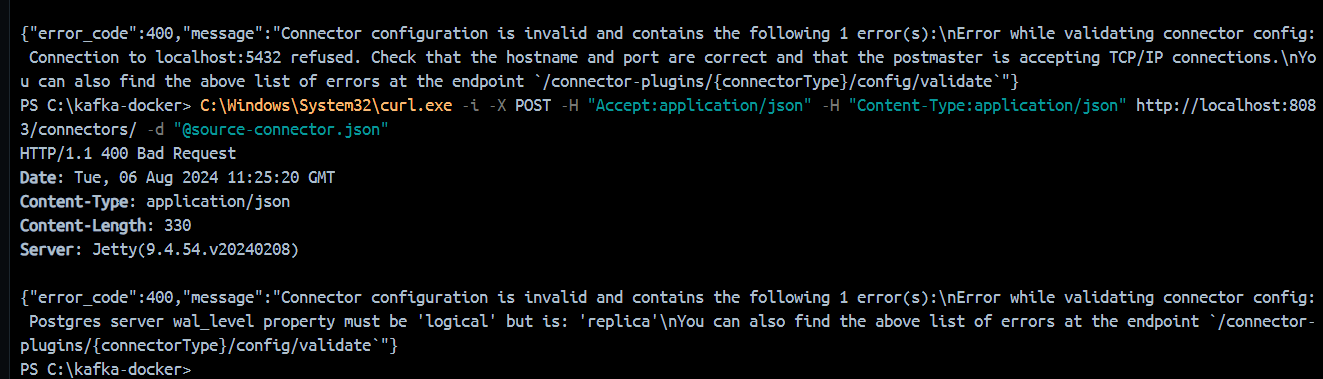
****

****

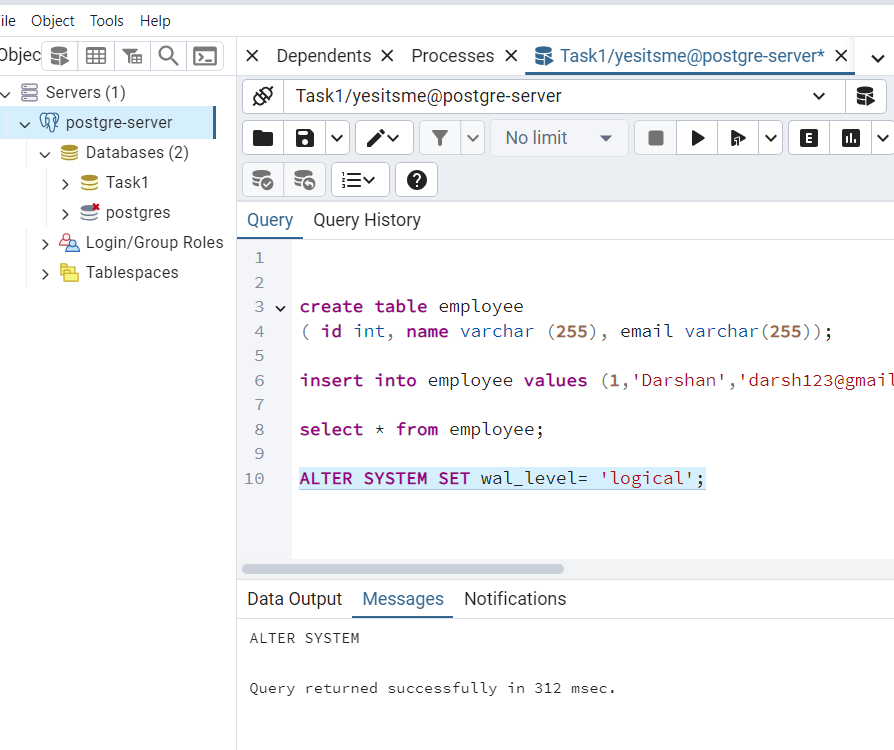
**This is the path where debezium-connector-postgresql is located.**

**Then run curl command:**

**Issues I Got:**

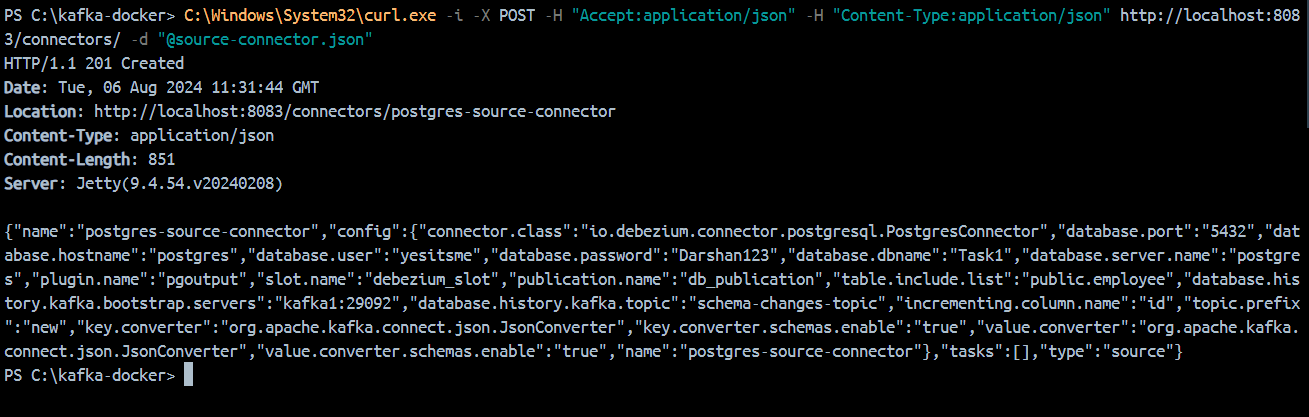
****

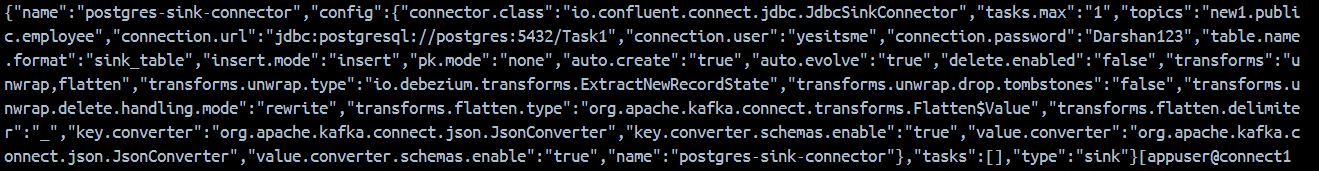
**As it is showing:**

****

**I have enabled by using alter and set that level to logical.**

****

****

**Now postg-source-connector is made succefully.** ****

**curl -X DELETE** [**http://localhost:8083/connectors/postg-source-connector**](http://localhost:8083/connectors/postg-source-connector) **:**

**to delete the existing already existing source-connector**

****

**Here, source is successfully created with using all configurations and network protocols.**

**Here data is reflecting after running following command**

**docker exec -it kafka1 kafka-console-consumer --bootstrap-server kafka1:9092 --topic new1--from-beginning:**

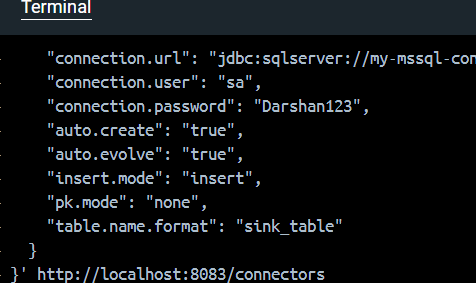
This command is use to fetch the real time data from source as whenever we will update it in postgre server it will reflect directly into the terminal

**Now, create sink-connector**

* + 1. **Go to shell first by using following command**

****

**docker exec -it kafka1 /bin/bash**

****

**Don’t forget to add http local host connection**

****

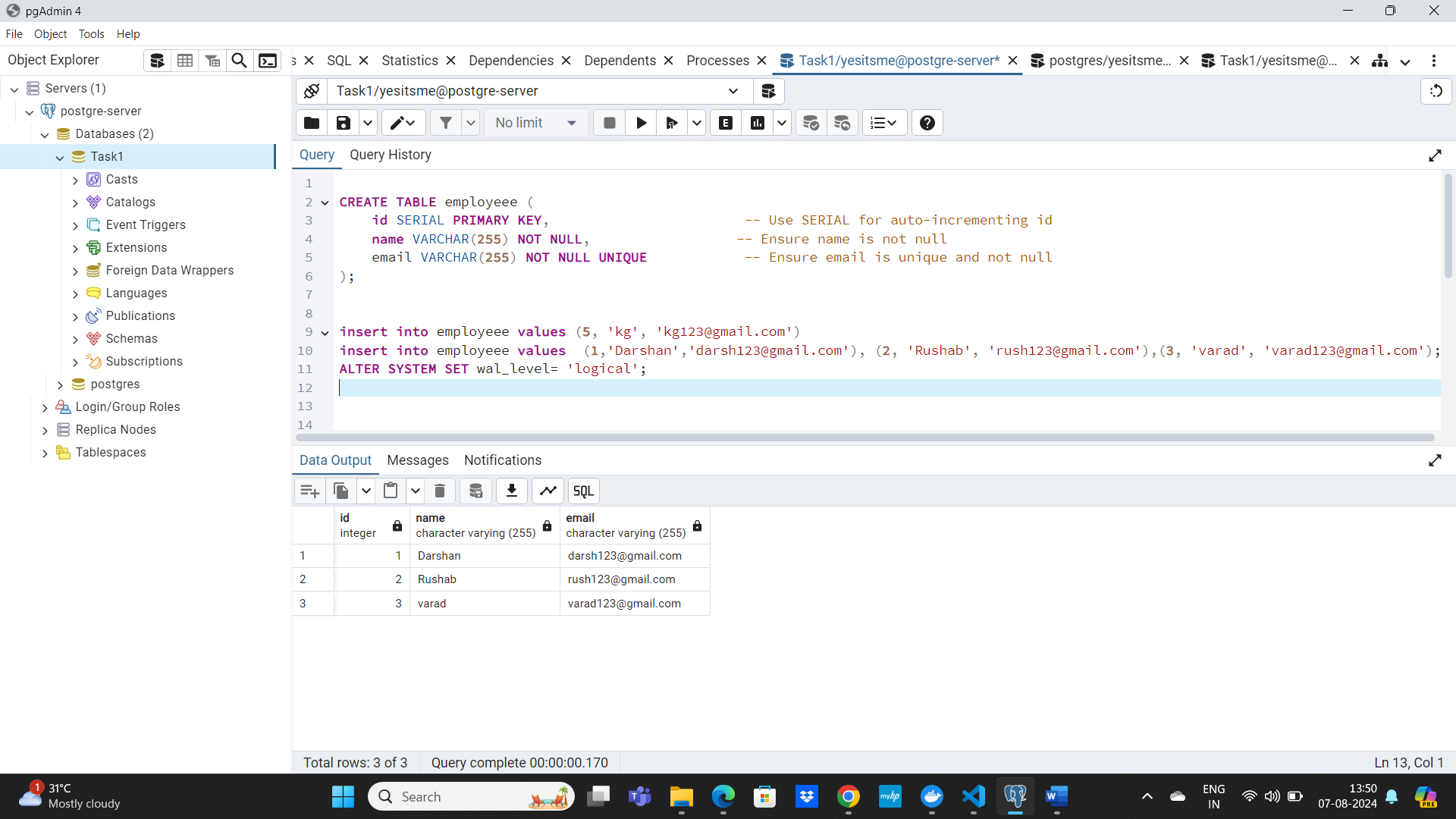
**Now, check the running status of both the connectors:**

**curl -X GET** [**http://localhost:8083/connectors/postg-source-connector/status**](http://localhost:8083/connectors/postg-source-connector/status)

**Issues I got in Step 5:**

* While doing connection using curl, actually I haven’t created topic initially.

**Step 6: Now do Crud operations into source data and see whether it is reflecting the same in terminal or not:**

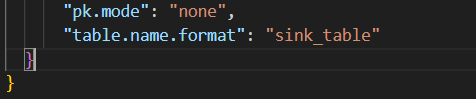
****

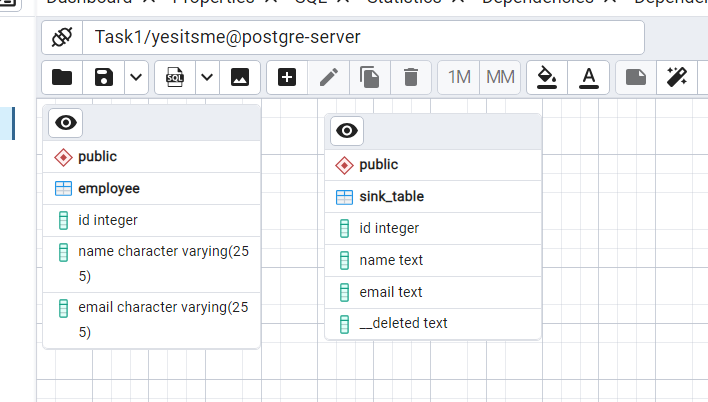
**Use following command on terminal after existing from shell ‘exit’:**

**docker exec -it kafka1 kafka-console-consumer --bootstrap-server kafka1:9092 --topic new1 --from-beginning**

**Step 7:**

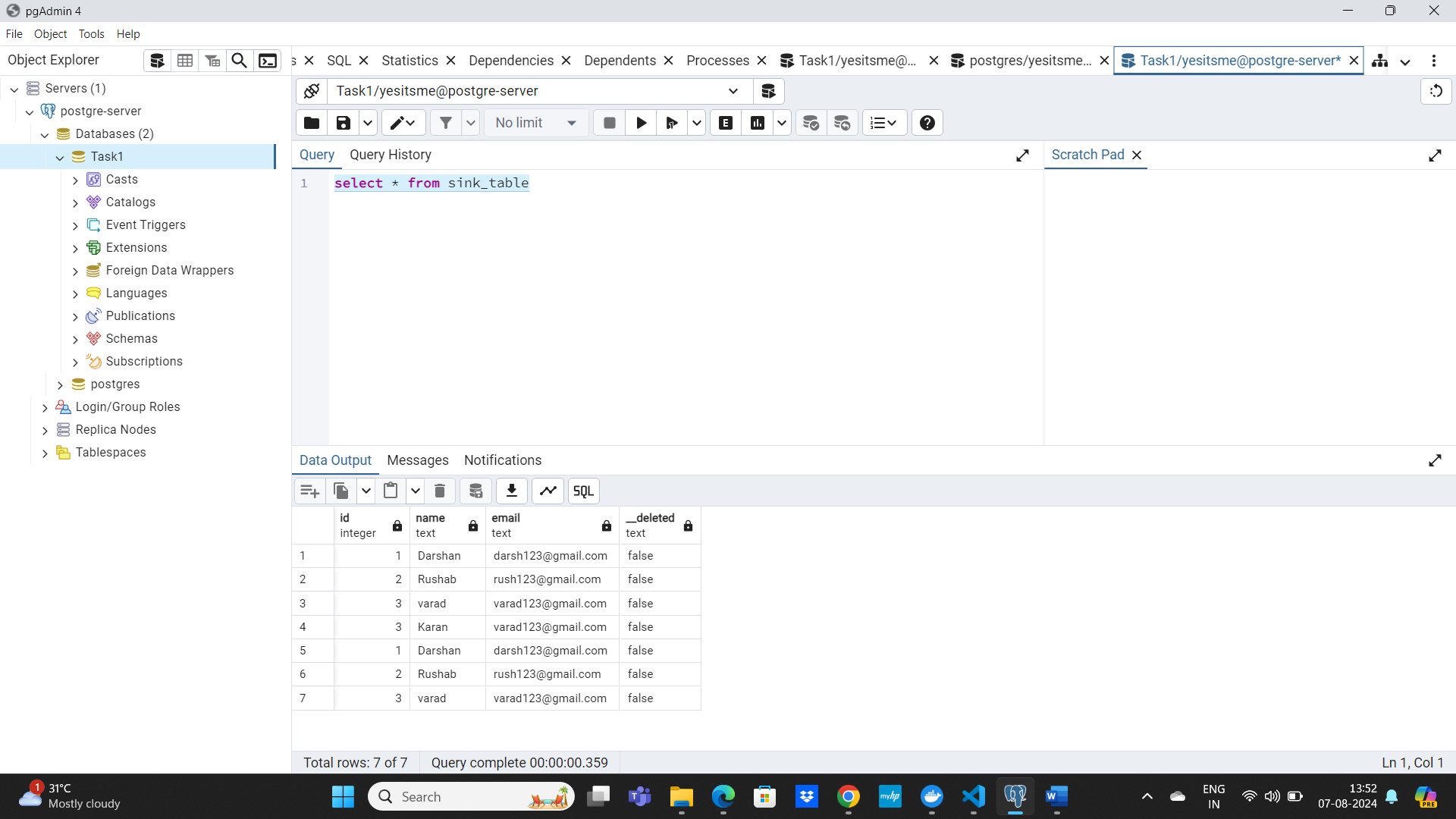
**Now, as in our sink connector we have manually created table name, so on that table the data of source has to be updated as we have made successful connection with database:**

****

****

**ERROR:  cannot update table "employee" because it does not have a replica identity and publishes updates  
HINT:  To enable updating the table, set REPLICA IDENTITY using ALTER TABLE.**

**The error message you're encountering indicates that your employee table is part of a logical replication setup, likely using PostgreSQL's replication features. In order to allow updates on a table that is being replicated, you need to set a replication identity for that table.**

****

**Issues I got in step 7:**

* **After updating data in source (In database) it was reflecting in source, but after in sink it wasn’t showing.**
* **Then I re-run all commands, I solved the error of Kafka-topic name,**
* **After that it was perfectly reflecting data in sink table name.**

**Final Outcome:**

**All tasks and subtasks concerning the setup of both source and sink connectors, as well as the operational status of Kafka, Broker and Zookeeper, have been successfully completed. The environment is now fully functional, with a reliable data flow established between Postgres-Sql and Kafka, enabling future data.**