The provided Docker Compose file defines a multi-container application setup using Docker. It specifies services for Zookeeper, three Kafka brokers, a PostgreSQL database, and a pgAdmin interface. The services are connected through a shared network called postgres-db-network.

**Services Overview:**

**Zookeeper:**

Zookeeper coordinates distributed applications by providing reliable data synchronization. It is crucial for managing the Kafka brokers.

• Ports: Exposes port 2181 to the host.

• Network: Connected to postgres-db-network for inter-container communication.

(Kafka Brokers:)

Three Kafka brokers are set up to ensure a robust and scalable message queue system. Each broker is configured to communicate with Zookeeper and other brokers.

• Ports: Each broker exposes unique internal and external ports to avoid conflicts.

• Environment Variables: Configured for advertised listeners, security protocol mappings, and broker settings.

• Dependencies: Each Kafka broker depends on Zookeeper, ensuring Zookeeper starts first.

(PostgreSQL:)

PostgreSQL is a powerful, open-source relational database management system.

• Ports: Exposes port 5432 to the host.

• Environment Variables: Includes default credentials for the database user.

• Volumes: Data persistence is ensured by mapping a local directory to the container's data directory.

• Network: Connected to postgres-db-network.

(pgAdmin:)

pgAdmin provides a web-based interface to manage and interact with the PostgreSQL database.

• Ports: Exposes port 16543 to the host.

• Environment Variables: Sets default login credentials for pgAdmin.

• Volumes: Configuration files are mapped to enable persistent settings.

• Network: Connected to postgres-db-network.

Access pgAdmin: Access the pgAdmin web interface via http://localhost:16543.

(Network Configuration:)

All services are connected through a custom Docker network named postgres-db-network, which uses the bridge driver.

---------------------------------------------------------------------------------------------------

Configuration Files:

(Database.ini:)

This is for a PostgreSQL that contains the necessary details to connect to a PostgreSQL database:

• host: The hostname of the PostgreSQL server (localhost).

• database: The name of the database (test).

• user: The username to connect to the database (postgres).

• password: The password for the user (password).

(Config Parser Script:)

This Python script reads the configuration from the above configuration file and returns a dictionary with the connection parameters:

• filename: The name of the configuration file (database.ini).

• section: The section of the configuration file to read (postgresql).

(pgAdmin Server Configuration:)

This JSON file configures the pgAdmin tool to connect to the PostgreSQL server:

• Name: The display name for the server in pgAdmin (test).

• Group: The group under which the server will be categorized (Servers).

• Host: The hostname of the PostgreSQL server (postgres).

• Port: The port number on which the PostgreSQL server is running (5432).

• MaintenanceDB: The maintenance database (postgres).

• Username: The username to connect to the database (postgres).

• PassFile: The path to the password file (/pgpass).

• SSLMode: The SSL mode to use for the connection (prefer).

(Kafka Producer:)

The producer file generates random user data comprising an address, age, and phone number, serialized into JSON format, and sends it to a Kafka topic named 'test'. It utilizes the KafkaProducer class from the Kafka library, configured to connect to the Kafka broker on 'localhost:9092', with a periodic message production interval of 20 seconds. Random data is generated using Python's 'random' and 'string' modules, ensuring a varied dataset for testing and development purposes.

(Kafka Consumer:)

The Kafka consumer reads messages from the test topic and inserts the received data into a PostgreSQL database. It ensures that the database table exists and then continuously processes incoming messages.

• Ensure that you created database test in PostgreSQL before running consumer.