Introduction

This report documents my project experience implementing a basic messaging application using Apache Pulsar deployed within Docker on macOS. The goal was to explore Pulsar's capabilities in a containerized environment, focusing on setting up message producers and consumers.

Architecture of the Solution:

Overview

The project utilized Docker to host Apache Pulsar in standalone mode. This setup allowed me to run Pulsar locally without needing a full cluster deployment. The architecture consisted of two main parts: a producer and a consumer, communicating through the Pulsar broker.

Components,

1. **Apache Pulsar Docker Container**
    - Deployed using the official Apache Pulsar Docker image (apachepulsar/pulsar:latest).
    - Configured to run in standalone mode (bin/pulsar standalone), with ports exposed (6650 for Pulsar service and 8080 for admin interface).
2. **Producer**
    - Connected to the Pulsar broker (pulsar://localhost:6650) within the Docker container.
    - Sent messages ("Hello World") to a predefined topic (my-topic).
3. **Consumer**
    - Subscribed to the same topic (my-topic) to receive and process messages asynchronously.

Interaction Flow

- **Message Flow:** The producer published messages to the Pulsar broker, which then routed them to any subscribed consumers.
- **Broker Communication:** Both producer and consumer established connections with the Pulsar broker running in Docker, ensuring seamless message exchange.

Implementation Details:

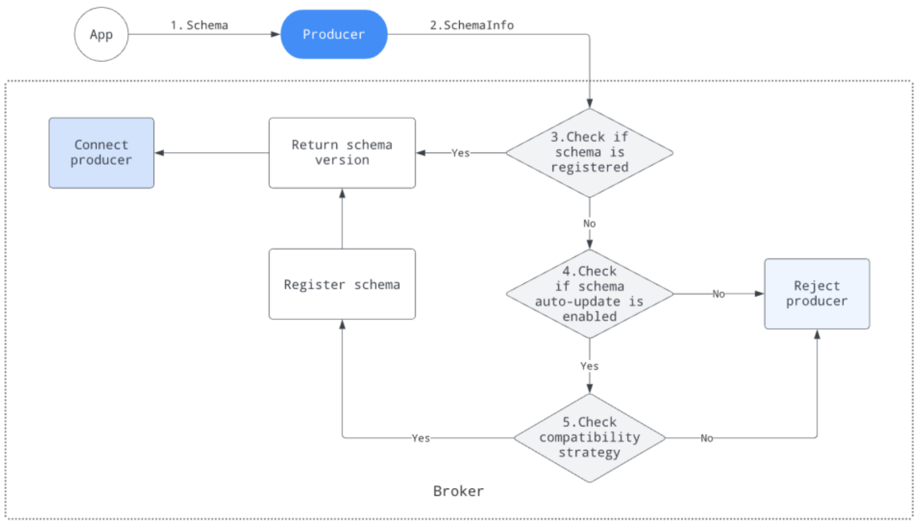
Tools and Technologies

- **Apache Pulsar:** Utilized via Docker for containerized deployment.

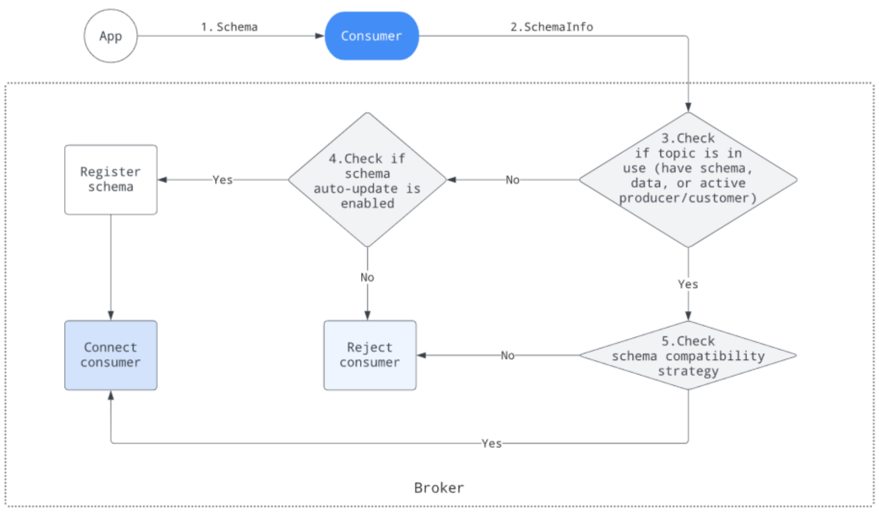
- **Python Scripts:** Developed using Python 3.11 (Producer.py and Consumer.py).
- **Pulsar Client:** Integrated pulsar-client Python library (pip install pulsar-client) for interacting with Pulsar.

Workflow

4. **Setting Up Docker Container:**
    - Pulled the Apache Pulsar Docker image and started a container (docker run ...).
    - Mapped necessary ports to allow communication (-p 6650:6650 -p 8080:8080).
5. **Producer Script (Producer.py):**
    - Established connection to the Pulsar broker (pulsar://localhost:6650).
    - Created a producer for the topic my-topic and sent messages.
    - This diagram illustrates how Pulsar schema works on the producer side.



6. **Consumer Script (Consumer.py):**
    - Connected to the same Pulsar broker instance.
    - Subscribed to my-topic to consume and process incoming messages.
    - This diagram illustrates how schema works on the consumer side.


App
1. Schema
Consumer
2.SchemaInfo
3.Check if topic is in use (have schema, data, or active producer/customer)
No
4.Check if schema auto-update is enabled
Yes
Register schema
No
Yes
5.Check schema compatibility strategy
No
Reject consumer
Yes
Connect consumer
Broker

Challenges Faced and Resolutions:

Challenges

7. **Dependency Management (Anaconda Issue):**
    - Initially faced issues installing pulsar-client using pip, as it installed into the Anaconda environment, causing compatibility problems with scripts.
8. **Local Environment Setup (VSCode Execution):**
    - Encountered difficulties running Python scripts (Producer.py and Consumer.py) within VSCode due to environment path and configuration mismatches.

Resolutions

9. **Isolating Dependencies:**
    - Addressed by creating a separate Python virtual environment (venv) outside Anaconda to manage dependencies cleanly and ensure proper installation of pulsar-client.
10. **Environment Configuration:**
    - Updated VSCode settings to use the correct Python interpreter path (/usr/bin/python3 or Anaconda path /opt/anaconda3/bin/python), aligning with the virtual environment setup for script execution.

Conclusion

Deploying Apache Pulsar within Docker on macOS provided valuable insights into scalable messaging solutions. By overcoming initial setup challenges and leveraging Docker's containerization benefits, the project successfully demonstrated core message queuing capabilities with Apache Pulsar.

Future Enhancements

Future improvements could focus on:

- **Automation:** Utilizing Docker Compose or Kubernetes for orchestrating Pulsar clusters.
- **Advanced Features:** Implementing message partitioning, schema management, and integration with cloud services for enhanced scalability and resilience.

References

- Apache Pulsar Documentation: https://pulsar.apache.org/docs/
- Docker Documentation: https://docs.docker.com/
- Python Pulsar Client Documentation: https://pulsar.apache.org/docs/en/client-libraries-python/

Video Link:Screen Recording 2024-06-13 at 15.07.56.mov