# Apache Pulsar "Hello World" Example with Python

-Spoorti Basarkod Math

## Introduction

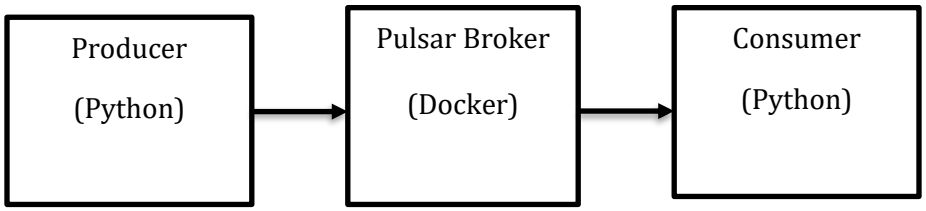
Apache Pulsar is a distributed pub-sub messaging system that offers features like multi-tenancy, high throughput, and low latency. In this project, we set up Apache Pulsar using Docker and created a simple 'Hello World' example with Python to demonstrate the basic operations of sending and receiving messages.

## Architecture of the Solution

The architecture of our solution consists of the following components:

1. Apache Pulsar: Runs in standalone mode using Docker, providing the messaging infrastructure.
2. Producer: A Python script that sends messages to a specified topic on the Pulsar broker.
3. Consumer: A Python script that subscribes to the topic and receives messages.

### Architectural Diagram




### Explanation

1. The Producer connects to the Pulsar broker and sends a message to the topic 'my-topic'.
2. The Pulsar Broker handles message delivery and ensures messages are available for consumers.
3. The Consumer subscribes to 'my-topic', receives messages, and acknowledges them.

## Implementation Details

### Environment Setup

1. Docker Installation: Ensured Docker is installed and running.
2. Pulsar Client Library: Installed the Pulsar client library for Python using pip.

### Starting Apache Pulsar

1. Pulled the Pulsar Docker Image.
2. Ran Pulsar in Standalone Mode.
3. Verified the Pulsar Container is Running.

### Python Scripts

#### *Producer Script (producer.py)*

The Producer script connects to the Pulsar broker, creates a producer for the topic 'my-topic', sends a 'Hello Pulsar!' message, and prints 'Message sent successfully' upon completion.

#### *Consumer Script (consumer.py)*

The Consumer script connects to the Pulsar broker, subscribes to the topic 'my-topic', receives a message, prints the received message, acknowledges the message, and then closes the client connection.

### Running the Scripts

1. Ran the Producer script which sent a 'Hello Pulsar!' message and printed 'Message sent successfully'.
2. Ran the Consumer script which received the 'Hello Pulsar!' message and printed 'Received message: 'Hello Pulsar!'.

## Challenges and Resolutions

### Docker Setup

Challenge: Ensuring Docker was installed and properly configured to run the Pulsar container.
Resolution: Followed Docker's official installation guides and verified the setup using `docker ps` to ensure the Pulsar container was running.

### Pulsar Client Library

Challenge: Installing the Pulsar client library for Python.
Resolution: Used pip to install the library, which provided a straightforward installation process.

**Message Acknowledgement**

Challenge: Ensuring messages are acknowledged by the consumer to avoid redelivery.

Resolution: Implemented message acknowledgment in the consumer script to confirm message receipt.

## Conclusion

In this project, we successfully set up Apache Pulsar using Docker and created simple producer and consumer scripts in Python to send and receive a 'Hello Pulsar!' message. This exercise demonstrated the fundamental operations of Apache Pulsar and how it can be utilized for messaging in distributed systems. The challenges encountered were primarily related to environment setup and were resolved through proper installation procedures and script implementation. By following this approach, we streamlined the setup process and effectively demonstrated a basic 'Hello World' example using Apache Pulsar and Python.