**KAFKA ASYNCHRONOUS PRODUCER**

* **Introduction to Kafka Producers**

Apache Kafka is a distributed streaming platform that allows developers to build real-time data pipelines and streaming applications. In Kafka, producers are responsible for writing records to Kafka topics. They can either operate in synchronous or asynchronous modes.

* **Synchronous vs. Asynchronous Production**

Synchronous: The producer sends data to Kafka and waits for an acknowledgment before proceeding.

Asynchronous: The producer sends data and immediately continues with the next task, without waiting for acknowledgment.

* **Understanding Asynchronous Producers in Kafka**

An asynchronous Kafka producer allows messages to be sent without waiting for confirmation, resulting in high throughput and reduced latency. In Python, the Kafka Producer API from the kafka-python library also provides an asynchronous, non-blocking way to send messages using the send() method.

* **Use Cases for Asynchronous Producers**

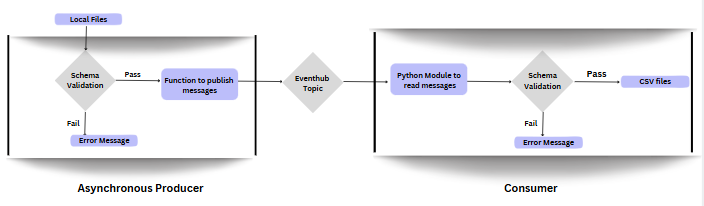
Asynchronous producers are ideal for applications requiring high throughput, such as:

. Real-Time Logging

. Telemetry Data Collection

. Event Stream Processing

**Architecture Diagram**



**Flow of Operation**

* **Configuration Loading**:
  + The system loads configurations from config.json, including credentials, Kafka topics, EventHub details, and connection parameters.
* **OAuth Authentication**:
  + Using eh\_cert\_oauth.py, the producer script authenticates with EventHub. This module leverages certificates for OAuth-based security, ensuring secure access to EventHub.
* **Data Production**:
  + The producer.py script generates or receives data to send to Kafka. The data is first validated via validate\_schema.py, ensuring adherence to predefined schemas.
  + Once validated, producer.py uses Kafka’s asynchronous API to send data. Messages are queued for transmission, allowing for high throughput and minimal delays in data publishing.
* **Data Consumption**:
  + Consumer.py listens to the designated Kafka topic(s) and processes incoming messages as they arrive. It can perform various actions based on the message content, such as data transformation or forwarding to other systems.
* **Data Flow Validation**:
  + Throughout the process, validate\_schema.py checks incoming and outgoing messages for consistency with predefined schemas. This validation step helps in detecting and handling malformed data early in the process.

**Usage Instructions**

**Running the Producer Module:**

1. Configure required parameters, including EventHub connection details and schema definitions.
2. Run the Producer script to initiate processing of files from local storage.

**Running the Consumer Module:**

1. Set up the necessary parameters, including EventHub connection details.
2. Execute the Consumer script to begin reading data from EventHub and writing it to CSV files.