BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Streaming Data Analytics for Web Application

Group 4

Team Members - Group 4

- Balaji O M - 2024mt03025
- Balasubramaniyan - 2024mt03053
- Deep Pokala - 2024mt03042
- Jagriti Sharma - 2024mt03116

ABSTRACT

- This project implements a comprehensive real-time data streaming pipeline using Apache Flume, Apache Kafka, and Python-based producer and consumer applications.

- The system demonstrates end-to-end event processing capabilities where a Python producer generates synthetic JSON events, writes them to a log file, which is then tailed by Apache Flume and forwarded to Apache Kafka.

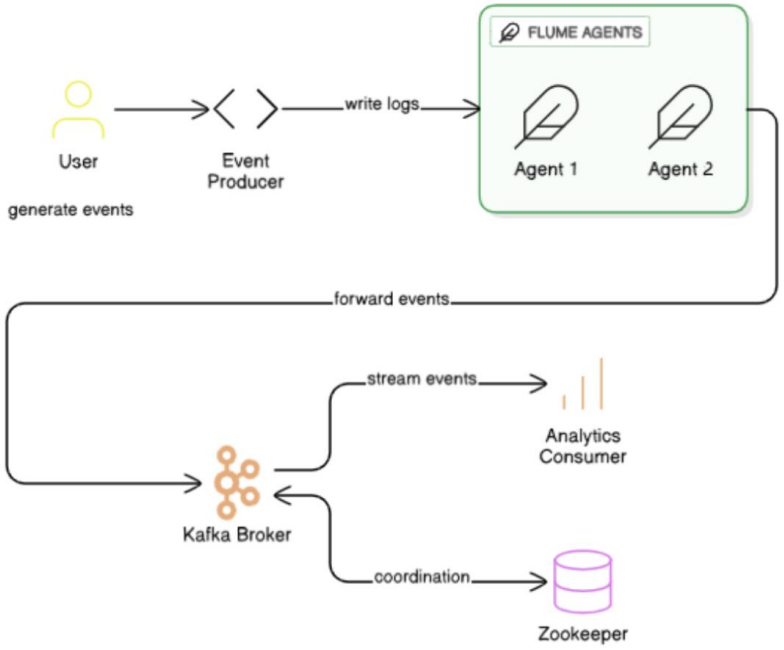
- A Python consumer subscribes to the Kafka topic and performs real-time analytics on the streaming data.

- The architecture showcases modern streaming data processing patterns, containerization with Docker, and resilient service integration with proper error handling and retry mechanisms.

Technology Stack

- Docker
- Python 3.11
- Apache Flume 1.9
- Apache Kafka 3.6 (with Zookeeper) and Docker Compose.

Streaming Architecture



eraser

Implementation Details - System Components

Python Producer Service:

- Generates synthetic e-commerce events with product information, prices, and timestamps.
- Code Snippet (**producer.py)**: The generate_event function creates a dictionary with product and user information.

```python
def generate_event(fake: Faker) -> dict:
    return {
        "event_id": fake.uuid4(),
        "ts": datetime.now(timezone.utc).isoformat(),
        "user": {
            "id": fake.random_int(min=1, max=1_000_000),
            "name": fake.name(),
            "email": fake.email(),
        },
        ...
        "product": {
            "id": fake.random_int(min=1, max=10_000),
            "name": fake.word(),
            "price": round(random.uniform(5.0, 500.0), 2),
            "currency": "USD",
        },
        ...
    }
```

Apache Flume Agent:

- Configured with an exec source and tail -F command to monitor the log file.
- Code Snippet (flume.conf): The configuration file defines the source, channel, and sink.

```
a1.sources = r1
a1.channels = c1
a1.sinks = k1

a1.sources.r1.type = exec
a1.sources.r1.command = tail -F /data/logs/input.log
a1.sources.r1.shell = /bin/bash -c
a1.sources.r1.batchSize = 100
```

Python Consumer Service:

- Subscribes to the Kafka events topic and performs real-time analytics.
- Code Snippet **(consumer.py)**: The processing logic categorizes prices into low/medium/high buckets.

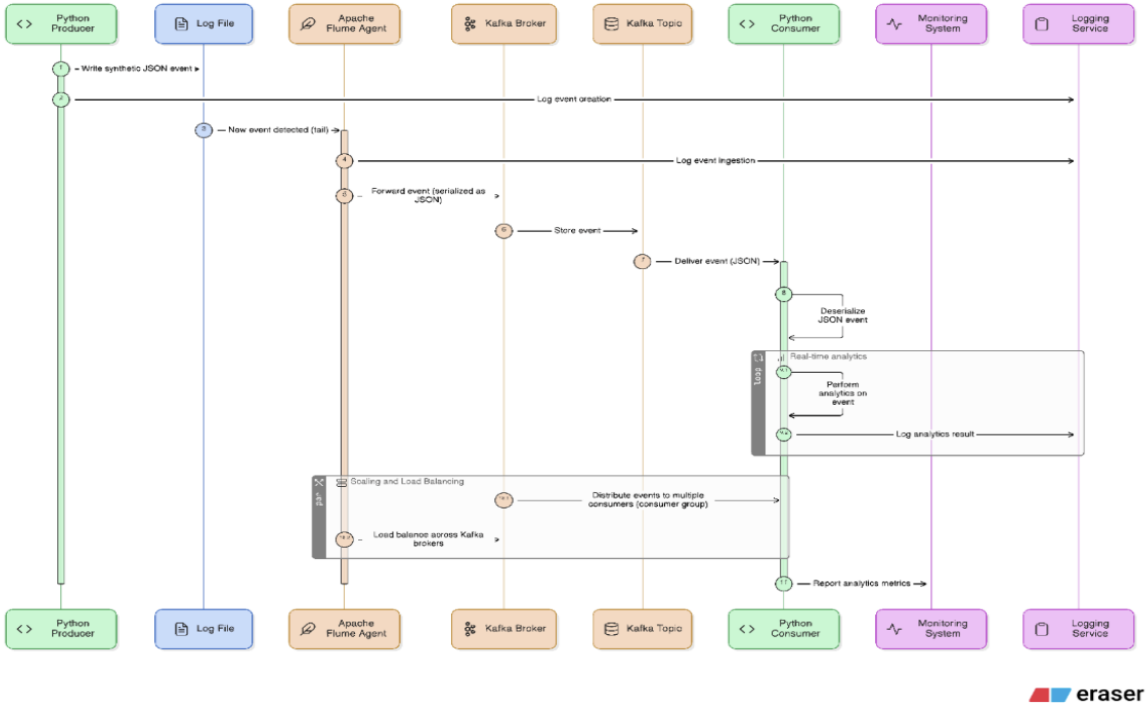
```
for message in consumer:
    event = message.value
    price = event.get("product", {}).get("price", 0)
    bucket = "high" if price >= 250 else ("medium" if price >= 50 else "low")
    print(f"offset={message.offset} key={message.key} bucket={bucket} event_id={event.get('event_id')} price={price}")
```

Data Flow and Containerization

Data Flow Architecture:

- **Event Generation:** The producer creates JSON events with product data.
- **File Persistence:** Events are written to a shared log file via a bind mount.
- **Event Ingestion:** Flume tails the file and buffers events in a memory channel.
- **Message Publishing:** Flume publishes events to the Kafka events topic.
- **Event Consumption:** The consumer subscribes to the topic and processes events in real time.

For diagram refer to next page



Containerization Strategy:

- Uses **Docker Compose** to orchestrate five containerized services (Zookeeper, Kafka, Flume, Producer, Consumer).
- A shared volume is used for log file access between the producer and Flume.

Use Case & Demo Illustration

Use Case: The consumer service performs real-time analytics by categorizing prices into "low", "medium", or "high" buckets.

Quickstart Commands: The demo can be run with a simple set of make commands.

- make up builds and starts all containers.
- make logs follows logs across services.

Expected Output: We would see the producer writing lines, Flume forwarding them to Kafka, and the consumer printing messages with a computed price bucket.

Visuals

We shall be seeing the producer writing lines, Flume forwarding them to Kafka, and the consumer printing messages with a computed price bucket.

- Start the Application

```
[+] Building 6.4s (19/27)                                                                                      docker:default
 => => naming to docker.io/library/stream-processing-assignment-1-flume                                                  0.0s
 => [producer internal] load .dockerignore                                                                               0.2s
 => => transferring context: 2B                                                                                          0.0s
 => [producer internal] load build definition from Dockerfile                                                            0.2s
 => => transferring dockerfile: 270B                                                                                     0.0s
 => [consumer 1/5] FROM docker.io/library/python:3.11-slim@sha256:30a49f1bf6cc823952693ac8cd6eb7f3224af9d930c3ff58f38a0e6e0dc999b4  0.3s
 => => resolve docker.io/library/python:3.11-slim@sha256:30a49f1bf6cc823952693ac8cd6eb7f3224af9d930c3ff58f38a0e6e0dc999b4  0.1s
 => => sha256:44350d10c02e7ab437e3fe5a05e3405115ece5972b2b9f7cd0d68d23c72d5833 0B / 1.29MB                               0.2s
 => => sha256:30a49f1bf6cc823952693ac8cd6eb7f3224af9d930c3ff58f38a0e6e0dc999b4 10.37kB / 10.37kB                         0.0s
 => => sha256:70f7abeaf1577b30229dd1d7784d6c053a29104a56bb353fe23217ad6f0fabc3 1.75kB / 1.75kB                           0.0s
 => => sha256:bf02a2b853727373d9065ccd2cc7d40df56d6f1b8256ae5f3612a34caae3c5be 5.38kB / 5.38kB                           0.0s
 => => sha256:8c7716127147648c1751940b9709b6325f2256290d3201662eca2701cadb2cdf 0B / 29.78MB                              0.2s
 => [consumer internal] load build context                                                                               0.2s
 => => transferring context: 68B                                                                                         0.0s
 => [producer internal] load build context                                                                               0.2s
 => => transferring context: 68B                                                                                         0.0s
```

- Check if the services are running using docker ps:

```
CONTAINER ID   IMAGE                                    COMMAND                  CREATED          STATUS          PORTS                                                NAMES
534a6dc07408   stream-processing-assignment-1-producer   "python -u producer.…"   42 seconds ago   Up 32 seconds                                                        producer
be20b7736e43   stream-processing-assignment-1-consumer   "python -u consumer.…"   42 seconds ago   Up 35 seconds                                                        consumer
d4631925ed99   stream-processing-assignment-1-flume      "tini -- /opt/flume/…"   4 days ago       Up 34 seconds                                                        flume
2d500ecaab89   bitnami/kafka:3.6                        "/opt/bitnami/script…"   4 days ago       Up 37 seconds   0.0.0.0:9092->9092/tcp                               kafka
f1c48fb5c3e0   bitnami/zookeeper:3.9                    "/opt/bitnami/script…"   4 days ago       Up 38 seconds   2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, 8080/tcp   zookeeper
```

- The consumer service output showing price categorization

```
offset=19148 key=None bucket=low event_id=a89e1e3d-4905-4fe9-aa26-053b505efd8d price=35.32
offset=19149 key=None bucket=high event_id=85debf28-99b2-459a-8578-1de485caede3 price=300.36
offset=19150 key=None bucket=medium event_id=925a5cf4-90c4-4d55-b7d3-03c8bdae8207 price=65.53
offset=19151 key=None bucket=medium event_id=896e6d1c-8453-4f8f-8922-ffd7cb7571bf price=50.67
offset=19152 key=None bucket=high event_id=c89bc4f2-9f3a-4bfe-8eda-f37b01304c14 price=424.98
offset=19153 key=None bucket=high event_id=55d1b3f7-5537-42fd-8008-6c24959916c7 price=436.24
offset=19154 key=None bucket=high event_id=2ee34fff-ef5f-49f2-9c6d-4af23045b2e5 price=418.89
offset=19155 key=None bucket=medium event_id=b739406b-98cf-44bd-9744-de494cb65632 price=122.35
offset=19156 key=None bucket=high event_id=7b7a05b6-f901-4242-ad86-270f807d3796 price=366.62
offset=19157 key=None bucket=medium event_id=c985f67e-b3be-4892-8a3e-85934ec7e581 price=80.05
offset=19158 key=None bucket=high event_id=8051eaaf-8224-42c8-8f80-45e0dfb88ee5 price=330.07
offset=19159 key=None bucket=medium event_id=0ed04575-a109-42ff-9960-6918cdc683cd price=80.27
offset=19160 key=None bucket=high event_id=cb8e80fb-5c33-4897-b122-a202dca1073b price=416.17
offset=19161 key=None bucket=medium event_id=14394919-48e9-4b06-89b6-58470f2a764c price=161.49
offset=19162 key=None bucket=high event_id=cf14ed43-b869-4fb4-b79b-29f6d9bcb998 price=454.59
offset=19163 key=None bucket=high event_id=3b953697-5c26-4d57-9b79-00e4d2fe825f price=264.6
offset=19164 key=None bucket=medium event_id=638e7a70-6729-44cb-a1c9-f722a2e050cc price=83.02
```

- The producer service logs showing events being written to the log file

```
(base) ➜ stream-processing-assignment-1 git:(main) X docker logs -f producer
Producer writing to /data/logs/input.log at ~5 events/sec
```

- Flume Service

```
(base) ➜ stream-processing-assignment-1 git:(main) X docker logs -f flume
Warning: No configuration directory set! Use --conf <dir> to override.
Info: Including Hive libraries found via () for Hive access
+ exec /usr/local/openjdk-8/bin/java -Xmx20m -Dflume.root.logger=INFO,console -cp '/opt/flume/lib/*:/lib/*' -Djava.library.path= org.apache.flume.node.Application -n a1 -f /opt/flume/con
f/flume.conf
log4j:WARN No appenders could be found for logger (org.apache.flume.util.SSLUtil).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Warning: No configuration directory set! Use --conf <dir> to override.
Info: Including Hive libraries found via () for Hive access
+ exec /usr/local/openjdk-8/bin/java -Xmx20m -Dflume.root.logger=INFO,console -cp '/opt/flume/lib/*:/lib/*' -Djava.library.path= org.apache.flume.node.Application -n a1 -f /opt/flume/con
f/flume.conf
log4j:WARN No appenders could be found for logger (org.apache.flume.util.SSLUtil).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

Scripts & Commands

Service Management:

```
# Start all services
make up
# or
docker compose up -d --build
# Stop all services
make down
# or
docker compose down -v
```

Data Management:

```
# Reset log file (clean slate)
make clean
# Create Kafka topic manually (if needed)
make topic
# or
docker exec -it kafka kafka-topics.sh --bootstrap-server localhost:9092
--create --if-not-exists --topic events --replication-factor 1 --partitions 1
```

Monitoring and Debugging:

```
# List Kafka topics
docker exec -it kafka kafka-topics.sh
--bootstrap-server localhost:9092 --list
# Consume messages directly from Kafka
docker exec -it kafka kafka-console-consumer.sh
--bootstrap-server localhost:9092 --topic events --from-beginning
# Inspect specific service logs
docker compose logs -f producer
```

Integration with External Platforms: The attached assignment report provides examples for integrating with Kafka Connect, Elasticsearch, and Apache Spark.

Conclusions & Future Work

Key Achievements:

- Successfully implemented an end-to-end data pipeline from event generation to real-time processing.
- Leveraged Docker and Docker Compose for consistent, reproducible deployments.
- Demonstrated proper service orchestration with dependency management.

Future Enhancements:

- Implement **Avro schemas** for data validation and evolution.
- Integrate a stream processing engine like **Apache Flink** or **Kafka Streams** for complex event processing.
- Add **Prometheus/Grafana** monitoring for operational visibility.

Learning Outcomes:

- Gained hands-on experience with distributed streaming architectures, container orchestration, and event-driven system design patterns.