2025

IoT Data Processing and Analytics System

Documentation

Author: Niroshima Pothupitiya

Date: 2025-07-18

Version: 1.0

Table of Contents

Overview

The **Internet of Things (IoT)** refers to a network of interconnected physical devices—such as sensors, appliances, vehicles, and machinery—that collect and exchange data over the internet. These devices are embedded with software, sensors, and communication hardware, enabling them to monitor, report, and sometimes act on data in real time without human intervention.

Purpose of This System

- This system serves as a foundational IoT data pipeline designed to:
- Simulate data from multiple virtual IoT devices.
- Ingest, validate, and process the data in real time.
- Store the structured data for visualization and further analysis.
- Provide a basis for building advanced features such as anomaly detection, alerting, and dashboarding.

It is particularly useful for:

- Prototyping IoT architectures.
- Testing message-driven microservices.
- Demonstrating real-time data flows in IoT.
- Learning and research in data engineering and observability.

System Architecture

Components

1. IoT Data Generator

The **iotdatagenerator** microservice emulates real-time data streams by generating telemetry from virtual IoT devices, closely replicating the behavior and output of actual physical sensors and hardware. Developed using Spring Boot, the service simulates real-world IoT devices by generating sensor data at regular intervals through Spring's scheduling capabilities.

This service produces sensor metrics such as temperature, humidity, pressure, battery levels, and GPS coordinates. Each data record is tagged with a unique device identifier and a timestamp, ensuring accurate tracking and sequencing.

Generated messages are asynchronously dispatched to the Kafka message broker, which acts as a resilient, scalable messaging backbone. This design enables downstream services to consume, process, and analyze high-throughput IoT data streams effectively.

Here's a sensor data message generated by simulated IoT Data Generator

```
{
  "deviceId":"device-1",
  "timestamp":"2025-07-19T05:17:16.933273287Z",
  "payload”: {
     "temperature": 18.606194183404114,
     "humidity": 41.99578471723534,
     "pressure": 963.1329612169322,
     "latitude": 39.32221356710727,
     "longitude": -107.95274793908153
     "battery": 97
   }
}
```

2. Kafka Broker

Kafka Broker acts as the central message queue in the system, enabling real-time communication between microservices.

Role in the System:

- **IoT Data Generator Service**: Publishes real-time sensor data (e.g., temperature, humidity) to Kafka topics every 30 seconds using Spring Scheduler.
- **Ingestion Service**: Consumes data from the IoT Data Generator Service, validates it, and then publishes it to Kafka for the Processing Service.
- **Data Processing Service**: Consumes messages from Kafka to perform filtering, transformation, and enrichment, and then publishes the processed data to the Persistence Service.
- **Persistence Service**: Subscribes to the processed data from the Processing Service and stores it for further analysis or querying.

3. Ingestion Service

The Ingestion Service consumes raw sensor data from the IoT Data Generator via Kafka. It validates each message to ensure the data is accurate and complete. Valid messages are then sent to another Kafka topic for further processing. This service uses Spring Kafka to consume and produce messages, and it handles errors to maintain smooth data flow. Its main role is to ensure only clean and correct data moves forward in the system.

4. Processing Service

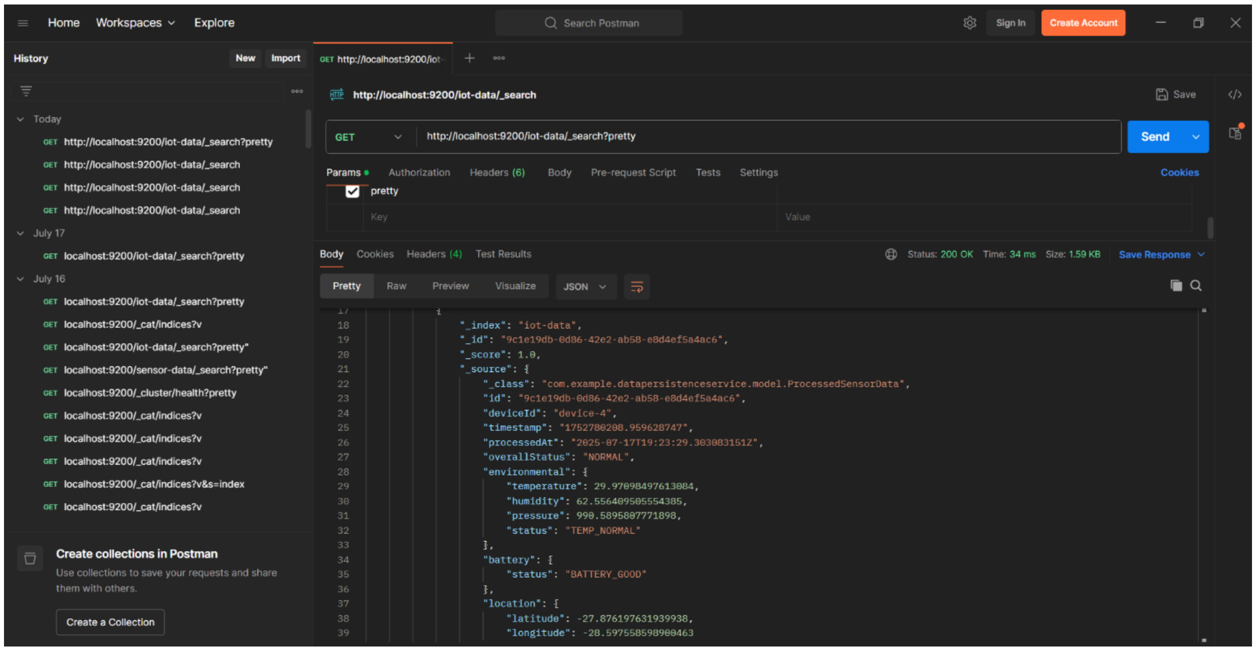
The Processing Service consumes validated sensor data from Kafka and transforms the raw payload into a detailed, structured format. It organizes the data into categories such as environmental conditions, location information, and battery status. The service applies filtering to exclude invalid or corrupt data based on predefined criteria, ensuring only realistic sensor readings proceed further. It evaluates sensor readings to identify any alerts—such as high temperature or low battery—and sets corresponding status flags along with an overall status message that clearly indicates any issues. Additional validation and quality checks guarantee data integrity before forwarding the enriched data. Once processed, the service publishes the enhanced messages to a Kafka topic for the Persistence Service, ensuring that only high-quality, meaningful data is stored for future analysis.

Here's a sensor data message generated after enriched:

```
{
  "id": "811dd86c-8236-4a5b-97c1-73d09966d323",
  "deviceId": "device-3",
  "timestamp": "2025-07-19T05:26:57.605832800Z",
  "processedAt": "2025-07-19T05:30:00.123456789Z",
  "overallStatus": "ALERT: Temperature High; ALERT: Low Battery",
  "environmental": {
    "temperature": 55.45640563700745,
    "humidity": 95.17558572167955,
    "pressure": 964.815053831346,
    "status": "TEMP_HIGH"
  },
  "battery": {
    "level": 15,
    "status": "LOW_BATTERY"
  },
  "location": {
    "latitude": 15.810095974995349,
    "longitude": -6.80282540983049
  }
}
```

5. Persistence Service

The Persistence Service subscribes to the enriched and validated data from the Processing Service via Kafka. It is responsible for storing IoT sensor data into Elasticsearch, a powerful distributed search and analytics engine. This service is configured to connect securely to the Elasticsearch cluster and maps incoming IoT data to appropriate Elasticsearch indices and document types. By saving data as documents in Elasticsearch, it enables efficient querying, visualization, and analysis of large volumes of time-series sensor data. This persistent storage layer supports real-time monitoring dashboards and historical data analytics for the overall system.


Home
Workspaces
Explore
Search Postman
Sign In
Create Account
History
New
Import
Today
GET http://localhost:9200/iot-data/_search?pretty
GET http://localhost:9200/iot-data/_search
GET http://localhost:9200/iot-data/_search
GET http://localhost:9200/iot-data/_search
July 17
GET localhost:9200/iot-data/_search?pretty
July 16
GET localhost:9200/iot-data/_search?pretty
GET localhost:9200/_cat/indices?v
GET localhost:9200/iot-data/_search?pretty"
GET localhost:9200/sensor-data/_search?pretty"
GET localhost:9200/_cluster/health?pretty
GET localhost:9200/_cat/indices?v
GET localhost:9200/_cat/indices?v
GET localhost:9200/_cat/indices?v
GET localhost:9200/_cat/indices?v&s=index
GET localhost:9200/_cat/indices?v
Create collections in Postman
Use collections to save your requests and share them with others.
Create a Collection
Save
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Key
Value
Body
Cookies
Headers (4)
Test Results
Save Response
Pretty
Raw
Preview
Visualize
JSON

http://localhost:9200/iot-data/_search

GET http://localhost:9200/iot-data/_search?pretty

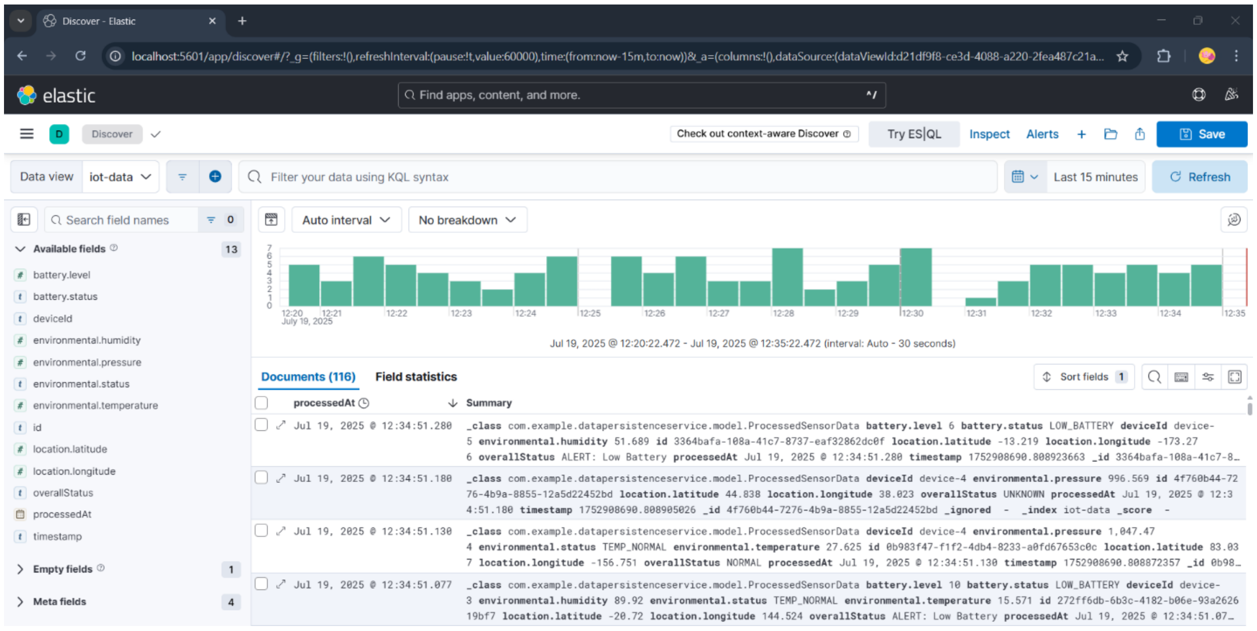
✓ pretty

Status: 200 OK Time: 34 ms Size: 1.59 KB

```
                {
                    "_index": "iot-data",
                    "_id": "9c1e19db-0d86-42e2-ab58-e8d4ef5a4ac6",
                    "_score": 1.0,
                    "_source": {
                        "_class": "com.example.datapersistenceservice.model.ProcessedSensorData",
                        "id": "9c1e19db-0d86-42e2-ab58-e8d4ef5a4ac6",
                        "deviceId": "device-4",
                        "timestamp": "1752780208.959628747",
                        "processedAt": "2025-07-17T19:23:29.303083151Z",
                        "overallStatus": "NORMAL",
                        "environmental": {
                            "temperature": 29.97098497613084,
                            "humidity": 62.556409505554385,
                            "pressure": 990.5895807771898,
                            "status": "TEMP_NORMAL"
                        },
                        "battery": {
                            "status": "BATTERY_GOOD"
                        },
                        "location": {
                            "latitude": -27.876197631939938,
                            "longitude": -28.597558598900463
```

6. Kibana Visualization

Kibana is used as the visualization and analytics interface for the IoT data stored in Elasticsearch. It enables creating interactive dashboards, charts, and graphs to explore real-time and historical sensor data. By connecting to the Elasticsearch indices where IoT documents are stored, Kibana allows users to filter, aggregate, and analyze data trends such as temperature fluctuations, battery status, and environmental conditions across devices. This helps stakeholders monitor system health, detect anomalies, and make data-driven decisions effectively.

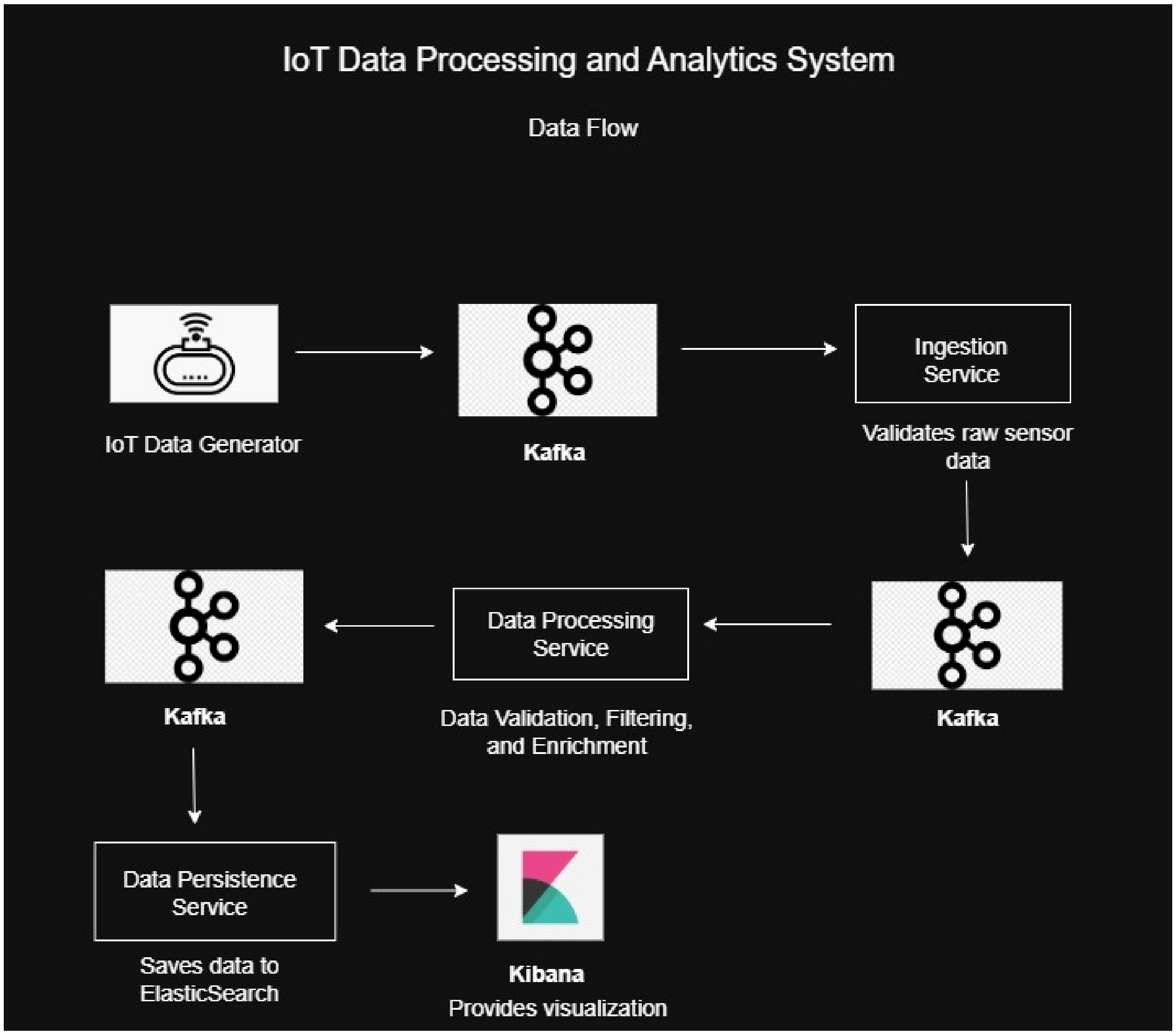


```
localhost:5601/app/discover#/?_g=(filters:!(),refreshInterval:(pause:!t,value:60000),time:(from:now-15m,to:now))&_a=(columns:!(),dataSource:(dataViewId:d21df9f8-ce3d-4088-a220-2fea487c21a...
```

```
Jul 19, 2025 @ 12:20:22.472 - Jul 19, 2025 @ 12:35:22.472 (interval: Auto - 30 seconds)
```

```
Jul 19, 2025 @ 12:34:51.280   _class com.example.datapersistenceservice.model.ProcessedSensorData battery.level 6 battery.status LOW_BATTERY deviceId device-
                              5 environmental.humidity 51.689 id 3364bafa-108a-41c7-8737-eaf32862dc0f location.latitude -13.219 location.longitude -173.27
                              6 overallStatus ALERT: Low Battery processedAt Jul 19, 2025 @ 12:34:51.280 timestamp 1752908690.808923663 _id 3364bafa-108a-41c7-8…
Jul 19, 2025 @ 12:34:51.180   _class com.example.datapersistenceservice.model.ProcessedSensorData deviceId device-4 environmental.pressure 996.569 id 4f760b44-72
                              76-4b9a-8855-12a5d22452bd location.latitude 44.838 location.longitude 38.023 overallStatus UNKNOWN processedAt Jul 19, 2025 @ 12:3
                              4:51.180 timestamp 1752908690.808905026 _id 4f760b44-7276-4b9a-8855-12a5d22452bd _ignored  -  _index iot-data _score  -
Jul 19, 2025 @ 12:34:51.130   _class com.example.datapersistenceservice.model.ProcessedSensorData deviceId device-4 environmental.pressure 1,047.47
                              4 environmental.status TEMP_NORMAL environmental.temperature 27.625 id 0b983f47-f1f2-4db4-8233-a0fd67653c0c location.latitude 83.03
                              7 location.longitude -156.751 overallStatus NORMAL processedAt Jul 19, 2025 @ 12:34:51.130 timestamp 1752908690.808872357 _id 0b98…
Jul 19, 2025 @ 12:34:51.077   _class com.example.datapersistenceservice.model.ProcessedSensorData battery.level 10 battery.status LOW_BATTERY deviceId device-
                              3 environmental.humidity 89.92 environmental.status TEMP_NORMAL environmental.temperature 15.571 id 272ff6db-6b3c-4182-b06e-93a2626
                              19bf7 location.latitude -20.72 location.longitude 144.524 overallStatus ALERT: Low Battery processedAt Jul 19, 2025 @ 12:34:51.07…
```

Data Flow


IoT Data Processing and Analytics System
Data Flow
IoT Data Generator
Kafka
Ingestion Service
Validates raw sensor data
Kafka
Data Processing Service
Data Validation, Filtering, and Enrichment
Kafka
Data Persistence Service
Saves data to ElasticSearch
Kibana
Provides visualization

Technologies Used

1. **Spring boot**

   Used to build modular and scalable microservices that enable rapid development. The application also utilizes the **MVC (Model-View-Controller)** architecture to organize code and separate concerns effectively.

2. **Apache Kafka**

   Serves as the central message broker enabling reliable, high-throughput, and fault-tolerant real-time data streaming between services. Kafka decouples producers and consumers ensuring scalability and resilience.

3. **Apache Zookeeper**

   Manages and coordinates Kafka brokers. Zookeeper handles tasks such as leader election, broker health monitoring, and metadata management, ensuring the reliability and availability of the Kafka cluster.

4. **Elasticsearch**

   Chosen as the primary data store for storing and indexing IoT sensor data. Elasticsearch provides fast, full-text search capabilities and analytics, enabling efficient retrieval and visualization of large volumes of time-series data.

5. **Kibana**

   Used for visualizing data stored in Elasticsearch. Kibana offers powerful, user-friendly dashboards to monitor IoT data trends and system health, supporting informed decision-making.

6. **Docker and Docker Compose**

   Used to containerize services and manage multi-container deployments, ensuring consistent environments across development, testing, and production.

Design Principles Used

01. Single Responsibility Principle (SRP)

Each service and class have a clear, focused responsibility.
Improves maintainability and makes testing easier by limiting the scope of changes.
Example: The IoT Data Generator only creates data, the Ingestion Service only validates and forwards, the Processing Service enriches and transforms data, and the Persistence Service handles saving to Elasticsearch.

02. Separation of Concerns

Different concerns (generation, ingestion, processing, persistence) are separated into distinct services.
Facilitates independent development, deployment, and scaling of microservices.

03. Dependency Injection (DI)

Used Spring's DI to manage dependencies between components like KafkaTemplate, services, and validators.
Decouples code, improves testability, and promotes flexibility.

04. Open/Closed Principle (OCP)

Services can be extended with new features (e.g., adding new validations or data enrichment) without modifying existing code significantly.
Enhances adaptability to changing requirements without breaking existing functionality.

05. Interface Segregation Principle (ISP)

Services expose only the methods necessary for their purpose, avoiding “fat” interfaces.
Prevents clients from depending on unnecessary methods, keeping contracts clean and focused.

06. Fail-Fast and Defensive Programming

Validation services and Kafka health checks help detect errors early and prevent faulty data from propagating.
Ensures data quality and system robustness.

07. Event-Driven Architecture

Kafka acts as a decoupled message broker between services, promoting asynchronous communication.
Enhances scalability and fault tolerance, allowing components to evolve independently.

Deployment Setup

Install Required Software

- **Docker Desktop**

Download and install Docker Desktop from
https://www.docker.com/products/docker-desktop
Ensure Docker is running before proceeding.

- **Elasticsearch & Kibana**

The system uses Elasticsearch and Kibana for data storage and visualization.
These services are included in the docker-compose.yml and will start automatically.

Clone the Repository

```
https://github.com/Niro-dev91/iot-stream-processor.git
cd iot-stream-processor
```

Start Services with Docker Compose

Open a terminal and navigate to the root directory of the project (where the docker-compose.yml file is located), then run the following command to build and start all services:

```
docker-compose up --build
```

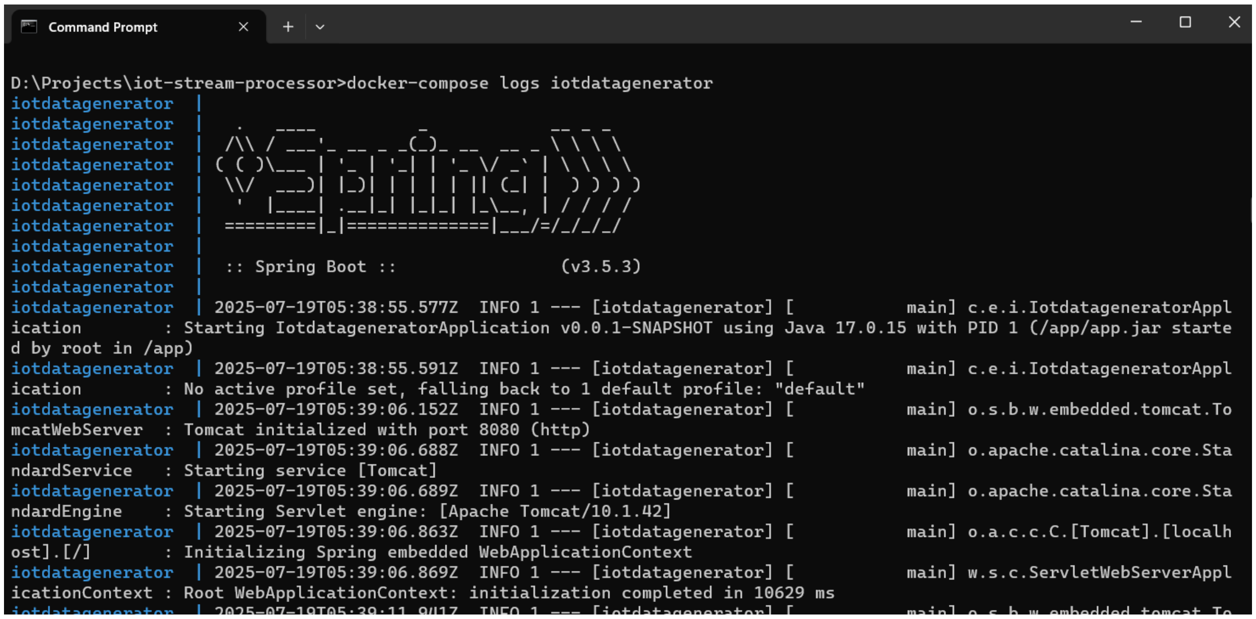
This will build and start all required containers, including:

- Kafka Broker
- Zookeeper
- Elasticsearch
- Kibana
- all microservices

Note:
When starting Elasticsearch using Docker, it may take a few moments to fully initialize before Kibana or other services can connect and function properly. Please wait a short while after starting the containers before accessing the Kibana dashboard or querying Elasticsearch. This is normal behavior when running the database and search services inside Docker.

Docker- compose log commands

> **docker-compose logs**
> **docker-compose logs iotdatagenerator**
> **docker-compose logs ingestion**
> **docker-compose logs processingservice**
> **docker-compose logs datapersistenceservice**



```
D:\Projects\iot-stream-processor>docker-compose logs iotdatagenerator
iotdatagenerator  |
iotdatagenerator  |   .   ____          _            __ _ _
iotdatagenerator  |  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
iotdatagenerator  | ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
iotdatagenerator  |  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
iotdatagenerator  |   '  |____| .__|_| |_|_| |_\__, | / / / /
iotdatagenerator  |  =========|_|==============|___/=/_/_/_/
iotdatagenerator  |
iotdatagenerator  |  :: Spring Boot ::                (v3.5.3)
iotdatagenerator  |
iotdatagenerator  | 2025-07-19T05:38:55.577Z  INFO 1 --- [iotdatagenerator] [           main] c.e.i.IotdatageneratorAppl
ication         : Starting IotdatageneratorApplication v0.0.1-SNAPSHOT using Java 17.0.15 with PID 1 (/app/app.jar starte
d by root in /app)
iotdatagenerator  | 2025-07-19T05:38:55.591Z  INFO 1 --- [iotdatagenerator] [           main] c.e.i.IotdatageneratorAppl
ication         : No active profile set, falling back to 1 default profile: "default"
iotdatagenerator  | 2025-07-19T05:39:06.152Z  INFO 1 --- [iotdatagenerator] [           main] o.s.b.w.embedded.tomcat.To
mcatWebServer   : Tomcat initialized with port 8080 (http)
iotdatagenerator  | 2025-07-19T05:39:06.688Z  INFO 1 --- [iotdatagenerator] [           main] o.apache.catalina.core.Sta
ndardService    : Starting service [Tomcat]
iotdatagenerator  | 2025-07-19T05:39:06.689Z  INFO 1 --- [iotdatagenerator] [           main] o.apache.catalina.core.Sta
ndardEngine     : Starting Servlet engine: [Apache Tomcat/10.1.42]
iotdatagenerator  | 2025-07-19T05:39:06.863Z  INFO 1 --- [iotdatagenerator] [           main] o.a.c.c.C.[Tomcat].[localh
ost].[/]        : Initializing Spring embedded WebApplicationContext
iotdatagenerator  | 2025-07-19T05:39:06.869Z  INFO 1 --- [iotdatagenerator] [           main] w.s.c.ServletWebServerAppl
icationContext : Root WebApplicationContext: initialization completed in 10629 ms
iotdatagenerator  | 2025-07-19T05:39:11.941Z  INFO 1 --- [iotdatagenerator] [           main] o.s.b.w.embedded.tomcat.To
```

Run Spring Boot Applications

Alternatively, you can run each service locally**:**

Download mvnd

https://github.com/apache/maven-mvnd/releases

```
# iot data generator service
cd iotdatagenerator
./mvnd spring-boot:run

# Ingestion Service
cd ingestion
./mvnd spring-boot:run

# Processing Service
cd processingservice
./mvnd spring-boot:run

# Persistence Service
cd datapersistenceservice
./mvnd spring-boot:run
```

Important considerations

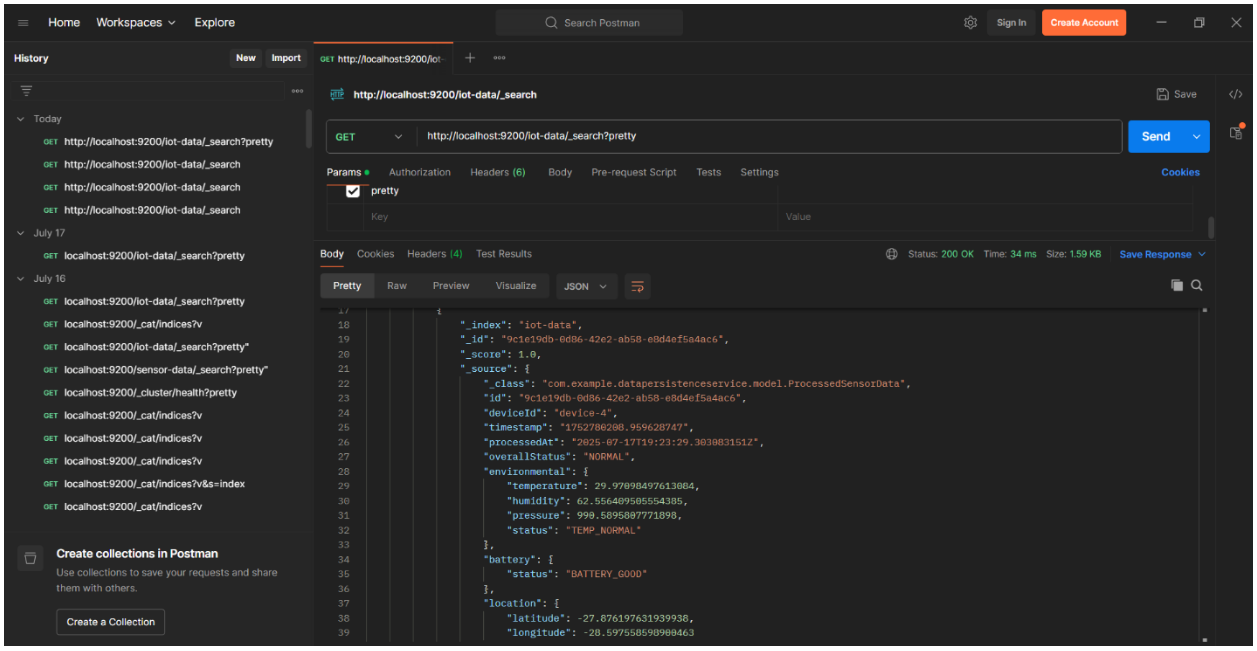
If running the project without Docker, ensure that all dependent services such as Kafka, Elasticsearch, and others are manually installed and running on the local machine. Also, make sure the microservices are configured to point to the correct local endpoints (e.g., localhost:9092 for Kafka and localhost:9200 for Elasticsearch).

Recommended: Use Docker and Docker Compose to run the full system easily. This ensures consistent environments, simplifies service orchestration, and avoids manual setup issues.

Search queries in Elasticsearch

- Using REST API (cURL or Postman)

```
curl -X GET "http://localhost:9200/iot-data/_search?pretty"
-H 'Content-Type: application/json' -d'
  {
      "query": {
      "match_all": {}
      }
   }'
```



History

Today
- GET http://localhost:9200/iot-data/_search?pretty
- GET http://localhost:9200/iot-data/_search
- GET http://localhost:9200/iot-data/_search
- GET http://localhost:9200/iot-data/_search

July 17
- GET localhost:9200/iot-data/_search?pretty

July 16
- GET localhost:9200/iot-data/_search?pretty
- GET localhost:9200/_cat/indices?v
- GET localhost:9200/iot-data/_search?pretty"
- GET localhost:9200/sensor-data/_search?pretty"
- GET localhost:9200/_cluster/health?pretty
- GET localhost:9200/_cat/indices?v
- GET localhost:9200/_cat/indices?v
- GET localhost:9200/_cat/indices?v
- GET localhost:9200/_cat/indices?v&s=index
- GET localhost:9200/_cat/indices?v

Create collections in Postman
Use collections to save your requests and share them with others.

http://localhost:9200/iot-data/_search

GET http://localhost:9200/iot-data/_search?pretty

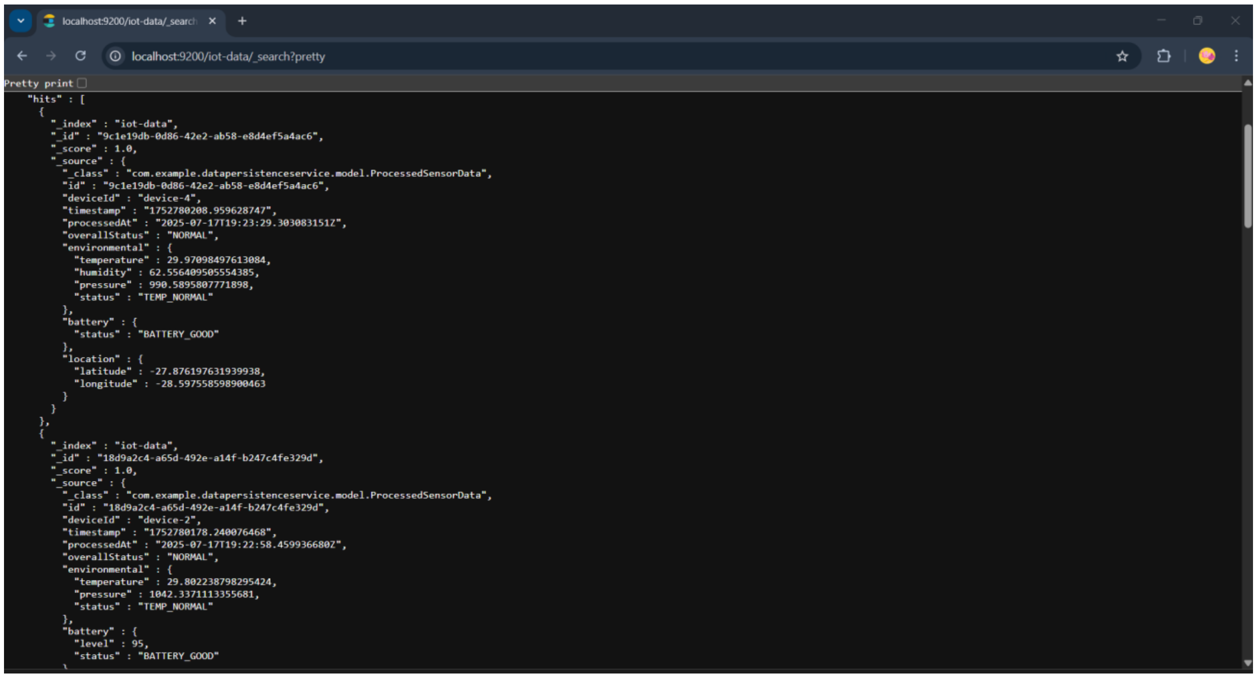
Params: pretty ✓

Status: 200 OK Time: 34 ms Size: 1.59 KB

```
            {
                "_index": "iot-data",
                "_id": "9c1e19db-0d86-42e2-ab58-e8d4ef5a4ac6",
                "_score": 1.0,
                "_source": {
                    "_class": "com.example.datapersistenceservice.model.ProcessedSensorData",
                    "id": "9c1e19db-0d86-42e2-ab58-e8d4ef5a4ac6",
                    "deviceId": "device-4",
                    "timestamp": "1752780208.959628747",
                    "processedAt": "2025-07-17T19:23:29.303083151Z",
                    "overallStatus": "NORMAL",
                    "environmental": {
                        "temperature": 29.97098497613084,
                        "humidity": 62.556409505554385,
                        "pressure": 990.5895807771898,
                        "status": "TEMP_NORMAL"
                    },
                    "battery": {
                        "status": "BATTERY_GOOD"
                    },
                    "location": {
                        "latitude": -27.876197631939938,
                        "longitude": -28.597558598900463
```

- **Basic browser-accessible**

  http://localhost:9200/iot-data/_search?pretty



localhost:9200/iot-data/_search
localhost:9200/iot-data/_search?pretty
Pretty print

```
  "hits" : [
    {
      "_index" : "iot-data",
      "_id" : "9c1e19db-0d86-42e2-ab58-e8d4ef5a4ac6",
      "_score" : 1.0,
      "_source" : {
        "_class" : "com.example.datapersistenceservice.model.ProcessedSensorData",
        "id" : "9c1e19db-0d86-42e2-ab58-e8d4ef5a4ac6",
        "deviceId" : "device-4",
        "timestamp" : "1752780208.959628747",
        "processedAt" : "2025-07-17T19:23:29.303083151Z",
        "overallStatus" : "NORMAL",
        "environmental" : {
          "temperature" : 29.97098497613084,
          "humidity" : 62.556409505554385,
          "pressure" : 990.5895807771898,
          "status" : "TEMP_NORMAL"
        },
        "battery" : {
          "status" : "BATTERY_GOOD"
        },
        "location" : {
          "latitude" : -27.876197631939938,
          "longitude" : -28.597558598900463
        }
      }
    },
    {
      "_index" : "iot-data",
      "_id" : "18d9a2c4-a65d-492e-a14f-b247c4fe329d",
      "_score" : 1.0,
      "_source" : {
        "_class" : "com.example.datapersistenceservice.model.ProcessedSensorData",
        "id" : "18d9a2c4-a65d-492e-a14f-b247c4fe329d",
        "deviceId" : "device-2",
        "timestamp" : "1752780178.240076468",
        "processedAt" : "2025-07-17T19:22:58.459936680Z",
        "overallStatus" : "NORMAL",
        "environmental" : {
          "temperature" : 29.802238798295424,
          "pressure" : 1042.3371113355681,
          "status" : "TEMP_NORMAL"
        },
        "battery" : {
          "level" : 95,
          "status" : "BATTERY_GOOD"
```

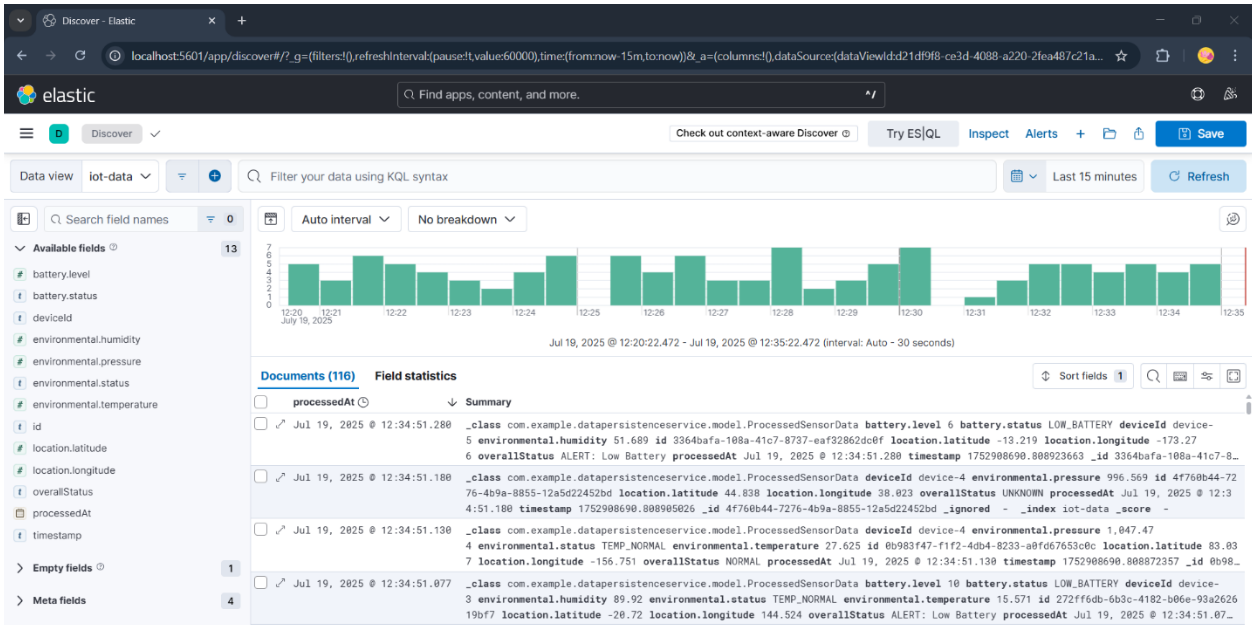
Access Kibana Dashboard

Use Kibana to visualize and analyze the ingested IoT data.

Open your browser and go to:

http://localhost:5601

- Navigate to Stack Management > index patterns(e.g., iot-data-*) to create or view Elasticsearch indices.
- or
- Navigate to Discover > Create data view > select index pattern (iot-data-*)



```
localhost:5601/app/discover#/?_g=(filters:!(),refreshInterval:(pause:!t,value:60000),time:(from:now-15m,to:now))&_a=(columns:!(),dataSource:(dataViewId:d21df9f8-ce3d-4088-a220-2fea487c21a...
```

Data view: iot-data — Last 15 minutes

Jul 19, 2025 @ 12:20:22.472 - Jul 19, 2025 @ 12:35:22.472 (interval: Auto - 30 seconds)

Documents (116)

```
processedAt                   Summary
Jul 19, 2025 @ 12:34:51.280   _class com.example.datapersistenceservice.model.ProcessedSensorData battery.level 6 battery.status LOW_BATTERY deviceId device-
                              5 environmental.humidity 51.689 id 3364bafa-108a-41c7-8737-eaf32862dc0f location.latitude -13.219 location.longitude -173.27
                              6 overallStatus ALERT: Low Battery processedAt Jul 19, 2025 @ 12:34:51.280 timestamp 1752908690.808923663 _id 3364bafa-108a-41c7-8…
Jul 19, 2025 @ 12:34:51.180   _class com.example.datapersistenceservice.model.ProcessedSensorData deviceId device-4 environmental.pressure 996.569 id 4f760b44-72
                              76-4b9a-8855-12a5d22452bd location.latitude 44.838 location.longitude 38.023 overallStatus UNKNOWN processedAt Jul 19, 2025 @ 12:3
                              4:51.180 timestamp 1752908690.808905026 _id 4f760b44-7276-4b9a-8855-12a5d22452bd _ignored  -  _index iot-data _score  -
Jul 19, 2025 @ 12:34:51.130   _class com.example.datapersistenceservice.model.ProcessedSensorData deviceId device-4 environmental.pressure 1,047.47
                              4 environmental.status TEMP_NORMAL environmental.temperature 27.625 id 0b983f47-f1f2-4db4-8233-a0fd67653c0c location.latitude 83.03
                              7 location.longitude -156.751 overallStatus NORMAL processedAt Jul 19, 2025 @ 12:34:51.130 timestamp 1752908690.808872357 _id 0b98…
Jul 19, 2025 @ 12:34:51.077   _class com.example.datapersistenceservice.model.ProcessedSensorData battery.level 10 battery.status LOW_BATTERY deviceId device-
                              3 environmental.humidity 89.92 environmental.status TEMP_NORMAL environmental.temperature 15.571 id 272ff6db-6b3c-4182-b06e-93a2626
                              19bf7 location.latitude -20.72 location.longitude 144.524 overallStatus ALERT: Low Battery processedAt Jul 19, 2025 @ 12:34:51.07…
```

Decisions & Justifications

1. **Microservices Architecture:**

The system is designed as a set of independent microservices (IoT Data Generator, Ingestion, Processing, Persistence) to improve modularity, scalability, and ease of maintenance. This separation allows teams to develop, deploy, and scale components independently based on load and functionality.

2. **Apache Kafka as the Messaging Backbone:**

Kafka is chosen as a distributed, fault-tolerant message broker to decouple services and handle high-throughput real-time data streams. It ensures reliable data flow between producers and consumers and supports asynchronous processing, improving system responsiveness.

3. Spring Boot:

Spring Boot simplifies microservice development with auto-configuration and dependency injection.

4. Data Modeling and Validation:

The system enforces strict data validation at ingestion and processing stages to maintain data integrity and prevent propagation of invalid or corrupted sensor data.

5. Elasticsearch for Data Storage and Kibana for Visualization:

The choice of Elasticsearch allows efficient storage and fast querying of time-series sensor data. Kibana provides a powerful, user-friendly visualization layer to monitor sensor trends and system metrics. Both are deployed via Docker containers to enable consistent and isolated environments with simplified setup and management.

6. Containerization with Docker:

Using Docker for Elasticsearch and Kibana, as well as for the microservices, ensures consistent runtime environments, easier deployment, and scalability. Docker Compose orchestrates service dependencies and startup order, improving reliability during system initialization.

7.Focus on Scalability and Fault Tolerance:

By using distributed Kafka brokers and containerized microservices, the system is designed to handle increased data volumes and recover gracefully from component failures.

The system is designed to efficiently handle data from thousands of IoT devices by using Kafka for fast, reliable messaging and containerized microservices for modularity and scalability. With Docker containerization and Kafka at its core, it can easily integrate with cloud container services that support auto-scaling, high availability, and fault tolerance. This architecture provides a strong foundation for future enhancements such as automated scaling, distributed deployment, and advanced cloud monitoring.

Testing Strategy

Unit Tests: Each service is tested using JUnit and Mockito.

Integration Tests: Kafka message flow and Elasticsearch persistence are tested with embedded Kafka and test containers.

Test Profile: Uses @ActiveProfiles("test") to isolate test configs.

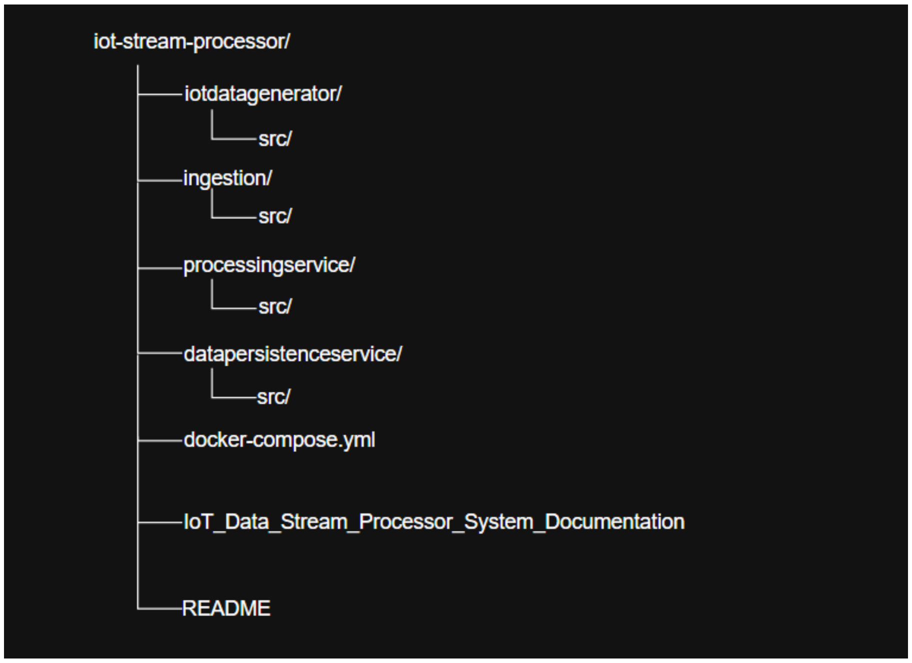
How to Run Tests:

```
./mvnd test
```

Tests will cover:

- Kafka producer/consumer flow
- Business logic validation
- Elasticsearch indexing

Project Structure



```
iot-stream-processor/
    ├──iotdatagenerator/
    │      └──src/
    ├──ingestion/
    │      └──src/
    ├──processingservice/
    │      └──src/
    ├──datapersistenceservice/
    │      └──src/
    ├──docker-compose.yml
    │
    ├──IoT_Data_Stream_Processor_System_Documentation
    │
    └──README
```

Conclusion

This IoT data processing system effectively manages large volumes of real-time data from numerous devices by leveraging Kafka's robust messaging capabilities and a microservices architecture built with Docker containers. It ensures reliable, scalable, and low-latency data ingestion, processing, and storage. The design supports easy integration with cloud infrastructure, enabling high availability and fault tolerance. This flexible foundation allows for future enhancements like auto-scaling, distributed deployments, and advanced monitoring, making it well-suited for growing IoT ecosystems.