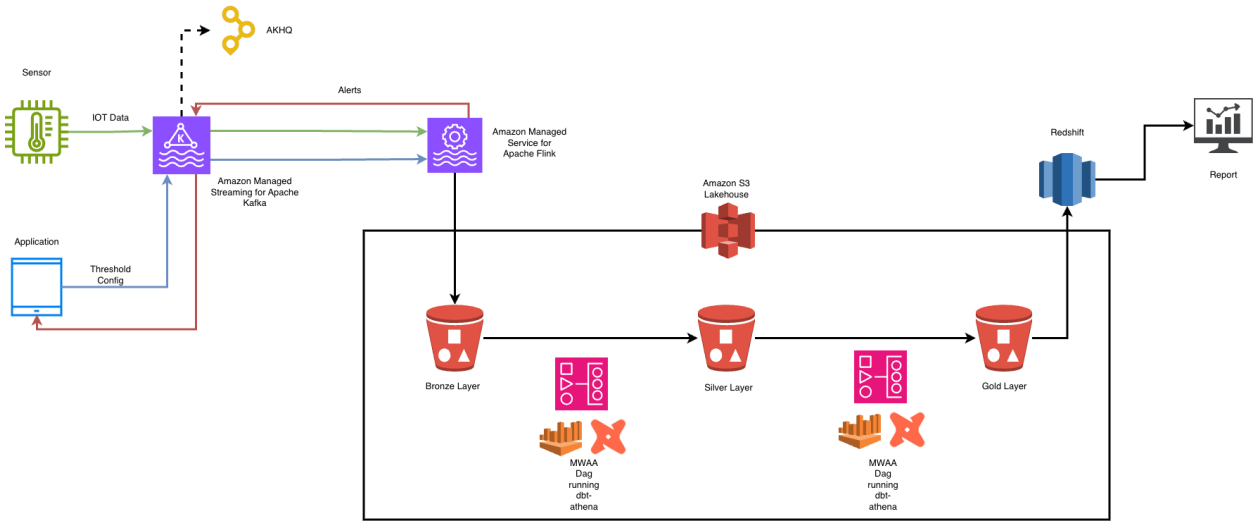
IOT Monitoring Data Pipeline – Documentation

This document describes the data pipeline for monitoring a room sensor. The project aims to solve:
- Sends alerts to application users if the temperature thresholds or light during no-occupancy thresholds, configured by the user, are breached.
- Store the hourly aggregated metrics from the sensor for analysis and reporting

Architecture Diagram



Description

The data pipeline, as the problem, can be split into two sections.
- A hot path (real-time) that monitors the sensor values and sends alerts based on the temperature & light thresholds
- A near-real-time processing path that prepares the hourly aggregated data for further reporting and analysis

Architecture Components

Sensor (Simulated)

A room sensor is simulated using a python script and a csv file containing measures of temperature, light, co2, occupancy etc. The data is pushed as json , including a dummy

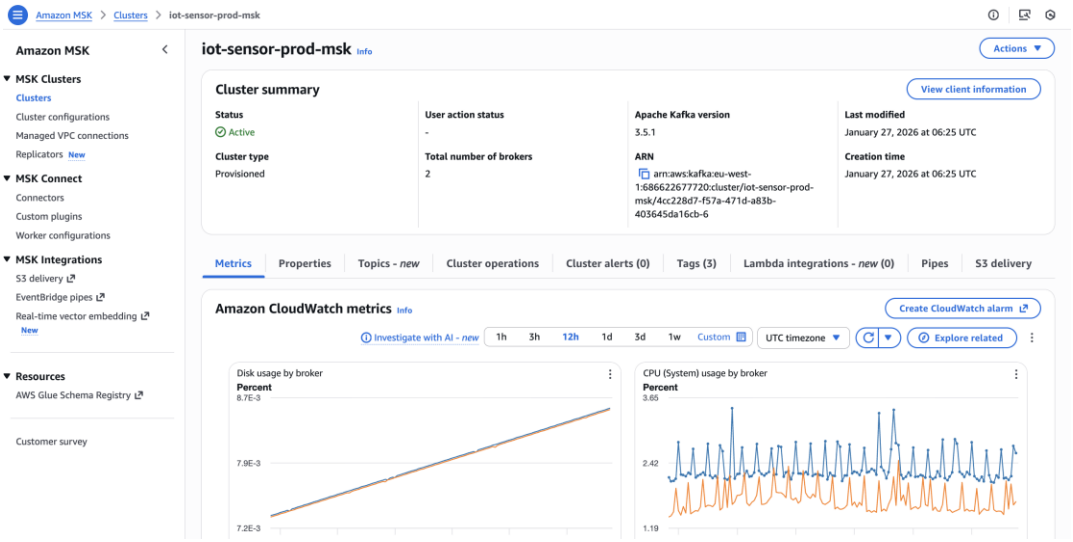
device_id, to a kafka topic - "iot-sensor-readings".

User Application (Simulated)

The user application is simulated using a python script and a csv file containing temperature - warning & violation, and no-occupancy light thresholds, per user and device. The data is pushed as json to a kafka topic - "user-threshold-config".

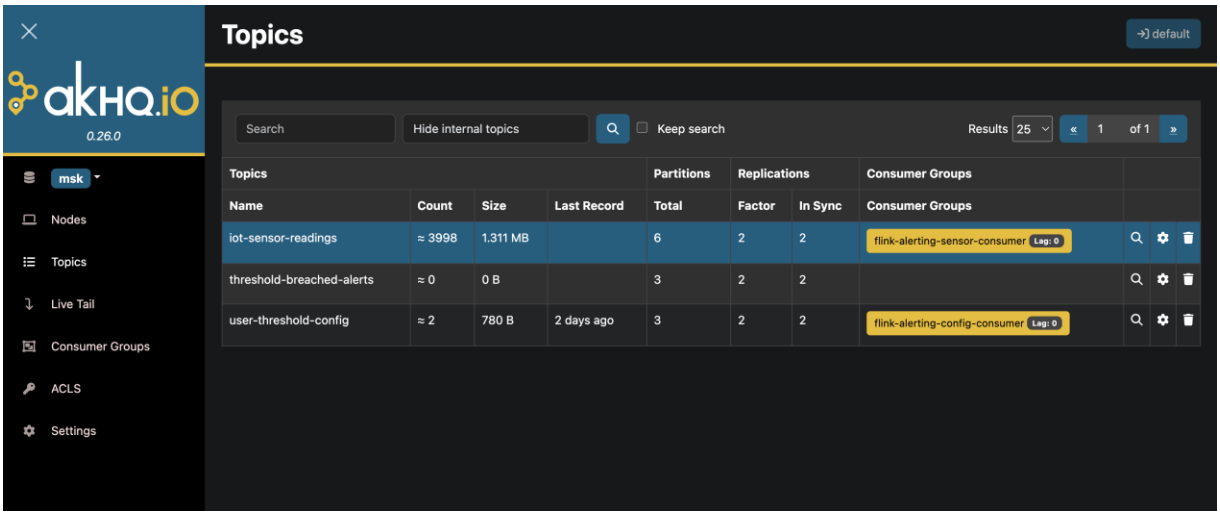
Amazon Managed Streaming for Kafka (MSK)

Amazon's MSK provides a fully managed service for Kafka applications. The infrastructure is deployed onto AWS using terraform.
The individual kafka topics along with configuration, such as retention, are also deployed via terraform (kafka provider).



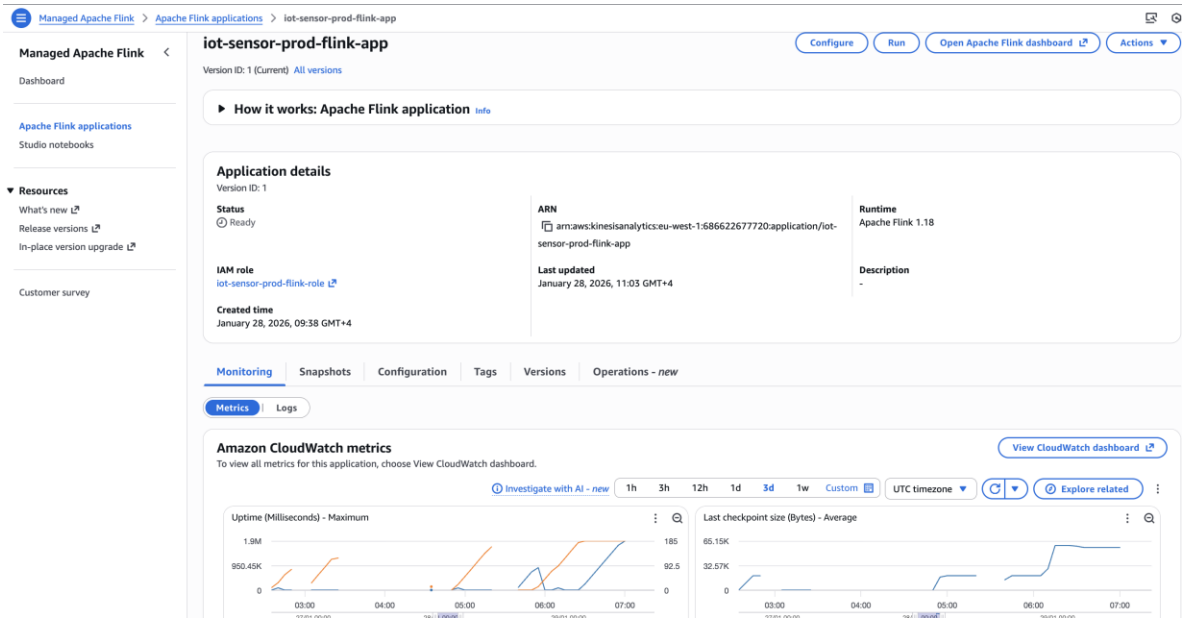
AKHQ

AKHQ is deployed via docker instance on EC2, to provide visibility on the kafka topics, consumers, lag etc.



Amazon Managed Service for Apache Flink

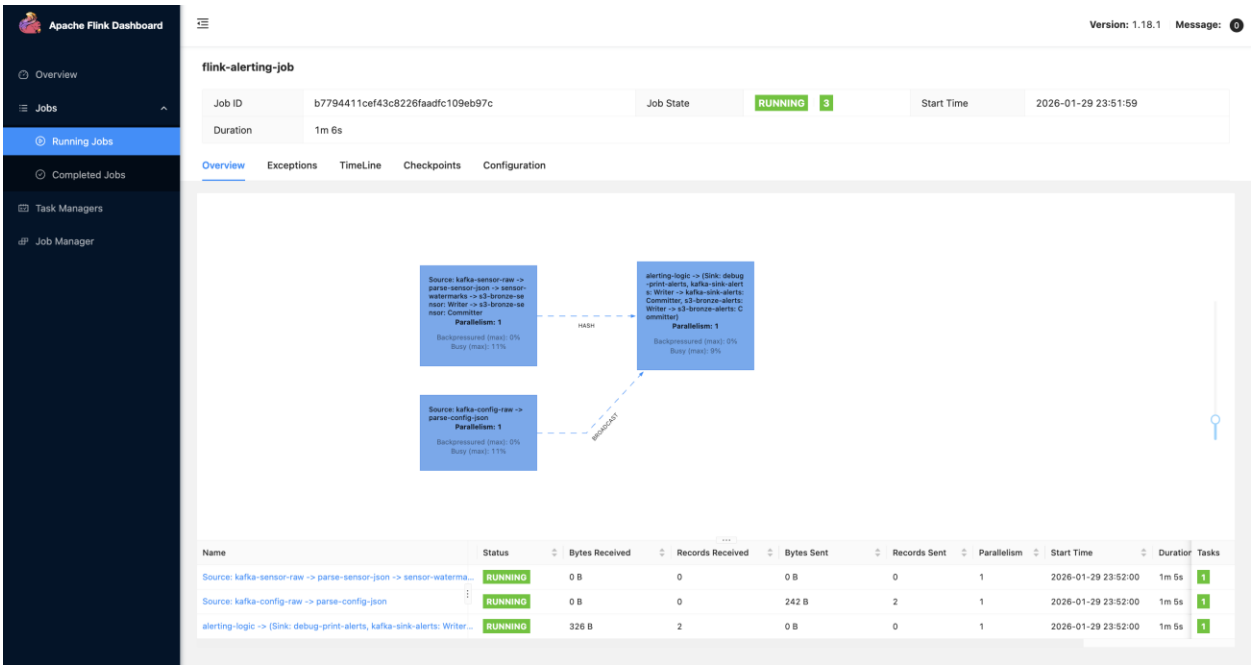
Amazon Managed Service for Apache Flink makes it easier to run build and run real-time data processing Flink applications, by provisioning and configuring Flink clusters and orchestrating job management.



Flink Application

The flink application, coded in java, serves two purposes:

- It processes the data from the sensor in real time, comparing with the user threshold configurations, and alerts in case of breaches.
- It also sinks the data into s3, which serves as the bronze layer of the lakehouse.

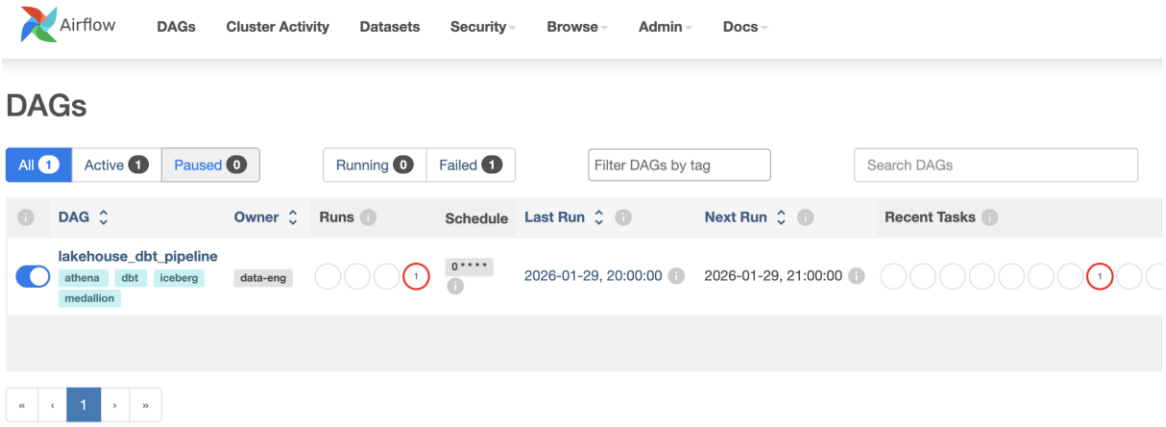


Lakehouse

S3 acts as the storage for the lakehouse. Separate s3 buckets are set for each layer of the medallion architecture. Athena is the query engine that is used to query the Apache Iceberg tables on top of the s3 files, while the table registry is stored in (Glue) Catalogs.

Dbt + MWAA

DBT is the framework that helps transform the data from the Bronze Layer of the warehouse into Silver and Gold Layers. This is achieved using the dbt-athena adapter. The dbt tool, apart from building the tables, also aids testing and data anomaly detections. MWAA (Amazon Managed Airflow) is the orchestration tool that allows to schedule and run the dbt tasks to enrich the silver & gold layers. These batch jobs can be scheduled based on a suitable time schedule.



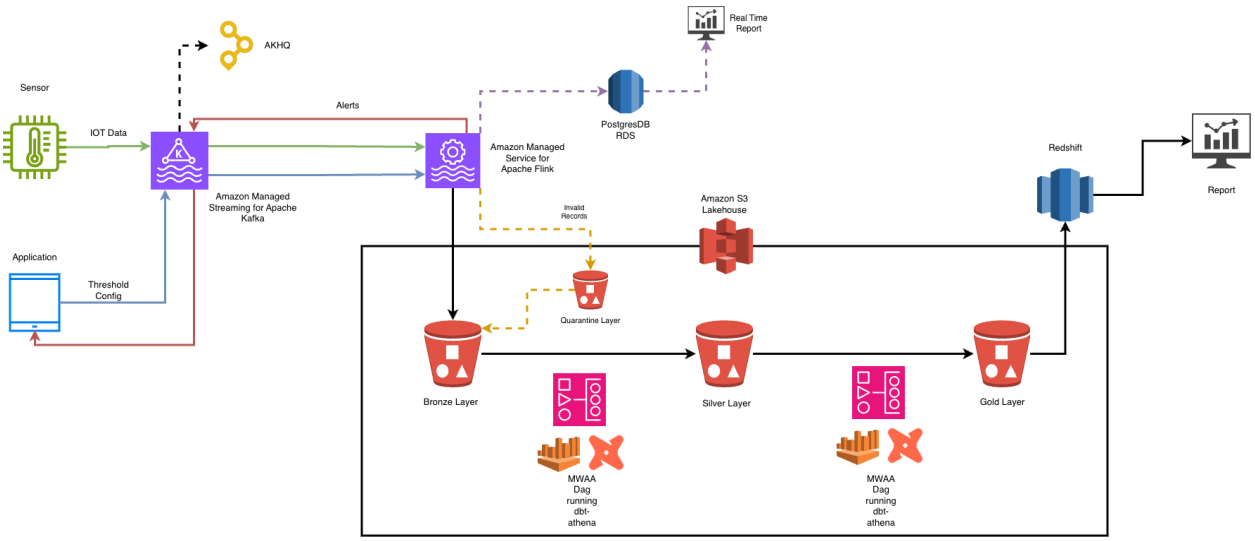
Redshift

Redshift is AWS's Data Warehouse solution, that can act as simply a powerful query engine on top of the gold layer of the Lakehouse to power reports. In addition to this, it is also a stage for further analytical processing and transformation.

BI Tool

The BI Reporting Tool should simply query the semantic layer tables from Redshift, thus reducing dashboard refresh delays.

Additional Components



Quarantine Layer

The Flink application can be modified to filter out any invalid records, that do not follow the expected schema and push them into a Quarantine Layer. The data from the Quarantine Layer can be cleansed & moved to Bronze Layer using adhoc processing

Real Time Reporting

A PostgresDB hosted via RDS can act a layer for real-time reporting of recent data without delays. Using mechanisms like generated columns & materialized views, and regular purging, a real-time report can be generated with minimum latency.

Github Repository Layout

data	Contains the two input data files: iot_data.csv (Data from IOT sensor) threshold_config.csv (Threshold configurations set by user)
producers	Contains the python scripts to simulate the Sensor: To emit Sensor Data into Kafka Application: To emit Thresholds into Kafka
infra-terraform-aws	Contains all the terraform code to deploy AWS & kafka components of the infrastructure. This includes everything from the VPC and onwards.
flink	Contains the code to build the Apache Flink Java application. This application is used to check for time-based threshold breaches and send alerts to Kafka topic. In addition, it also sinks the sensor data into the s3, which would act as the Bronze Layer of the Lakehouse
dbt	Contains the dbt environment (with athena adapter), and the dbt models corrsponding to silver & gold layer iceberg tables of S3 Lakehouse.
airflow	Contains DAGs code, that orchestrates the dbt run tasks, and supporting components for MWAA.

Deployment Steps

This section describes how to deploy the data pipeline from the github repository.
Pre-requisites to deploy:

- AWS CLI + User with administrator privileges
- Python
- Java + Gradle
- Terraform

Upload certificates to AWS Certificate Manager using the below commands (This is required to set up VPN to deploy kafka topics) :

```
git clone https://github.com/OpenVPN/easy-rsa.git
cd easy-rsa/easyrsa3
./easyrsa init-pki
./easyrsa build-ca nopass
./easyrsa build-server-full server.domain.tld nopass
/easyrsa build-client-full vpn-client nopass



aws acm import-certificate \
--region eu-west-1 \
--certificate fileb://pki/issued/server.domain.tld.crt \
--private-key fileb://pki/private/server.domain.tld.key \
--certificate-chain fileb://pki/ca.crt



aws acm import-certificate \
--region eu-west-1 \
--certificate fileb://pki/issued/vpn-client.crt \
--private-key fileb://pki/private/vpn-client.key \
--certificate-chain fileb://pki/ca.crt
```

Update the `infra-terraform-aws/terraform.tfvars` file accordingly.

Then, run terraform code as follows:

```
cd infra-terraform-aws
terraform init
terraform plan
terraform apply
```

Kafka deployments, would fail (if not connected to VPN).

Download the OVPN config from AWS console, append the client certificate and key. Then retry terraform deployment of kafka topics.

Verify components are up and running.

You might need to update default values for parameters in
` flink/alerting_job/src/main/java/com/example/alerts/AppConfig.java `

Re-build the java app using gradle, and zip the file.

```
gradle wrapper
./gradlew clean shadowJar

cd build/libs
zip -r flink-app-dev.zip flink-alerting-job-0.1.0.jar
```

Redeploy using terraform. Verify components are up and running.

Data can be pushed to source kafka topics using the producer scripts.

```
cd producers
python3 -m venv .venv
pip install -r requirements.txt



#To push thresholds
python -m producers.threshold_config_producer.main --csv ./data/threshold_config.csv

#To push sensor data
python -m producers.iot_producer.main --csv ./data/iot_data.csv --eps 10
```

To create bronze layer tables, run in Athena:

```
CREATE EXTERNAL TABLE IF NOT EXISTS `iot-sensor-prod_lake`.`bronze_iot_sensor_readings` (
`event_time` string,
`device_id` string,
`temperature` double,
`humidity` double,
`light` double,
`co2` double,
`humidity_ratio` double,
`occupancy` int
)
PARTITIONED BY (`dt` string, `hour` string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
'ignore.malformed.json' = 'TRUE',
```

```
'dots.in.keys' = 'FALSE',
'case.insensitive' = 'TRUE'
)
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://iot-sensor-prod-bronze/bronze/iot_sensor_readings/'
TBLPROPERTIES (
'classification' = 'json',
'has_encrypted_data' = 'false'
);



CREATE EXTERNAL TABLE IF NOT EXISTS `iot-sensor-prod_lake`.`bronze_threshold_breached_alerts` (
`alert_time` string,
`user_id` string,
`device_id` string,
`rule` string,
`severity` string,
`metric` string,
`observed_value` double,
`threshold_value` double,
`window_start_ms` bigint,
`window_end_ms` bigint
)
PARTITIONED BY (`dt` string, `hour` string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES (
'ignore.malformed.json' = 'TRUE',
'dots.in.keys' = 'FALSE',
'case.insensitive' = 'TRUE'
)
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://iot-sensor-prod-bronze/bronze/threshold_breached_alerts/'
TBLPROPERTIES ('classification' = 'json',
'has_encrypted_data' = 'false');

MSCK REPAIR TABLE `iot-sensor-prod_lake`.`bronze_iot_sensor_readings`;
MSCK REPAIR TABLE `iot-sensor-prod_lake`.`bronze_threshold_breached_alerts`;
```