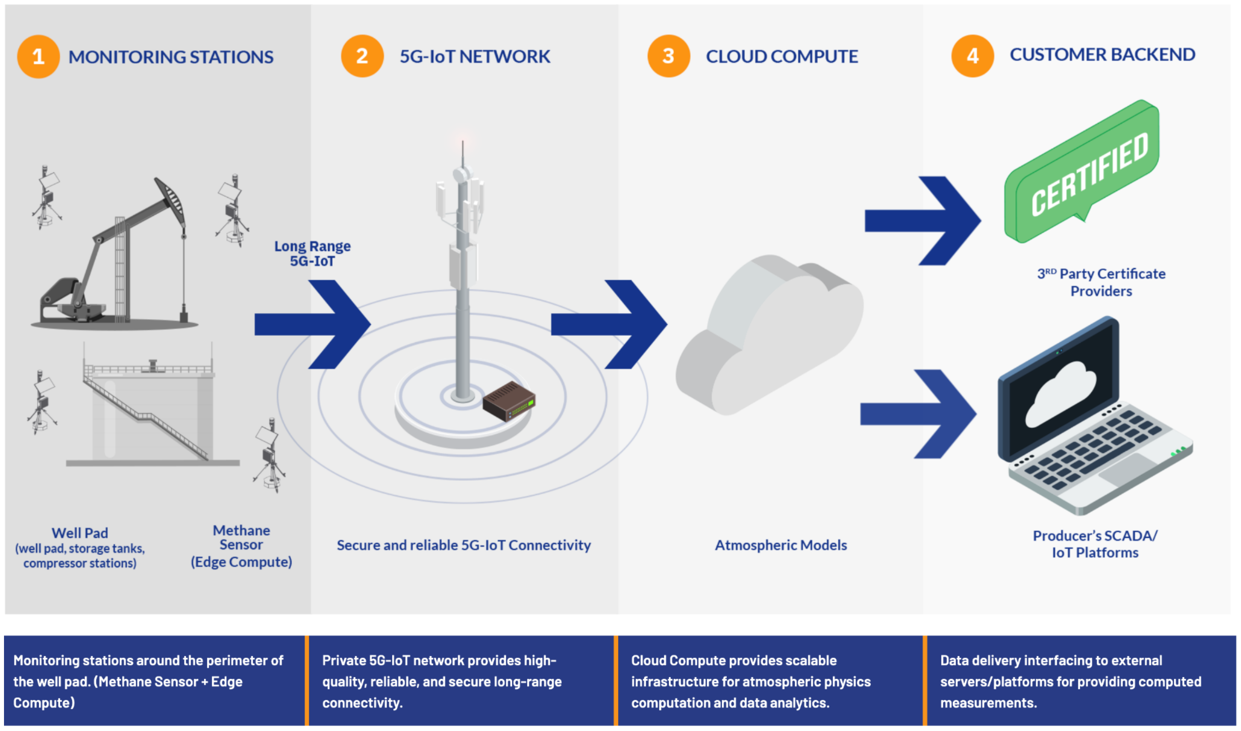
IoT Data Pipeline Design Doc

Project Description: Building end-to-end IoT solution to measure methane emissions


1 MONITORING STATIONS
2 5G-IoT NETWORK
3 CLOUD COMPUTE
4 CUSTOMER BACKEND
Long Range 5G-IoT
CERTIFIED
3RD Party Certificate Providers
Well Pad (well pad, storage tanks, compressor stations)
Methane Sensor (Edge Compute)
Secure and reliable 5G-IoT Connectivity
Atmospheric Models
Producer's SCADA/ IoT Platforms

Monitoring stations around the perimeter of the well pad. (Methane Sensor + Edge Compute)

Private 5G-IoT network provides high-quality, reliable, and secure long-range connectivity.

Cloud Compute provides scalable infrastructure for atmospheric physics computation and data analytics.

Data delivery interfacing to external servers/platforms for providing computed measurements.

Purpose: The idea of this document is to explore a modern and simple solution for a data pipeline that processes IoT data from a large number of devices, specifically sensors.

Design Questions

Bigger Picture Questions

- What is the current connection point between the 5G network and the cloud?

- What format is the data? What format should it be transformed to?
- How are we currently storing this incoming data?
- How are we currently reading this data?
- How much data is coming in? How much should we expect to scale to?

- How are we currently hosting the client application?

Device Specific Questions

- How sophisticated are the sensors? Do they have an OS?
- How do they send information? Do they have certificates?
- How much computation do you want done on the device?
- How are the devices currently being managed?
- What is the process for onboarding a new device?

Solution Type

We have two primary pathways to consider for our IoT solution:

The first option entails embracing a fully native AWS solution, which is particularly enticing given the extensive array of recently developed managed services tailored specifically for IoT solutions and device management.

The second path involves opting for a more customizable, cloud-independent solution that has the flexibility to be deployed across multiple cloud platforms.

Fully Native AWS Solution

Pros

Development Speed: AWS provides a wide range of managed services that can accelerate development by reducing the need for manual setup and configuration.

Tight Integration: A fully native AWS solution can take advantage of the seamless integration between AWS services, making it easier to build and maintain your application.

Scalability: AWS offers auto-scaling capabilities that allow your application to automatically adjust resources based on demand. This can simplify scalability management.

Cons

Vendor Lock-In: One drawback is that you might become more tied to AWS services, which could make it challenging to migrate to another cloud provider in the future if needed.

Cloud-Independent Solution with Docker and Kubernetes

Pros

Portability: Using Docker containers and Kubernetes can make your application more cloud-agnostic, allowing you to run it on different cloud providers or even on-premises infrastructure. This approach reduces vendor lock-in.

Customization: Kubernetes offers a high degree of flexibility, allowing you to customize your deployment to suit your exact needs. This can be beneficial if your project has unique requirements.

Cost Control: While AWS provides managed services, Kubernetes gives you more control over resource allocation, which can help optimize costs.

Cons

Learning Curve: Docker and Kubernetes have a steeper learning curve compared to AWS managed services. It might take more time and effort to set up and maintain the infrastructure.

Operational Overhead: Managing Kubernetes clusters and Docker containers can introduce additional operational complexity, which may require more effort and expertise.

Conclusion

In this specific use case, our top priority is to expedite the deployment of our IoT solution. Therefore, we have decided to opt for the AWS native solution, despite potential higher costs. This choice will significantly reduce development time and effectively handle many aspects of scaling and infrastructure management, streamlining our overall process.

Design Overview

Our data pipeline will consist of the following components:

1. Data Ingestion
2. Data Streaming
3. Data Pre-Processing
4. Data Storage
5. Data Visualization

Other Important Components:

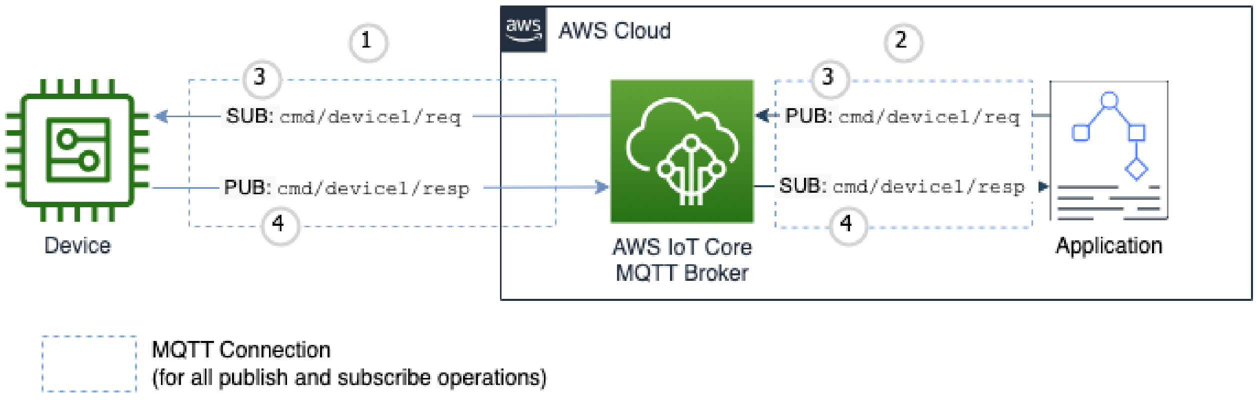
1. Device Registration
2. Device Management
3. Device Testing

4. User Management
5. User Dashboard

Data Ingestion

AWS IoT Core

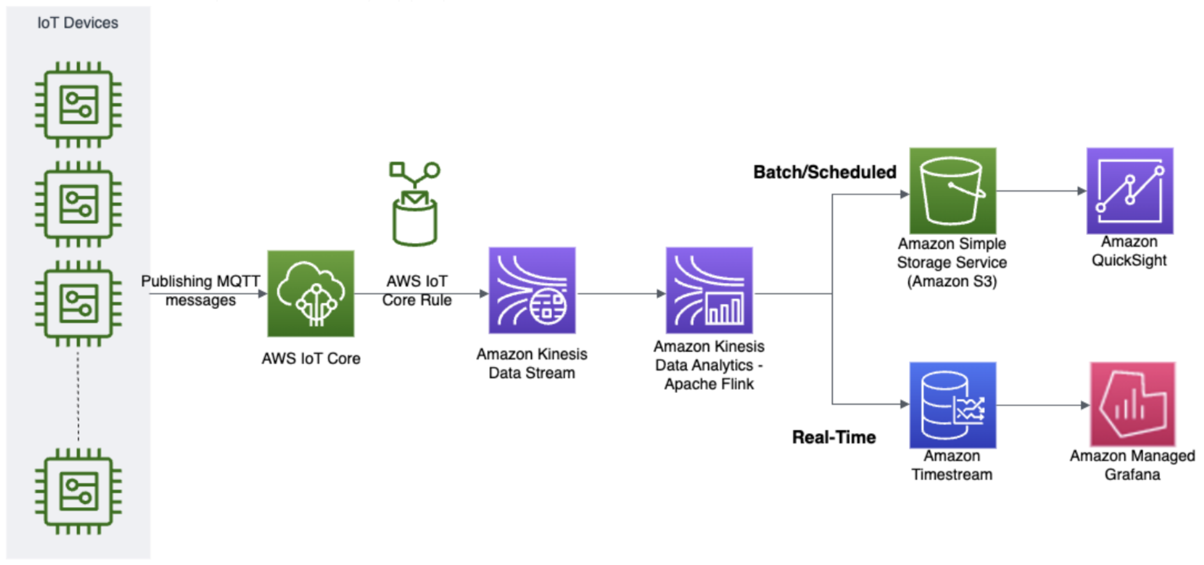
- Lets you connect billions of IoT devices and route trillions of messages to AWS services without managing infrastructure
- Provides a fully managed palette of MQTT-based messaging features
- Note: MQTT (Message Queuing Telemetry Transport) - extremely simple and lightweight messaging protocol (pub-sub) designed for limited devices and networks with high latency, low bandwidth or unreliable networks



Ingesting data from devices using AWS IoT Core and/or Amazon Kinesis

- We prefer IoT Core as the initial point of contact due to ability to support MQTT protocol, permanent authentication with certificates, fine-grained device control, bi-directional communication, and
- We also know we have smaller payloads per device so IoT will be fine
- However, we can publish data to Amazon Kinesis after initial ingestion to perform high-frequency data processing

Use Case: “Imagine that you have thousands of devices spread across a field. Every device reports its operational metrics and generates a small amount of data. To gain an overall view of operational status, drive anomaly detection, perform predictive maintenance, or analyze historical data, you need to control all devices and aggregate all data to get real-time or batch insights. AWS IoT Core provides the communication, management, authorization, and authentication of the devices and Kinesis Data Streams provides ingestion of high-frequency data. **You start by publishing data to AWS IoT Core, which integrates with Amazon Kinesis, allowing you to collect, process, and analyze large bandwidths of data in real time.**”



Note: Although we cannot use Kinesis to initially ingest our data, we can use it later in order to process our data in batches. See below.

Data Streaming

Purpose: The main reason we want to use an intermediate streaming service from IoT Core to S3 is so that we can batch together messages from the devices. We can then direct these batched messages to S3, Kinesis Data Analytics, Lambda, etc.It also decouples the architecture so we can achieve more scalability and flexibility for our system.

AWS Kinesis Data Streams vs Kinesis Data Firehose
Kinesis data streams

- Highly customizable and best suited for developers building custom applications or streaming data for specialized needs.
- Need to configure shards, scale, and write your own custom applications for both producers and consumers.

Kinesis Data Firehose

- Handles loading data streams directly into AWS products for processing
- Fully managed, scaling is handled automatically

Conclusion: Use Kinesis Firehose → Less to manage, only need to process data in AWS

Data Pre-Processing

Purpose: We can use some sort of compute engine in order to enrich and transform our data before storing or sending to the next AWS service.

Joining and Enriching Streaming Data on Amazon Kinesis

1. **Enrich with Static Data**

- The best method seems to be Amazon Kinesis Analytics coupled with S3. We store our static data in S3 which Kinesis Analytics can pull from and attach to our data payload in real time.

2. **Enrich with Dynamic Data**

- The best method seems to be Lambda coupled with DynamoDB. If we have user data that may be randomly changing, then we can use Lambda to process data batches, pull records from DynamoDb, and attach to the incoming data. DynamoDb is a good option because it can support random reads and writes efficiently (faster than S3).

In both cases, the enriched data is then passed onto another data stream, where it will continue down the pipeline.

* In this step, we may be able to perform new calculations that do not come from a reference database, but rather are generated from the existing attributes from the sensor data.

Side Note: IoT Analytics may seem like another good option as it provides some pre-processing features with pipelines but the use cases are tailored more to analyze data over a longer period of time. We need to display real-time data analytics to the user, at least for now. See the FAQ section on IoT Analytics vs. Amazon Kinesis Analytics here.

Data Storage

Because we are dealing with such a high-volume of data, any RDBMS will be infeasible. There we can use a NOSQL data store such as S3 to store raw and processed data. This data can later be referenced by analytics engines such as Amazon Quicksight to do offline analysis.

For our real-time analytics data, the most obvious choice seems to be Amazon Timestream. It automatically optimizes the storage based on the most recent events, while other options such as S3 and DynamoDb would need to be manually configured. See more details below.

Amazon Timestream - fast, scalable, and serverless time-series database service that makes it easier to store and analyze trillions of events per day up to 1,000 times faster

Pricing details

- 1 million write of 1KB size = $0.50, writes to S3 object storage
- Per 1 GB scanned = $0.01
- Memory Storage: Price per GB stored per HOUR = $0.036
- Disk Storage: Price per GB stored per MONTH = $0.03

Workflow: Write data to S3 → Store recent data in memory → Push older data to disk

Benefits:

- One tenth the cost of relational databases
- Only pay for the data you ingest, store, and query
- Keeps most recent data in memory and stores older data in cost-optimized storage service based of custom policies
- Can access data easily without specifying the exact location within the database
- Built-in analytics functions
- Serverless - no need to manage and provision servers
- Can process trillions of events and millions of queries per day
- Partitions data by time - faster query response on time series data

Note: We can create multiple tables within one database instance.
Each table should represent one natural gas site.

Resources:
Timestream AWS Docs
Setting up Timestream in CDK

Data Visualization

1. Grafana
    a. This solution publishes data from Iot Core to Amazon TimeStream
    b. Grafana will connect to Timestream table and generate graphs in real time
    c. Can specify different time ranges (i.e. Last 5 min, Last hour, Last 6 hours, etc)

2. Amazon Managed Service for Apache Flink
    a. With this solution, we can directly act on the data that is passing through the stream using Java Code
    b. We can then create an analytics application and publish it to users

3. CloudWatch
    a. This solution creates a Lambda to ingest a batch of incoming messages from the data pipeline (can be Kinesis Firehose, IoT analytics, etc).
    b. This Lambda will verify the message structure and write the data to Cloudwatch. We can now use the dashboarding features of Amazon CloudWatch to plot our custom metrics on graphs
    c. These cloudwatch dashboards can be shared to users as detailed here

Conclusion: I prefer Option 1 due to the fact that Timestream was developed for this exact use case. It also easily connects with Grafana, a robust visualization service that can be made public to clients. Option 2 introduces unnecessary complexity with Apache Flink and acts directly on the data stream before it is even stored in the database. We could technically store it first, and then use the Apache Flink analytics, but at this point we might as well use Grafana for its simplicity. Option 3 could work but the Cloudwatch visualization tools are not as powerful.

Side Note: Amazon QuickSight may seem like a good solution.

- Can generate datasets and quickly create graphs based on the available parameters.
- We can publish these graphs to be viewable by clients and manager the user authentication with Amazon Cognito

However, Quicksight only has access to aggregated datasets over a certain time period, NOT real-time data. Datasets can only be generated at maximum on an hourly basis.

Device Registration

Follow this guide to provision new devices.

When provisioning a new device the following resources are creates:
- IoT thing - representation of the device in the cloud
- X.509 certificate - allows device to perform mutual authentication with AWS IoT
- IoT policy - define the operations that a device can perform (i.e. publish, subscribe, etc)

Device Management

AWS IoT Device Management
- Helps you register, organize, monitor, and remotely manage IoT devices at scale
- Concept: create a representation of your physical device in the cloud, called a “thing”
- Developer Documentation

Device Testing

Iot Device Simulator
- Allows customers to create and simulate hundreds of connected devices, without having to configure and manage physical devices, or develop time-consuming scripts.
- helps customers test device integration and improve performance of their IoT backend services

Pricing Details
- Cost for 100 device simulations per month for 6 hours per day = $3.05/month

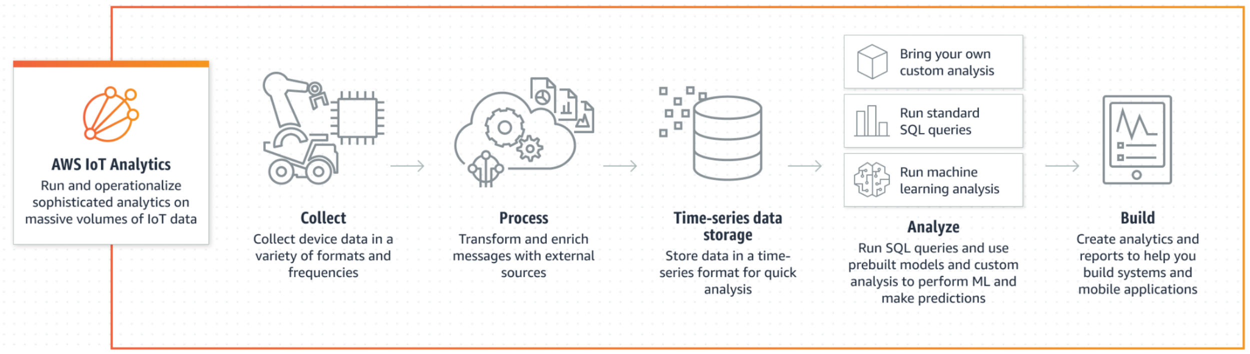
Other Notes

AWS IoT Analytics

Overview: Allows you to ingest, process, store and analyze/visualize your data.

AWS IoT Analytics

- Fully managed, operationalized analytics for your IoT devices



AWS IoT Analytics
Run and operationalize sophisticated analytics on massive volumes of IoT data
Collect
Collect device data in a variety of formats and frequencies
Process
Transform and enrich messages with external sources
Time-series data storage
Store data in a time-series format for quick analysis
Bring your own custom analysis
Run standard SQL queries
Run machine learning analysis
Analyze
Run SQL queries and use prebuilt models and custom analysis to perform ML and make predictions
Build
Create analytics and reports to help you build systems and mobile applications

Collect

- Can ingest data from IoT topic, specify this info in Analytics dashboard
- Can create a rule in IoT Core dashboard, automatically pushed data to Analytics channel
- Allows us to collect data from multiple sources including IoT Core, S3, and Kinesis

Pipelines

- Filter, transform, and enrich messages
- Custom-preprocessing - can specify lambdas to add vital context (i.e. geolocation)
- Batch messages prior to enrichment to ensure scalability
- Can easily replicate pipelines as fleet of devices grows

Data Storage - See more information on Amazon Timestream

- Stores both processed and raw data

Creating Datasets

- Can query the Data Store using standard SQL
- Queries an run ad hoc or scheduled (hourly, daily, weekly, etc)
- Available via API, console, and Jupyter Notebooks
- Can easily connect to QuickSight for visualizations

Main Drawback: Can not do real time visualizations with IoT Analytics

- However, we still may be able to do useful analyses in future features
- Use cases: Predictive maintenance, Process efficiency scoring, Training ML models

Other References

- AWS Solutions: Real-Time IoT Device Monitoring with Kinesis Data Analytics
- Visualizing Data in Amazon Timestream using Grafana