Project Title: Wildlife Health Surveillance Cloud Solution for Wildlife Analysis and Technology Center for Health (WATCH)

Contents

Project Description

The Wildlife Health Surveillance (**WHS**) is a fictional Program dedicated to monitoring and detecting pathogens affecting wildlife populations, which can potentially spill over to humans, posing a significant public health risk. To facilitate this work, the WHS Program requires a robust and scalable IT solution for the genomic analysis of wildlife samples, storage of genomic sequence data, and sharing of results with various stakeholders, including wildlife health organizations, research institutions, and public health agencies.

An integral part of this initiative is the *Human Health Opt-In Program*, where residents near wildlife habitats can voluntarily share health data during checkups or blood donations. This sensitive data, securely stored and anonymized, will be analyzed alongside wildlife genomic data using AI to identify potential health risks.

The aim of the WHS Cloud Solution Project is to design and implement an IT solution using Amazon Web Services (AWS) for efficient genomic analysis, secure storage, and seamless data sharing. This project will enable real-time data sharing across different organizations, enhance collaborative efforts in monitoring wildlife health, and support timely decision-making based on genomic insights.

Use Cases:

1. **Rabies Virus Surveillance:** Monitoring and analysis of rabies virus samples collected from wildlife, providing data to support control measures and prevent transmission to humans.
2. **Chronic Wasting Disease (CWD) Monitoring:** Tracking the spread and evolution of CWD among deer populations to inform management strategies and reduce risks to wildlife and human health.

Key IT System Components:

- **Bidirectional Data Sharing with Access Controls:** Users can securely upload and download sequence files and metadata within a shared file directory, with role-based access control to ensure data privacy.
- **Data Storage:** Efficient and scalable storage for large genomic sequence files, with different tiers for rapid access and long-term archival.
- **Data Analysis:** Provision of Linux-based environments for executing bioinformatics workflows, requiring high computational resources for batch processing of genomic data.
- **Data Visualization:** Tools for displaying genomic results, including dashboards and phylogenetic trees, accessible to authenticated users.
- **System Administration Interface:** A user-friendly interface for managing user accounts, configuring access permissions, and monitoring system status.

Roles Required:

- **Cloud Architect and Lead:** Responsible for designing the cloud architecture and leading the implementation.
- **Data Management and Storage:** Focuses on data storage solutions, ensuring security and compliance, and optimizing data flow.

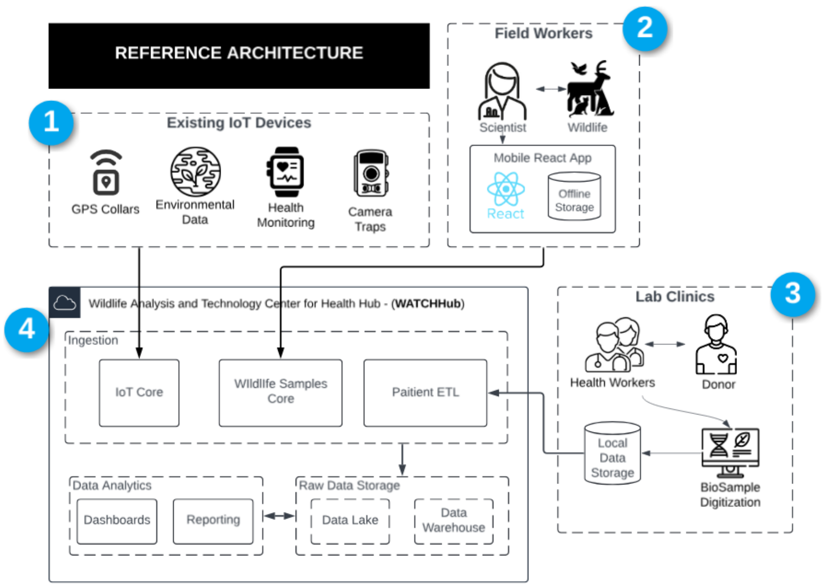
System Overview

This document outlines a full-stack cloud-native data ingestion and analytics pipeline for wildlife health monitoring, built on AWS using infrastructure-as-code. It demonstrates best practices in IoT provisioning, secure data ingestion, scalable ETL, and metadata-driven analytics, all orchestrated with CDK and visualized via Metabase.

Reference Architecture and Scenario Overview

Commented [CS1]:
- **"Reference Architecture Mapping"**
- **"Scenario Flow"**

Add callout (box or icon) to highlight cross-cutting concerns like security/monitoring.



This reference architecture presents a real-world scenario where data related to wildlife health is currently being collected through various existing channels.

It's a crucial document when starting a project because it address 4 main purposes:

1. **Establishes a common understanding** among all audiences and stakeholders to facilitate feasibility of the system and ensures to capture.
2. **Identifies key integration points** between devices, users, and backend systems.
3. **Clarifies scope and boundaries**, reducing ambiguity and misalignment early in the project.
4. **Supports planning and decision-making** by illustrating dependencies, technologies, and data movement.
5. **Accelerates solution development** by reusing proven patterns and best practices.

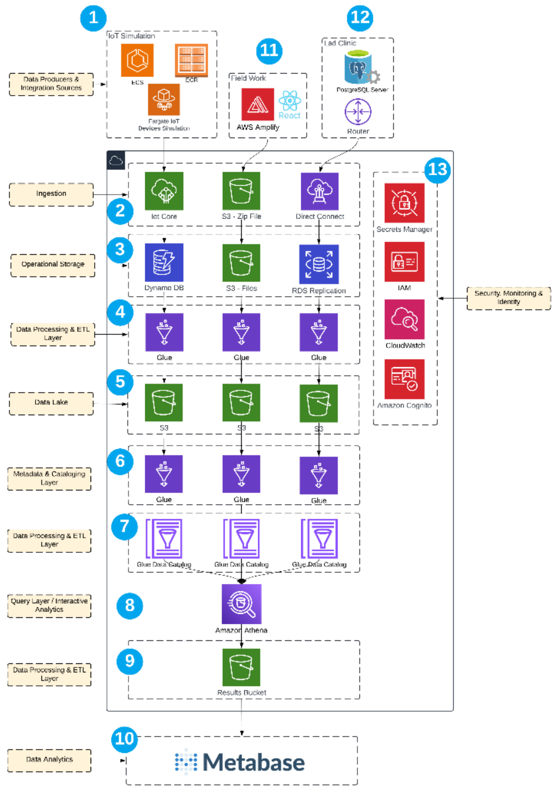
The purpose of this project is to design and implement the WATCHHub—a centralized, cloud-native ingestion and processing platform that brings these disparate data streams together into one scalable, secure system. The 4 main components are:

1. **Existing IoT Devices**
    a. IoT sensors such as GPS collars, environmental monitors, and camera traps are already deployed and maintained by third-party contractors. These devices transmit telemetry data via satellite to cloud endpoints.
2. **Field Scientists & Mobile Data Collection**
    a. Field workers are actively collecting wildlife samples in remote regions. Using a mobile React application with offline storage, they digitize and submit observational and sample metadata to cloud storage when connectivity is restored.

3. **WATCHHub (Project Focus)**
    a. The core objective of this project is to build the Wildlife Analysis and Technology Center for Health Hub (WATCHHub)—a centralized ingestion and coordination layer that connects all upstream data sources. WATCHHub will normalize data, enforce security and compliance, and provide shared access for stakeholders such as researchers, health agencies, and wildlife authorities. This architecture is designed to facilitate technical alignment across engineering, scientific, and executive teams.
4. **Lab Clinics & Human Health Data**
    a. Clinics near regions of concern are collecting opt-in health data from human donors. These clinics already operate with localized systems for biosample digitization and storage, but lack a unified integration point.

High Level Outline of Solution Architecture.

Commented [CS2]: **Suggestions:**
•Consider labeling the core architecture diagram as "Figure 1" and referencing it.
•Add a short paragraph describing how the solution layers interact: ingestion, processing, analytics.



Establishing a high-level solution architecture at the outset of the project is critical for aligning technical direction with business goals. It provides a clear, visual representation of system components and acts as a high-level blueprint that ensures all teams understand the technical scope of the project.

It also allows tech teams to align services to layers of the ingestion-to-analytics data architecture.

1. IoT Simulation

- **Reference Arch**: Simulated GPS collars, environmental sensors, and other wildlife monitoring devices.
- **AWS Arch**: ECS, ECR, and Fargate simulate continuous IoT data generation across a device fleet.

2. Field Work / Mobile App

- **Reference Arch**: Field workers record wildlife observations and health data through a mobile interface.
- **AWS Arch**: AWS Amplify + React enable an offline-capable web app with secure deployment and sync features.

3. Lab Clinics & BioSample Integration

- **Reference Arch**: Data captured in clinical settings or research facilities, synced periodically.
- **AWS Arch**: Local PostgreSQL servers feed into AWS via Router and Direct Connect, supporting hybrid ingestion.

4. Ingestion Layer

- **Reference Arch**: Entry point for structured and unstructured data streams from IoT, clinics, and fieldwork.
- **AWS Arch**: IoT Core, S3 (for zipped file ingestion), and Direct Connect capture and route data into the platform.

5. Operational Storage

- **Reference Arch**: High-speed storage for hot, unprocessed, and structured data sources.
- **AWS Arch**: DynamoDB (fast, structured telemetry), S3 (raw files), and RDS Replication (for clinic databases).

6. Data Processing & ETL Layer

- **Reference Arch**: First transformation stage - cleansing, enrichment, deduplication, filtering.
- **AWS Arch**: AWS Glue handles distributed ETL jobs across storage layers.

7. Metadata & Cataloging Layer

- **Reference Arch**: Schema tracking, discovery, lineage tracking, and downstream query optimization.
- **AWS Arch**: AWS Glue Data Catalog defines metadata and enables Athena and Metabase to interact with the data lake.

8. Query Layer / Interactive Analytics

- **Reference Arch**: Enables on-demand querying of large, semi-structured datasets without preprocessing.
- **AWS Arch**: Amazon Athena executes SQL-like queries on top of S3 using Glue Data Catalog.

9. Results Bucket / Query Output Storage

- **Reference Arch**: Stores query results from Athena for downstream visualization or export.
- **AWS Arch**: S3 results bucket configured as the Athena query output destination.

10. Data Analytics & Visualization

- **Reference Arch**: Reports, dashboards, charts, and field insights consumed by researchers, analysts, and decision-makers.
- **AWS Arch**: Metabase connects to Athena, enabling interactive dashboards and cross-source visualization.

11. External Data Source – Field Work

- **Reference Arch**: Amplify-based apps used in field environments to capture direct observations and sync later.
- **AWS Arch**: Amplify/React app uploads data to S3 or API Gateway → Lambda pipeline when online.

12. External Data Source – Lab Integration

- **Reference Arch**: Sample metadata and test results are digitized and uploaded from laboratory tools.
- **AWS Arch**: PostgreSQL server uploads data via router → Direct Connect → RDS replication.

13. Security, Monitoring & Identity (Cross-Cutting Layer)

- **Reference Arch**: System-wide enforcement of security, access control, and observability.
- **AWS Arch**: Secrets Manager (credentials), IAM (role-based access), CloudWatch (monitoring), and Amazon Cognito (authentication).

Cloud Development Kit Deployment

The **AWS Cloud Development Kit (CDK)** is an open-source software development framework for defining cloud infrastructure using familiar programming languages. The main driving reason for using CDK is due to high-useage in governments that desire on-prem solutions.

IoT Data Solution

The IoT solution for the Wildlife Health Surveillance (WHS) Program leverages AWS to efficiently collect, transmit, and analyze data from various IoT sensors deployed in the Great Bear Rainforest. IoT devices, including GPS collars, environmental sensors, health monitoring devices, and camera traps, gather real-time data on wildlife movement, environmental conditions, and health metrics. This data is transmitted via satellite to AWS IoT Core, where it is processed, stored, and made accessible for analysis.

The solution ensures scalable, secure, and seamless integration of data, enabling effective wildlife monitoring and timely decision-making.

IoT Devices

GPS Collars - Collects location data from wildlife, including latitude and longitude coordinates.

Environmental Sensors - Measures temperature, humidity, wind speed, and proximity to human populations

Health Monitors - body temperature, heart rate, and signs of infection.

Camera Traps - Records wildlife behavior and physical signs of illness through photographic images

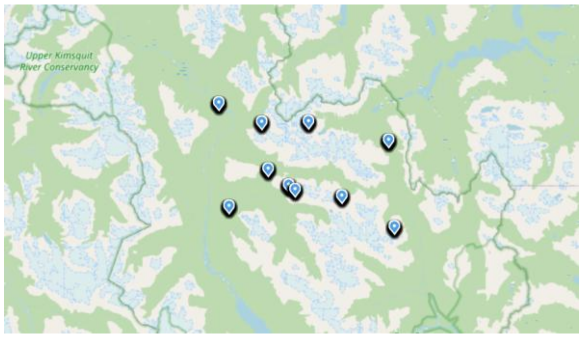
Data Definition Types

Data Definition Types

Data Type	Frequency of Upload	Data Fields
GPS Collars	Near Real-Time	Latitude, Longitude
Environmental Sensors	Hourly	Temperature, Humidity, Wind Speed, Proximity to Human Populations
Health Monitoring Devices	Every Minute	Body Temperature, Heart Rate, Signs of Infection
Camera Traps	Nightly at 11:59 PM	Image Data

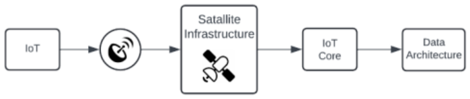
GPS Collar Analytics Result:

Environmental Sensors

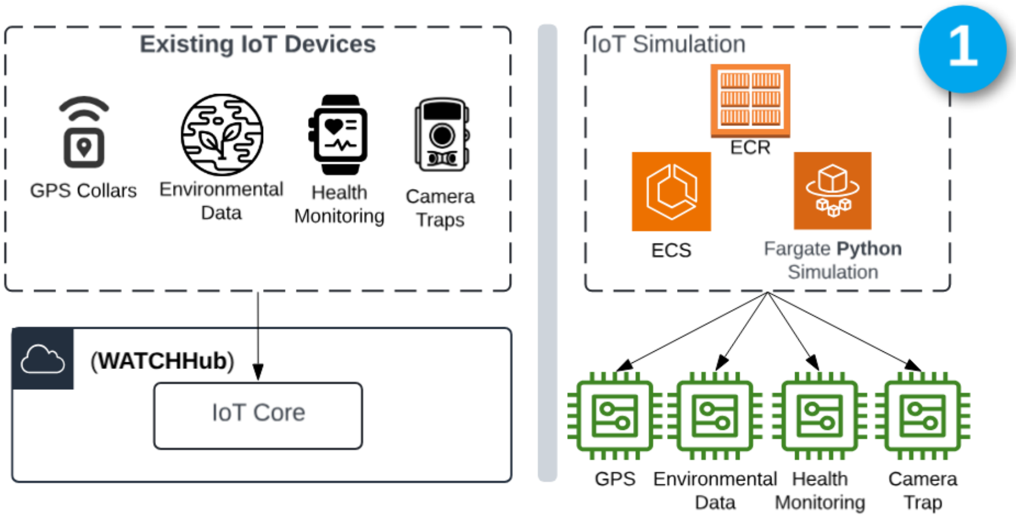


General Real-World Data Flow

Commented [CS3]: Option: show both *real-world* and *simulation* paths side-by-side.

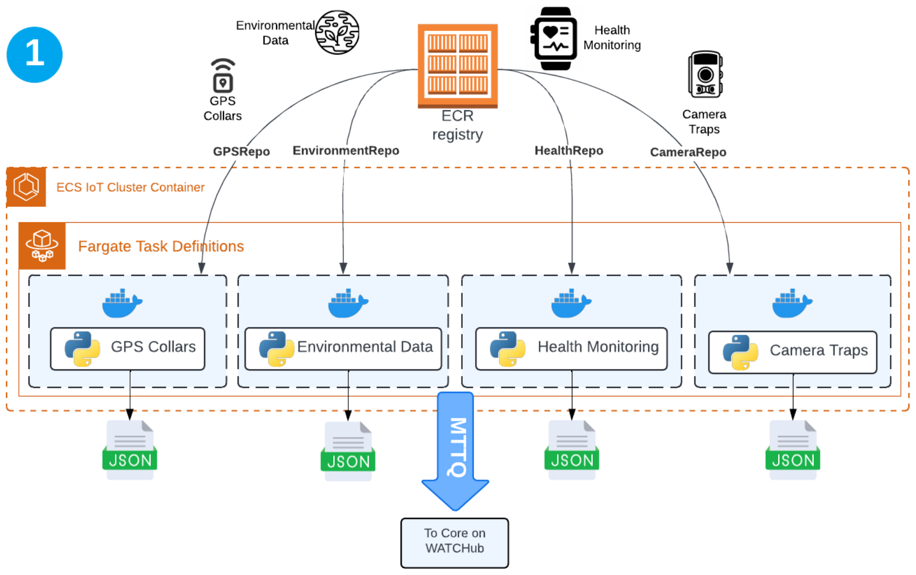


Mock Setup for Testbed



This diagram illustrates how the architecture decouples real-world IoT data from simulated input during development and testing.
The left image represents the analogous reference architecture, and the right shows the high-level services used to create this simulation, using containerized Python simulations running on ECS Fargate.

We will look in depth at (1) and the services used for the mock-sensor solution and how we configure them.

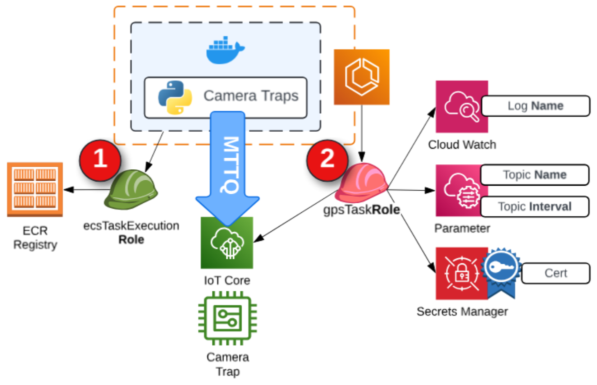


This diagram shows how simulated IoT data is generated and ingested into the WATCHHub system using AWS Fargate. Each device type—GPS collars, etc.—is containerized with **Python scripts** and stored in dedicated **ECR repositories**. **Fargate services** pull and run these containers and generate JSON payloads that are streamed via MQTT to the core WATCHHub ingestion layer.

This setup allows for scalable, automated testing and ingestion of wildlife health data.

Below, we detail the IAM structure that enforces least-privilege access from Fargate tasks to required AWS services.

IAM Structure Summary



Each Fargate task is provisioned with two roles:

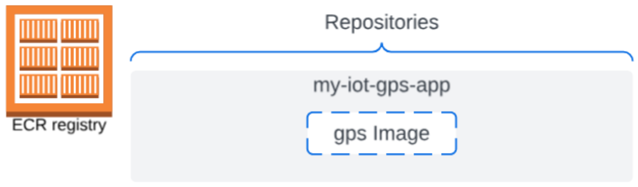
1. **Execution Role** (ecsTaskExecutionRole): Grants the ECS agent permission to pull container images from **Amazon ECR** and configure **CloudWatch log streams**.
2. **Task Role** (e.g., gpsTaskRole): Grants the Python application inside the container permission to:
   a. Retrieve TLS certificates from **Secrets Manager**.
   b. Publish telemetry to **AWS IoT Core** via MQTT.
   c. Send structured logs to **CloudWatch**.

This separation of responsibilities ensures **least-privilege access** and follows AWS best practices for containerized workloads.

Below we will discuss each service, why it was chosen and how it's configured at a high-level.

CDK ECR-Stack Breakdown

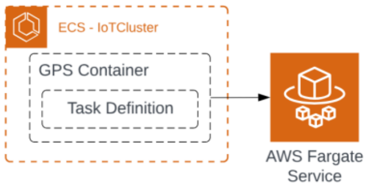
Amazon Elastic Container Registry (ECR)



Purpose: Stores the Docker container image that contains the Python simulation code.
Rationale: ECR allows secure and efficient management of the container image (MyIotGpsAppRepository) used by ECS to deploy the simulation. It's necessary to store, version, and retrieve the Docker image for the GPS simulation.
CfnOutput: Outputs the ECR repository URI, allowing other stacks (like ECS) to reference the image easily. This is crucial for enabling the automated workflow across the different AWS services.

CDK ECS-Stack

(ECS) with Fargate



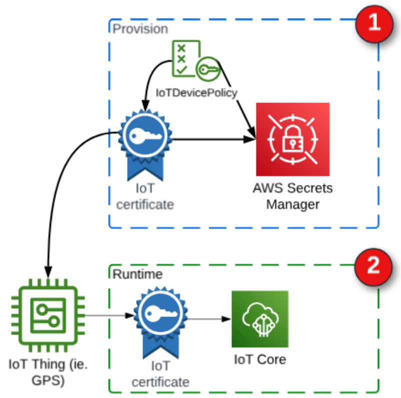
Purpose: Deploys and manages the containerized application for the IoT simulation.
Rationale: ECS orchestrates the deployment of the simulation in a Docker container. Using Fargate, a serverless compute engine, we offload infrastructure management to AWS, enabling the application to scale automatically based on demand without managing servers.
Task Definition: Defines how much CPU and memory are allocated for the simulation, along with the container image to be pulled from ECR. This is critical for ensuring the simulation runs with the correct resources.

Fargate Service: Maintains the desired number of running containers and ensures that the task is publicly accessible if needed. The service guarantees that the simulation is always running and handles scaling and networking for the container.

- Fargate is a **serverless** compute engine that allows you to run Docker containers without managing the underlying servers or infrastructure.
- With Fargate, you don't need to provision or manage EC2 instances. It automatically allocates the right amount of CPU, memory, and networking for your containers based on the **task definition** in ECS.
- Fargate pulls the image from ECR (via ECS) and runs it as a **container**, scaling and managing the infrastructure transparently.

From here, IoT Rules are setup to ingest data into the platform using the Data Ingestion Stack

CDK IoT-Stack



In order for the python scripts to be able to publish to the platform, certificates need to be provisioned

This stack configures the secure ingestion path from AWS IoT Core into the broader telemetry pipeline. It includes the creation of IoT rules that trigger Lambda-based ingestion, as well as reusable mechanisms (via factory functions) to provision authenticated device identities and credentials for secure MQTT communication.

1 Provisioning (Device Identity & Credential Lifecycle)

1. **IoT Thing**
   - Logical identity representing a simulated device (e.g., GPS collar).
2. **IoT Certificate**
   - Created using the createKeysAndCertificate API via custom resource Lambda.
   - Used to authenticate MQTT connections securely (TLS).
3. **IoT Device Policy**
   - Grants scoped permissions for iot:Connect, iot:Publish, iot:Subscribe.
4. **Principal Attachments**
   - The certificate is attached to both the Thing and the policy.
5. **Secrets Manager Entry**
   - Certificate, private key, and Thing metadata are stored securely for later retrieval by ECS tasks.

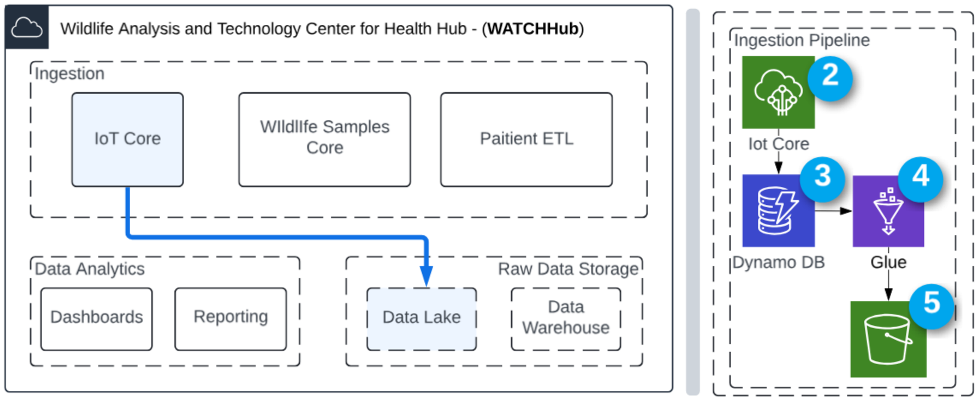
2 Runtime (Telemetry Ingestion Path)

These components are directly defined within IoTStack.ts and are used during device data publishing.

1. **IoT Topic Rule**
   - Listens to MQTT topics from simulated devices.
   - Routes matching payloads to a Lambda function.
2. **IoT Rule Action (Lambda Invocation)**
   - The rule triggers TopicProcessorLambda, passing incoming telemetry data.
3. **IAM Role for IoT Rule**
   - Grants permission for IoT to invoke the Lambda securely.
4. **Log Group (Optional)**
   - May be used for debugging or tracing IoT Rule invocations.

Next we will investigate the Data Ingestiong Stack (how data is extracted from Topics and sent to DynamoDB and the ETLd into the S3 bucket within the data lake.

Data Ingestion Stack



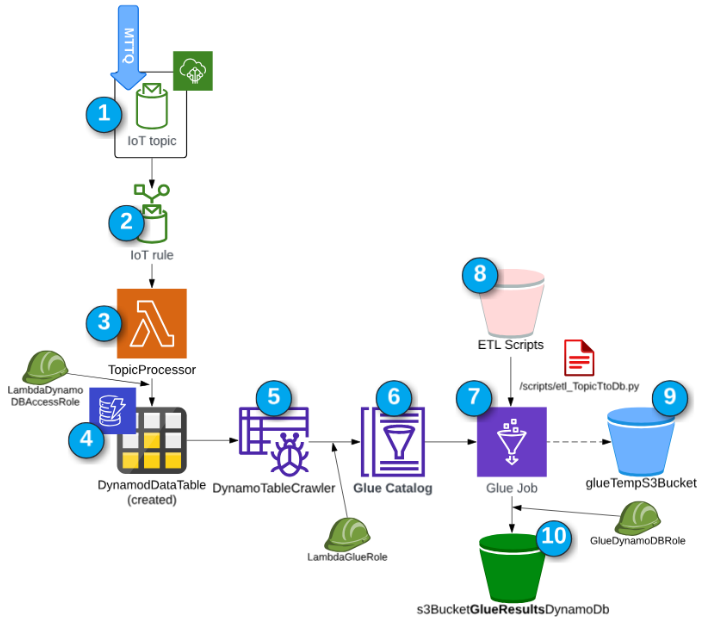
This **Data Ingestion Stack** illustrates the end-to-end AWS pipeline used to collect, process, and store wildlife health data in WATCHHub.

The flow begins with **(5) AWS IoT Core**, which ingests JSON payloads from simulated or real IoT devices via MQTT. These incoming messages are temporarily stored in **(6) DynamoDB**, enabling quick access for near-real-time applications or interim processing. **(6) AWS Glue** icon encapsulates crawlers, data catalogue and Glue ETL - cleaning, transforming, and structuring the data. Finally, the processed data is written to **Amazon S3**, where it can be stored in a data lake for further processing into the analytics pipeline or long-term archival.

This stack ensures scalable, secure, and automated ingestion from edge to storage—supporting downstream analytics and decision-making.

Below we will look at depth how this process is enacted and rationale behind each services.

Data Ingestion Stack in depth look



This architecture provides a solution for handling IoT data and ensuring that data flows smoothly from ingestion to storage. This solution is leveraged to store data to later be processed for the analytics pipeline.

1. **IoT Topic**: The entry point for data, where IoT devices publish their data. This represents the raw data coming from various sources.
2. **IoT Rule**: Acts as a filter and router for incoming IoT data, directing it to appropriate services for processing. It determines how data from the IoT topic is handled and where it should go next.
3. **TopicProcessor (Lambda Function)**: Processes the data received from the IoT rule. This serverless component can transform, enrich, or filter the data as necessary before storing it.
4. **GpsDataTable (DynamoDB)**: A NoSQL database table where the processed data is stored. This provides fast, flexible, and scalable storage.
5. **DynamoTableCrawler (Glue Crawler)**: Automatically scans the data in DynamoDB, inferring schema and adding metadata to the Glue Catalog. This facilitates managing the dataset's schema and using it in ETL processes.
6. **Glue Catalog**: A central repository where metadata related to various data stores and their schemas are stored. It serves as a reference point for data analysis tools and ETL jobs.
7. **Glue Job**: Utilizes the defined ETL scripts to perform further transformations and data processing tasks. This step is critical for preparing data for analytics or additional processing.
8. **ETL Scripts**: Stored scripts that define the operations performed by the Glue job. Typically, these scripts are written in PySpark or Scala and dictate how data transformation should be performed.
9. **Glue Temp S3 Bucket**: Used by the Glue Job for temporary data storage during processing. It holds intermediate data and supports efficient ETL operations.

10. **S3 Bucket (GlueResultsDynamoDb)**: The final destination for the processed data, where it is stored in a format optimized for querying and analysis, such as Parquet or ORC. This data can then be used for further analysis, reporting, or as input for other applications.

IAM Structure Summary

Ingestion Lambda Role (LambdaDynamoDBAccessRole):
Grants the Lambda function (triggered by an IoT Rule) permission to:

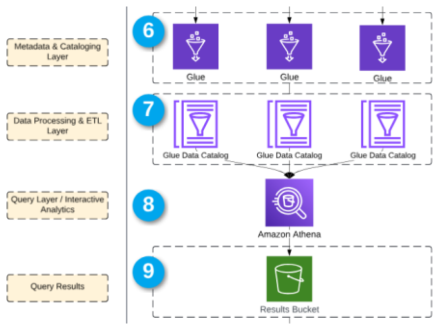
- Insert decoded MQTT message payloads into a DynamoDB table.
- Emit operational logs to CloudWatch.

ETL Orchestration Role (LambdaGlueRole):
Grants the ETL Lambda functions permission to:

- Start AWS Glue Crawlers and Jobs programmatically.
- Read script assets from Amazon S3.
- Log status and execution metrics to CloudWatch.

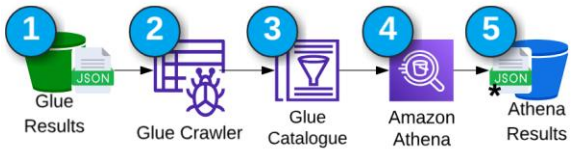
Glue Job Execution Role (GlueDynamoDBRole):
Grants the Glue Job runtime permission to:
Scan the DynamoDB source table.
Write transformed data to the designated S3 output bucket.
Access temporary storage in S3 during job execution.

Data Analytics Stack



Query Enablement Workflow (Preparing for Metabase)

This portion reflects the downstream metadata and analytics workflow already outlined in the high-level architecture. AWS Glue Crawlers register schema metadata in the Glue Data Catalog, enabling serverless querying via Amazon Athena. Query results are stored in S3 for downstream access by tools like Metabase.



1. **Glue Results (S3 Bucket)**
   Final ETL outputs from AWS Glue Jobs are written as JSON files into a designated S3 bucket.
2. **Glue Crawler**
   Automatically scans the contents and infers the schema, updating the Glue Data Catalog with table definitions.
3. **Glue Catalogue**
   Serves as the metadata registry, allowing Amazon Athena to interpret raw files as queryable datasets.
4. **Amazon Athena**
   Executes SQL queries on the data using schema information from the Glue Catalogue. This enables interactive and ad hoc analysis without additional transformation.
5. **Athena Results (S3 Bucket)**
   Stores the output of Athena queries. This data is then visualized via Metabase, completing the analytics pipeline.

This end-to-end flow ensures that clean, queryable, and up-to-date data is readily available for dashboards and field insights.