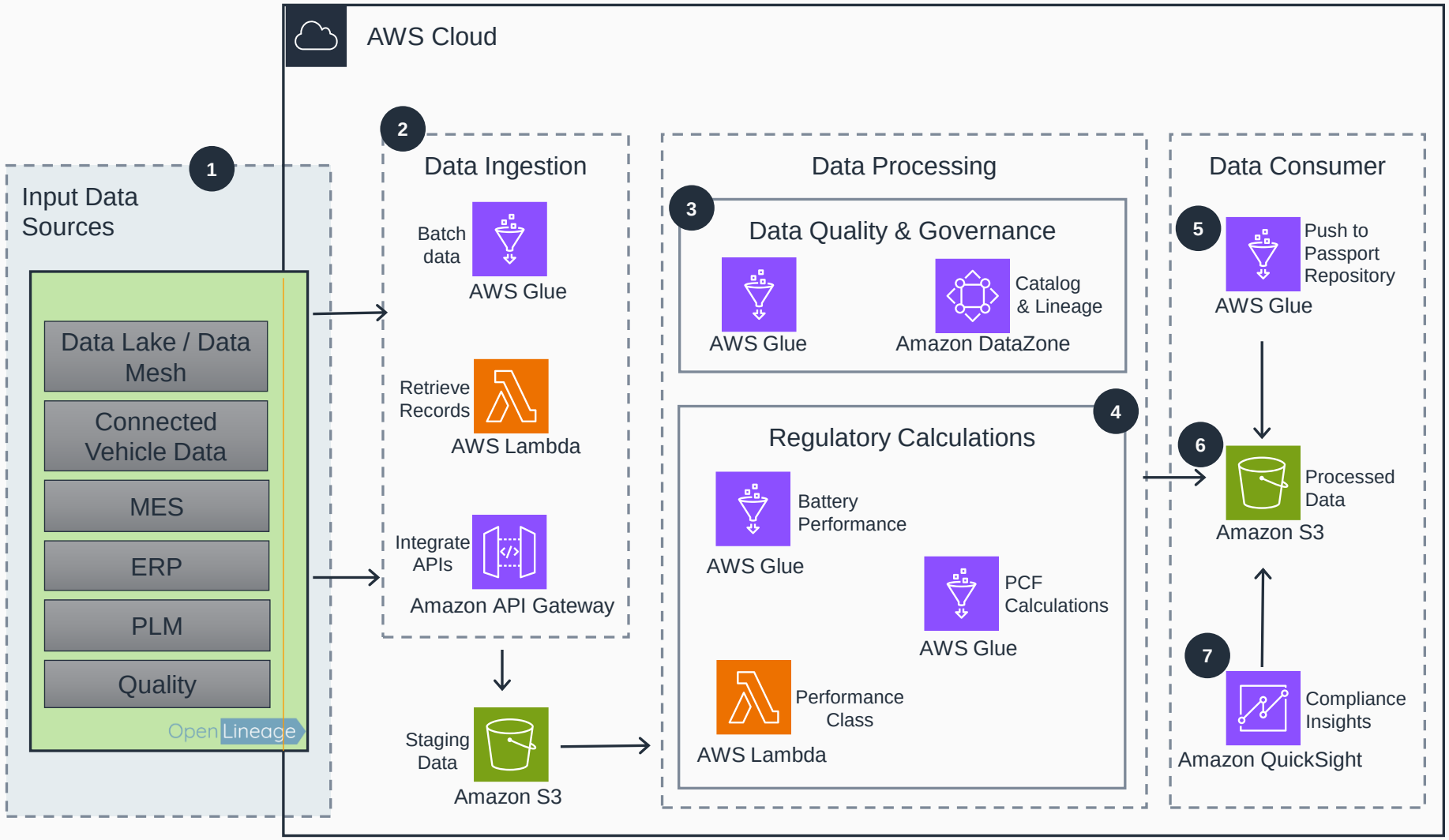
Guidance for Battery Passport on AWS

Data Pipeline

This architecture diagram illustrates how to construct a data pipeline that captures all the necessary data required to create a Battery Passport. It demonstrates the key components and their interactions, offering a comprehensive overview of the architecture's structure and functionality.



1. Battery Passport input data is stored in various systems across the enterprise, both within and outside of AWS. These systems contain raw data that needs to be first captured and transformed into the values required to showcase in a passport.

2. **AWS Glue** is used to fetch batch data, **AWS Lambda** is used to retrieve records from data on AWS Cloud, and **API Gateway** access on-prem APIs and ingest data into **Amazon Simple Storage Service (Amazon S3)** buckets for staging.

3. **AWS Glue** jobs validates the staging data and correlate the data coming from multiple sources. **Amazon DataZone** will contain the data lineage from source to processed data, while OpenLineage is used for data sourced from systems outside of AWS or not supported by default DataZone

4. The Regulation (EU) 2023/1542 has very specific requirements about the data unit (e.g., PCF should be per Kwh for four different lifecycle stages). **AWS Glue** jobs is used to batch calculations, **AWS Lambda** is used for complex and single-record calculations (e.g., Performance class), and **Amazon S3** will store all the processed data.

5. Passport data needs to be accessed through a QR code and has different access requirements. **AWS Glue job** will push the data into the passport data repository and maintain the transparency and authenticity of the data.

6. **Amazon Simple Storage Service (Amazon S3).** will store all the processed data. This data accessed for other regulatory requirements (e.g.; CSRD)

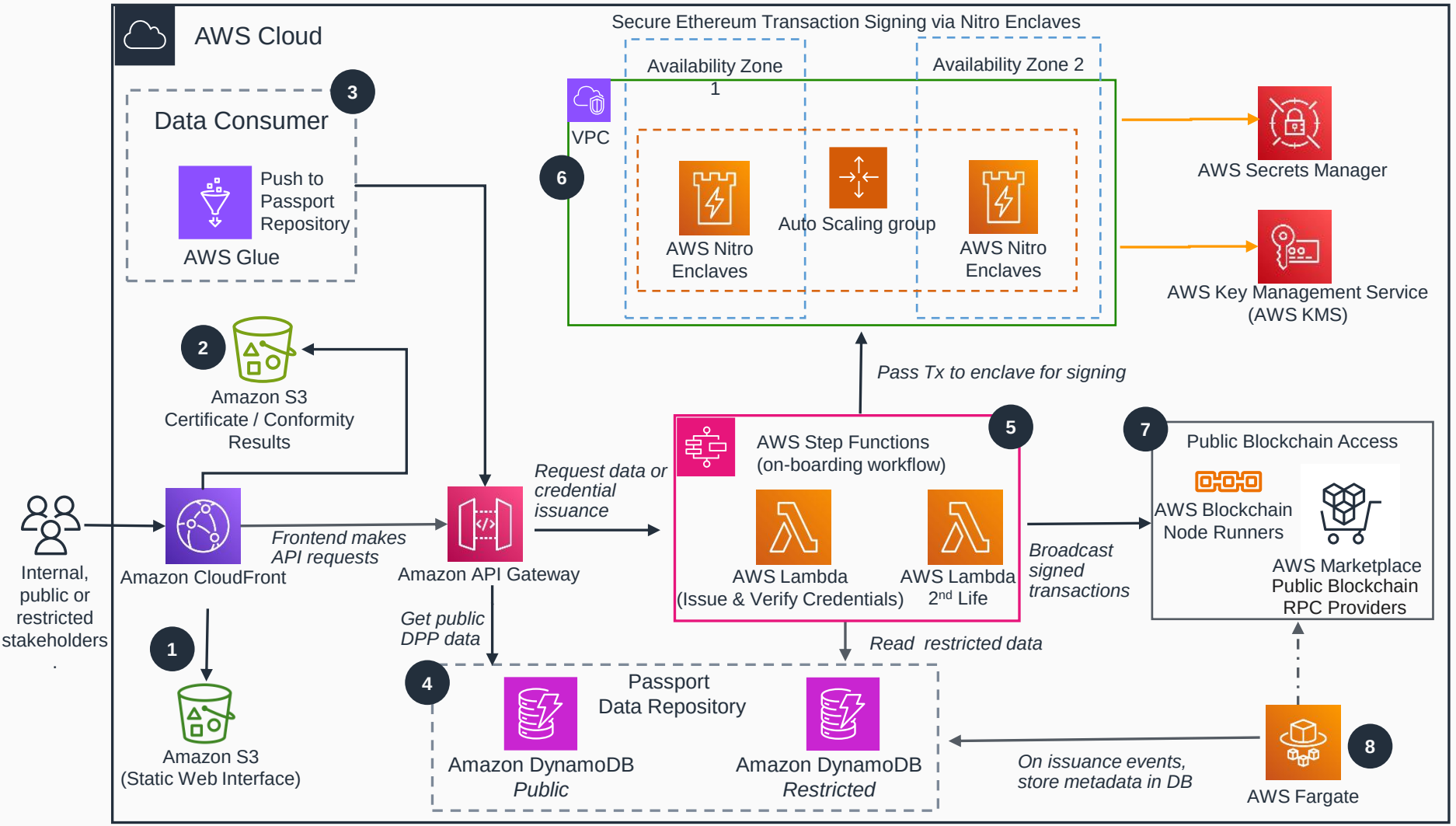
7. Regulation has certain milestones. **Amazon QuickSight** is used to analyze how the economic operator is performing against the milestones, e.g., Target material Rate of recoverability

AWS Reference Architecture

Guidance for Battery Passport on AWS

Battery Passport

This architecture diagram illustrates how to build Battery Passport on AWS. It demonstrates the key components and their interactions, offering a comprehensive overview of the architecture's structure and functionality.



1. Passport data accessed through a QR code, and access requirements may vary depending on whether the user is internal, public or restricted stakeholders. Users access the data via a web interface hosted in **Amazon S3** and accessed through **Amazon CloudFront**.

2. **Amazon S3** store various documents related to the Battery Passport, such as conformity results, certificates, etc. These documents get access through the web portal using a signed URL,

3. Data Pipeline fetch data from various internal and external sources and and make it available for Data Consumers. **AWS Glue** push the data into the Passport data repository and maintain its transparency and authenticity by inserting a hash into the blockchain node.

4. **AWS DynamoDB** stores passport data in different tables for public and restricted data.

5. **AWS Step Functions** workflows manage the passport data update and 2nd life processes. **AWS Lambda** functions issue and verify credentials to access restricted passport data. When a battery is repurposed, re-used, or remanufactured, The data shared with the second-life operator.

6. The data integrity of the Battery Passport is verified via a public blockchain. Ethereum nodes store the hash of the Battery Passport data. **AWS Nitro Enclaves** secure the signing of Ethereum transactions. The private key is stored in **AWS Secrets Manager**, and **AWS KMS is** used for signing

7. **AWS Lambda** functions in the Step Functions workflow write a subset of battery data to one or more public blockchains using self-managed, or partner-managed blockchain nodes.

8. **AWS Fargate** is used to run containerized event listener applications that subscribe to on chain events and trigger downstream actions.

Reviewed for technical accuracy March 16, 2026

AWS Reference Architecture