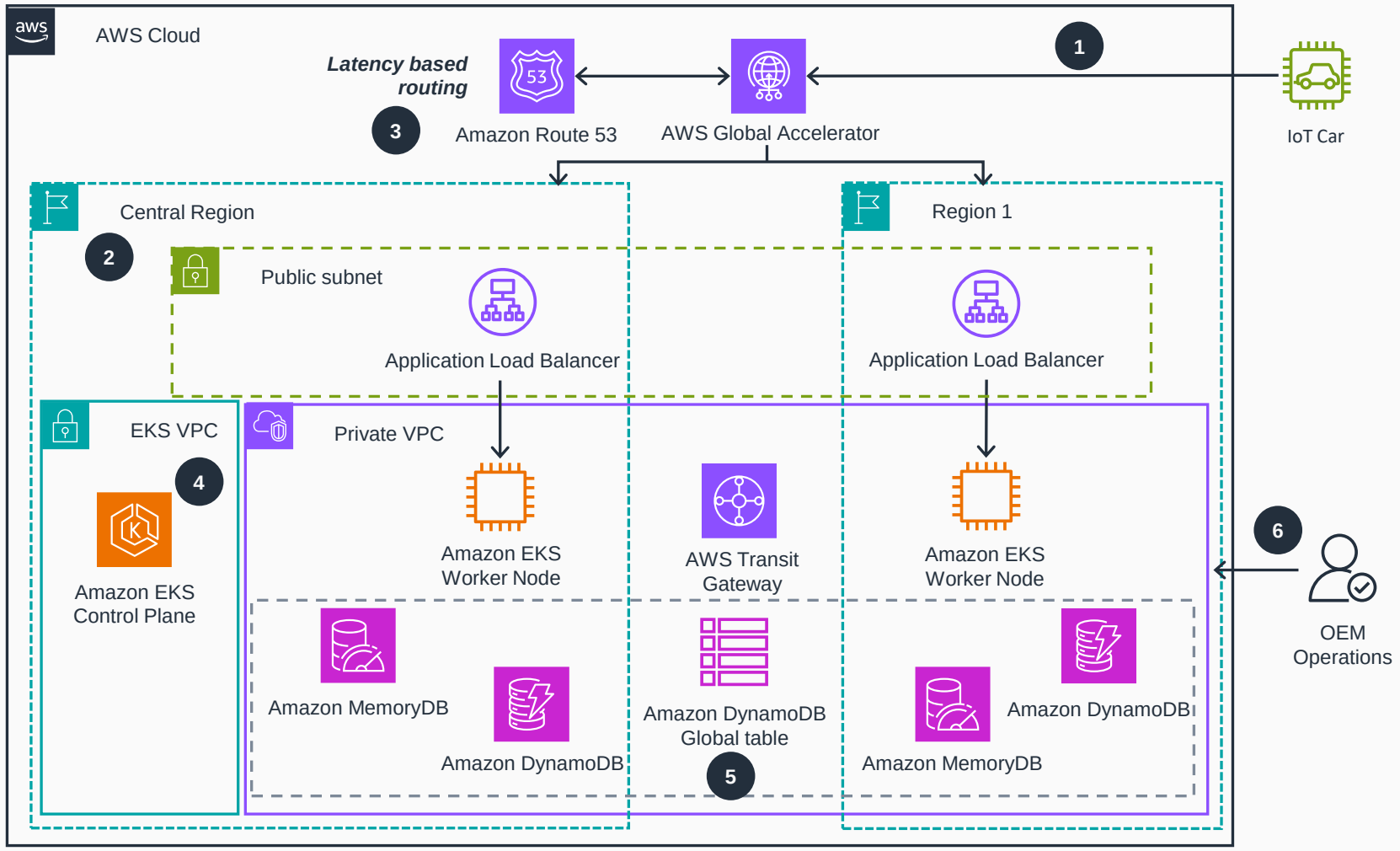
Guidance for Connected Mobility on AWS

Connected Vehicle Discovery Service

This architecture diagram shows how to build a multi-region, global discovery service to provide localization and configuration to each segment of the vehicle lifecycle.



1. Throughout a vehicle's lifecycle, it will periodically check to see if a new configuration exists by calling an **AWS Global Accelerator** endpoint to receive the latest vehicle configuration. To implement a multi-region architecture for Discovery APIs, customers can use **Global Accelerator** instead of building this solution using Amazon CloudFront based on **Amazon Route 53.**

2. The criticality of the service demands a multi-region approach to ensure high availability and low latency across the global footprint of the OEM. This architecture could span multiple regions across the globe where the OEM operates.

3. OEMs use **Amazon Route53** to host their own domain, and latency-based routing will handle routing the request to healthy **Application Load Balancer** instances in the lowest latency respective region.

4. For High Availability worker nodes to process requests, we recommend **Amazon Elastic Kubernetes Service (Amazon EKS)** setting up clusters that span both Availability Zones and AWS Regions.

5. For stateful applications, there are multiple ways to implement cross-region replication. For a fully-managed, multi-region, multi-active database and fast local read and write performance, implement **Amazon DynamoDB global tables.**

6. Each step of a vehicle's lifecycle requires different software configurations. The automaker uses internal software to update the production status of the vehicle, which modifies the configuration of the vehicle for that stage.

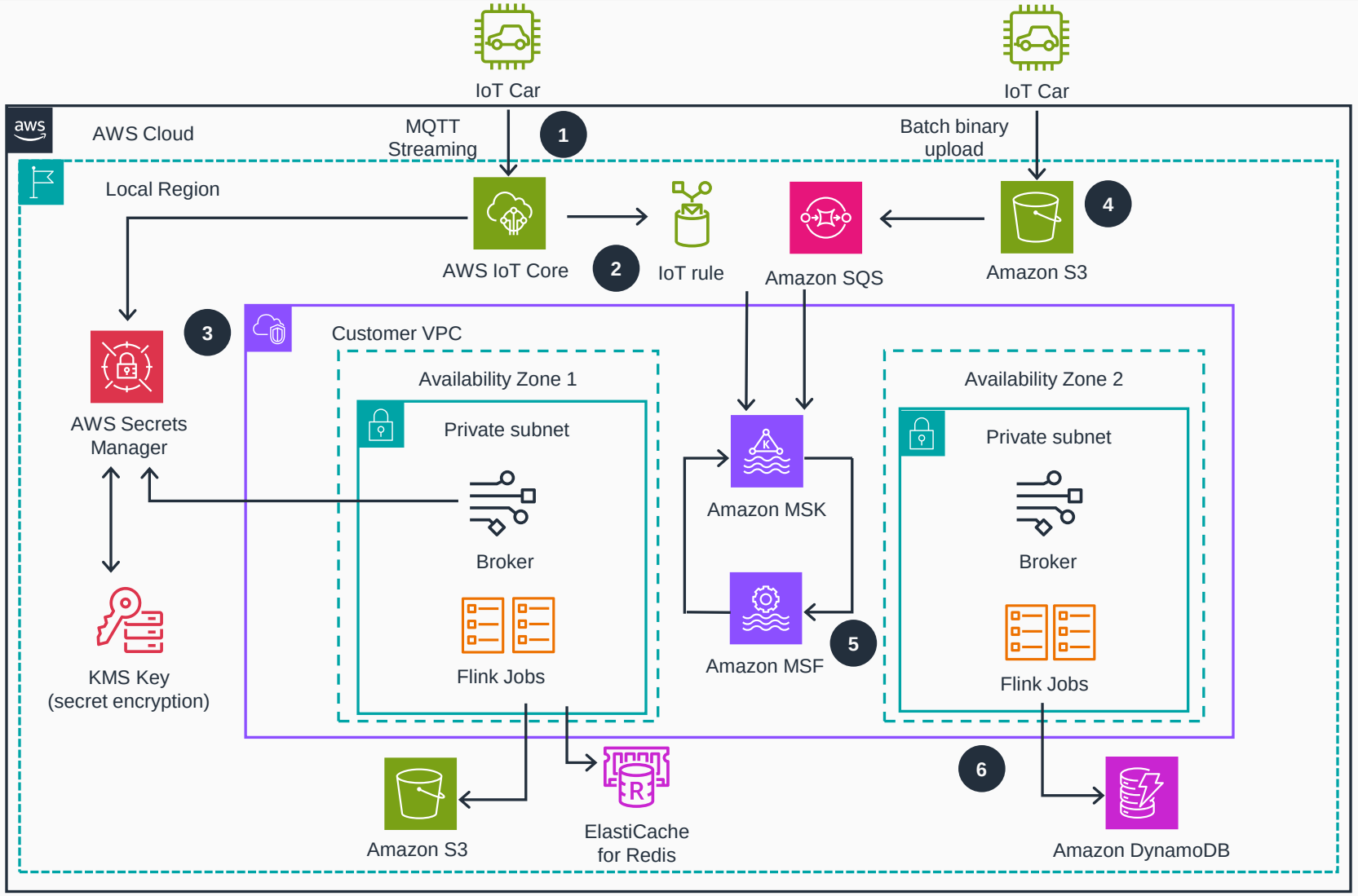
Reviewed for technical accuracy September 3, 2025

AWS Reference Architecture

Guidance for Connected Mobility on AWS

Connected Vehicle Ingestion Pipeline

This architecture diagram shows how to build a scalable endpoint to provide connectivity and communication services for a connected fleet of vehicles.



1. For scalable connectivity, OEMs should use **AWS IoT Core** as the managed broker to manage connectivity and data ingest from the vehicle to the cloud. **AWS IoT Core** supports X.509 mTLS authentication to support fan-in MQTT streaming from the connected fleet.
2. Using **AWS IoT Core** Basic Ingest and IoT Rules with direct integration with **Amazon Managed Streaming for Apache Kafka (Amazon MSK)**, vehicles send encoded and compressed telemetry data directly to a Kafka topic for decoding, decompression and message enrichment.
3. For the authentication method into **Amazon MSK** from the IoT Core rule, use SCRAM/SASL and store those credentials in **AWS Secrets Manager** for real-time IoT rule access.
4. OEMs also batch telemetry and upload directly to **Amazon Simple Storage Service (Amazon S3)** using a pre-signed URL. They upload the batch file directly to S3 for further processing in Kafka to downstream.
5. The **Amazon Managed Service for Apache Flink (Amazon MSF)** applications subscribe to the Kafka message stream and decode and enrich the message prior to republishing to a separate **Amazon MSK** topic. OEMs use Flink to perform streaming event processing for high throughput vehicle use cases.
6. **Amazon MSF** interacts with **Amazon DynamoDB** using the dedicated connector, which enables writing (or reading) data to **Amazon DynamoDB** tables as a sink. To enable this communication, configure the Flink application to access the VPC endpoint.

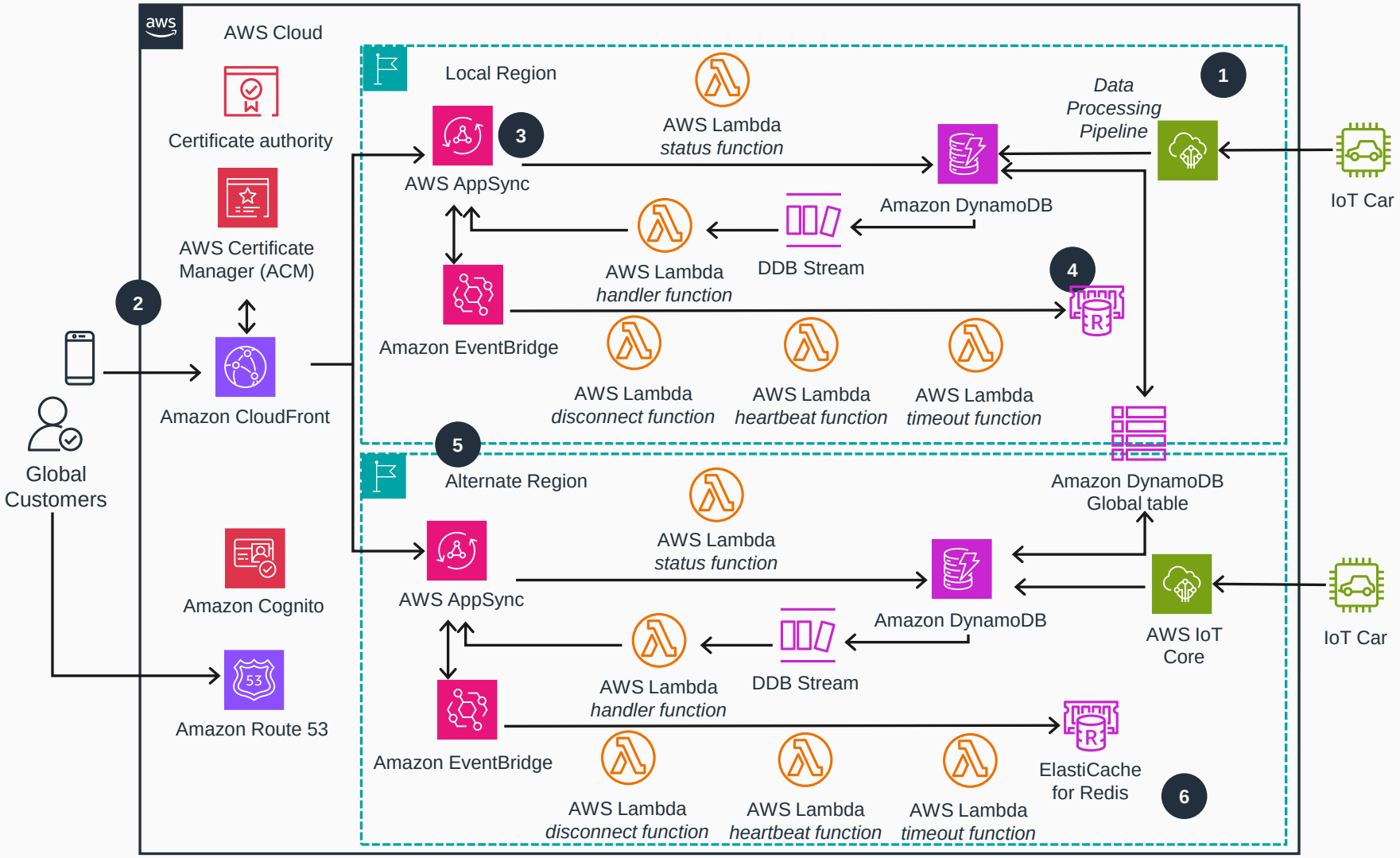
Reviewed for technical accuracy September 3, 2025

AWS Reference Architecture

Guidance for Connected Mobility on AWS

Connected Vehicle Telemetry Distribution

This architecture diagram shows how to build a GraphQL-based, multi-region data distribution platform that delivers near real-time vehicle telemetry to customers.



1. Standard data processing pipelines using **AWS IoT Core** as the connectivity layer save the vehicle's Last Known State to **Amazon DynamoDB.** From there **AWS AppSync** provides a GraphQL API to the companion application to deliver the vehicle status to the customer.

2. Implementing **Amazon CloudFront** seamlessly routes customer requests to the API in the AWS Region with the lowest latency to the client's location, reducing latency for end users while increasing the application's availability by providing GraphQL API endpoints in multiple Regions.

3. Customers want the companion application to be up-to-date and accurate. Subscriptions for this scenario are better as the data changes are small and incremental relative to the large amount of information displayed. **AWS AppSync** enables those clients to listen to real-time data changes through GraphQL subscriptions.

4. The vehicle's Last Known State is stored in **Amazon DynamoDB** and uses global tables to replicate the state between regions. This provides an added layer of multi-region redundancy, reducing latency to the user.

5. Using **Amazon Event Bridge** as an **AWS AppSync** target to enrich events before sending to subscribers and to create a persistence API to ensure mutations are only sent when a customer is connected.

6. The persistence API keeps track of the connection current state in **ElasticCache for Redis** to ensure only companion applications currently connected to AWS receive mutations from **AWS AppSync**.

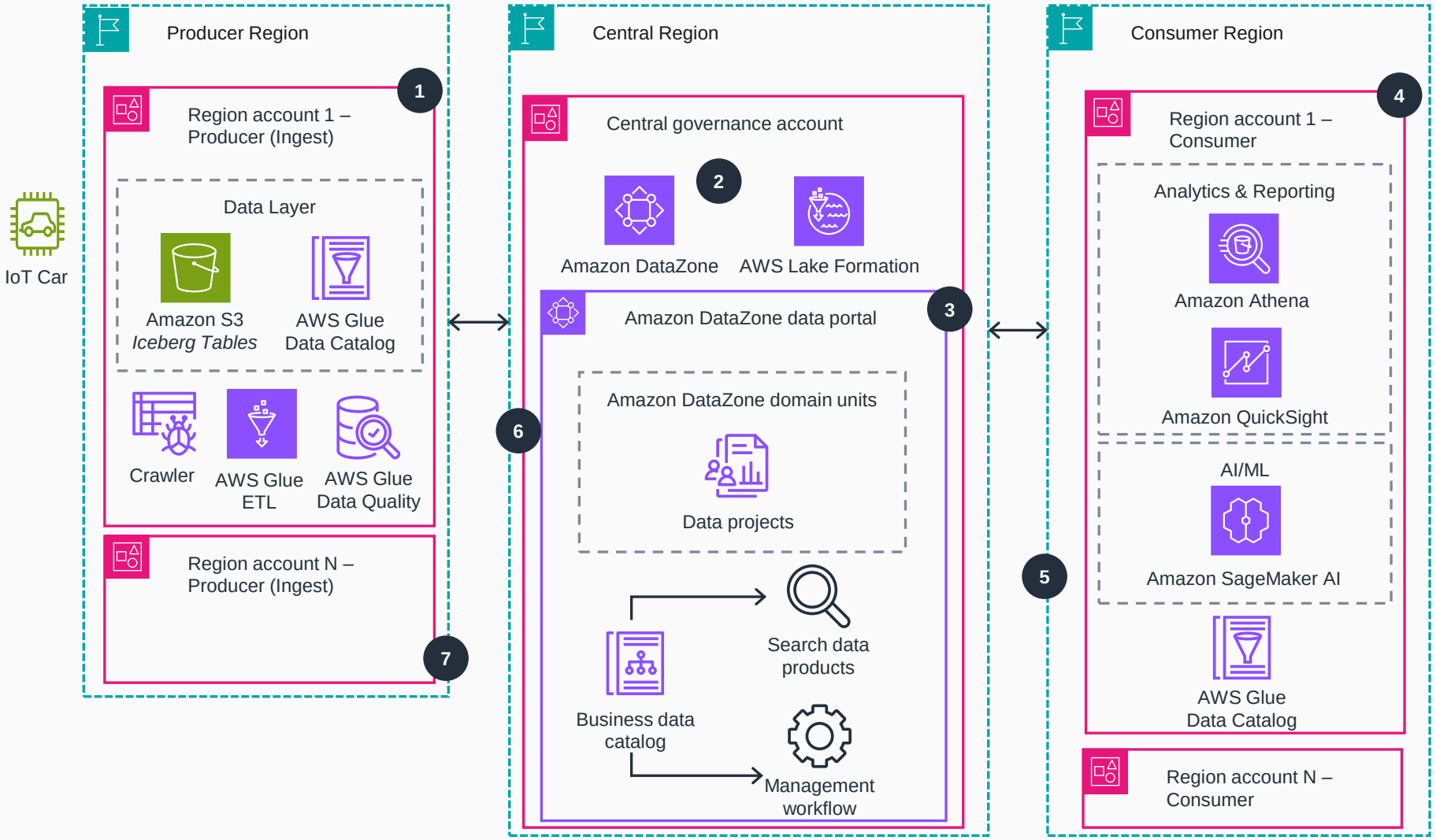
Reviewed for technical accuracy September 3, 2025

AWS Reference Architecture

Guidance for Connected Mobility on AWS

Connected Vehicle Data Mesh

This architecture diagram shows how to build a custom data mesh service on AWS that helps deliver data products via managed processes to end consumers. This slide shows steps 1-5.



1. The data producers (vehicles) publish data products in the business catalog provided by the **Amazon DataZone** data portal hosted in the central governance account. The data producers publish data products in the **AWS Glue** Data Catalog of the central governance account. **AWS Lake Formation** manages access to the entities of the central Data Catalog.

2. Data consumers (users) log in to the data portal with their AWS credentials or single sign-on credentials. They browse the catalog and search for the data products of their interest with keywords. The data mesh concept focuses on a decentralized, product-driven organization with explicit data products, responsible agile teams, and strong ownership inside domains.

3. After the data users belonging to the consumer teams find the data product of their interest, they request access to the data. **Amazon DataZone** has a built-in access-management workflow that the data owner uses to review and approve the request.

4. The data consumer teams consume the data to empower their artificial intelligence and machine learning (AI/ML), analytics and reporting, and extract, transform, and load (ETL) use cases.

5. Data consumers can request access to respective datasets through the **Amazon DataZone** data portal. Data providers who are responsible for granting access to their datasets can grant **AWS Lake Formation** fine-grained permissions to a consumer account.

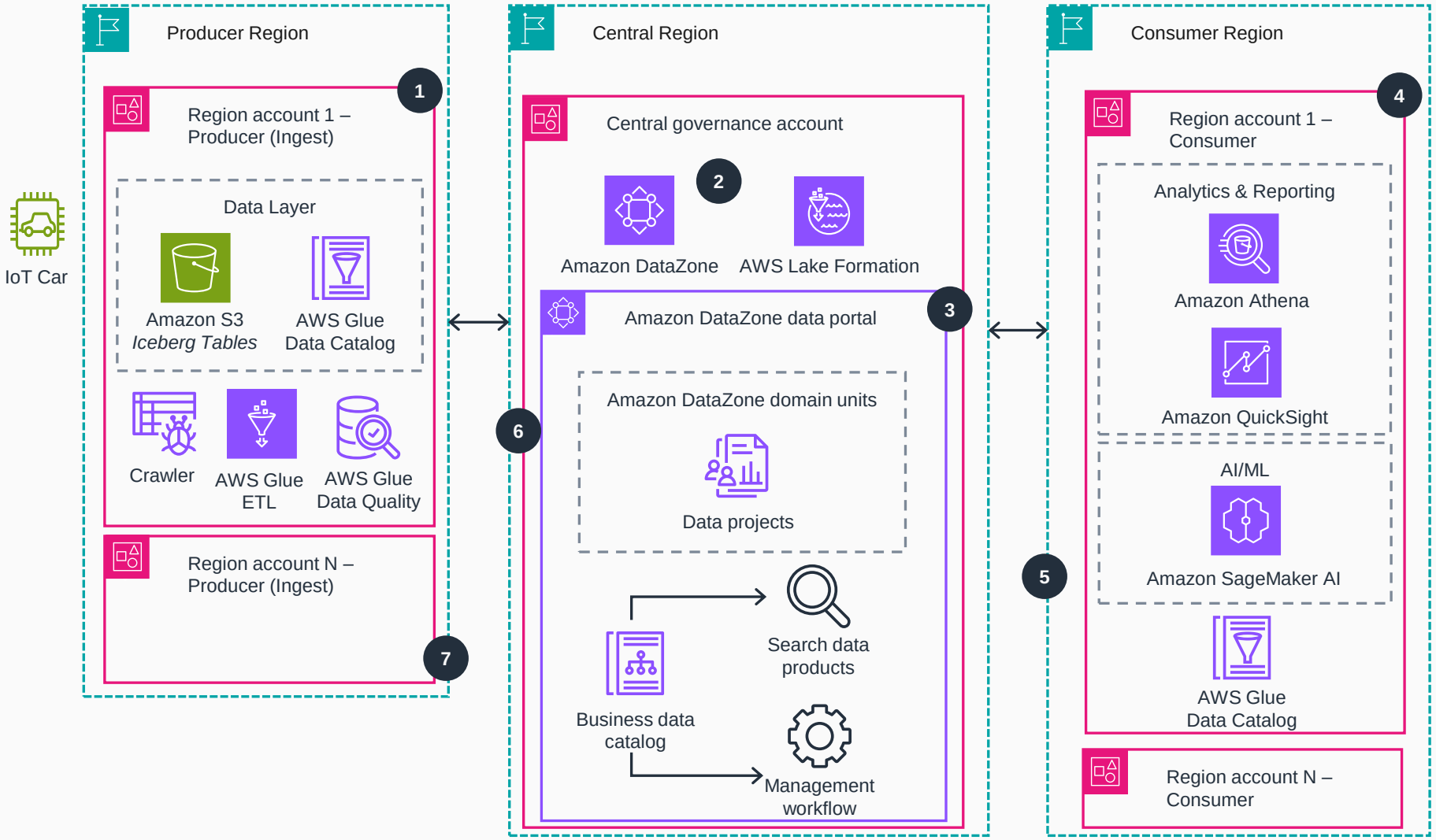
Reviewed for technical accuracy September 3, 2025

AWS Reference Architecture

Guidance for Connected Mobility on AWS

Connected Vehicle Data Mesh

This architecture diagram shows how to build a custom data mesh service on AWS that helps deliver data products via managed processes to end consumers. This slide shows steps 6-7.



6 To ensure PII data stays in region, out of access to unauthorized accounts, perform proper data masking and anonymization to protect PII, especially when sharing data across domains. Use secondary **Amazon Simple Storage Service (Amazon S3)** buckets with sanitized data that data consumers might need to access.

7 To ensure OEMs comply with local regulatory laws such as the EU Data Act and GDPR, store data in **Amazon S3**, as it has built-in controls to help customers with access control (with fine granular access and geo-restrictions), monitoring and logging, encryption at rest, and strong compliance frameworks and security standards.

Reviewed for technical accuracy September 3, 2025

AWS Reference Architecture