Connected Mobility Solution on AWS: Implementation Guide

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.
# Table of Contents

**Solution overview** .................................................................................................................. 1
  - Features and benefits ............................................................................................................ 2
  - Use cases ........................................................................................................................... 4
  - Concepts and definitions ................................................................................................. 5

**Architecture overview** .......................................................................................................... 8
  - CMS on AWS solution overview ...................................................................................... 8
    - Automotive Cloud Developer Portal (ACDP) and Backstage ..................................... 9
    - Authentication ............................................................................................................... 9
    - AWS IoT Core and MQTT ............................................................................................ 9
    - Vehicle Provisioning ..................................................................................................... 9
    - Connect and Store ......................................................................................................... 10
    - API .............................................................................................................................. 10
    - Alerts ......................................................................................................................... 10
    - EV Battery Health ....................................................................................................... 10
    - Vehicle Simulator ....................................................................................................... 10
  - Automotive Cloud Developer Portal deployment ............................................................ 11
  - CMS on AWS module deployment .................................................................................... 13

**AWS Well-Architected design considerations** .................................................................... 14
  - Operational excellence .................................................................................................. 14
  - Security .......................................................................................................................... 14
  - Reliability ...................................................................................................................... 15
  - Performance efficiency .................................................................................................. 15
  - Cost optimization .......................................................................................................... 15
  - Sustainability ............................................................................................................... 15

**Architecture details** ........................................................................................................... 17
  - Automotive Cloud Developer Portal .............................................................................. 17
  - Backstage module ......................................................................................................... 19
  - Authentication module .................................................................................................. 21
  - Vehicle Provisioning module ......................................................................................... 23
  - Connect and Store module ............................................................................................ 25
  - API module ................................................................................................................... 27
  - CMS Alerts module ....................................................................................................... 28
  - EV Battery Health module ............................................................................................. 31
  - Vehicle Simulator module .............................................................................................. 34
AWS services in this solution ........................................................................................................ 35

Plan your deployment .................................................................................................................. 40

Cost ........................................................................................................................................... 40

  CMS on AWS static cost tables ................................................................................................. 40

  Example costs ............................................................................................................................ 43

Security ....................................................................................................................................... 52

  Authentication and authorization .............................................................................................. 52

  IAM roles and policies .............................................................................................................. 52

  Amazon CloudFront .................................................................................................................. 53

  Amazon API Gateway .............................................................................................................. 54

  AWS AppSync .......................................................................................................................... 54

  Backstage ................................................................................................................................. 54

  Security groups ......................................................................................................................... 54

  PII data .................................................................................................................................... 54

  Customer managed AWS KMS keys ....................................................................................... 55

  AWS WAF ................................................................................................................................. 55

Supported AWS Regions ............................................................................................................ 55

Quotas ......................................................................................................................................... 56

  Quotas for AWS services in this solution .................................................................................. 56

  AWS CloudFormation quotas .................................................................................................... 56

Deployment prerequisites ............................................................................................................ 57

  Prerequisite tools ..................................................................................................................... 57

  Install prerequisites ................................................................................................................ 58

  Install solution dependencies ................................................................................................. 60

  Create an Amazon 53 hosted zone ......................................................................................... 60

  Set up environment variables ................................................................................................. 60

Deploy the solution ..................................................................................................................... 62

  Before you deploy .................................................................................................................... 62

  Deployment process overview ................................................................................................. 62

  Step 1: Deploy ........................................................................................................................ 63

  Step 2: Bootstrap AWS Proton ................................................................................................. 64

  Step 3: Deploy CMS modules by using Backstage ................................................................ 66

    CMS module deployment order .............................................................................................. 66

    Example module deployment by using Backstage ................................................................. 67

Monitoring the solution with Service Catalog AppRegistry .................................................... 72

  Activate CloudWatch Application Insights ............................................................................. 73
Activate AWS Cost Explorer ................................................................. 74
Confirm cost tags associated with the solution ........................................ 74
Activate cost allocation tags associated with the solution .......................... 75

Securing the solution .................................................................................. 77
Network Access Control ............................................................................ 77

Troubleshooting ......................................................................................... 78
Problem: Lambda Runtime not supported .................................................... 78
Problem: Multiple ProvisionedVehicles active certificates ....................... 78

Uninstall the solution .................................................................................. 79
Capture the deployment UUID ..................................................................... 79
Delete CMS modules in AWS Proton and CloudFormation ....................... 79
Delete AWS Proton Service Templates, Environment, and Environment Templates ......................................................................................... 81
Delete the Backstage CloudFormation Stacks ............................................. 83
Delete the CMS on AWS CloudFormation stacks ...................................... 86
Manually cleanup resources ....................................................................... 87
  Deleting the Amazon S3 buckets .............................................................. 88
  Deleting the Amazon DynamoDB tables .................................................. 88
  Deleting the Amazon CloudWatch logs .................................................. 89
  Deleting the AWS KMS customer managed keys .................................... 89
  Deleting the Amazon Cognito user pools ................................................. 90
  Deleting the Amazon Relational Database Service Snapshots ............... 91

Developer guide ......................................................................................... 92
Source code ............................................................................................... 92

Reference .................................................................................................... 93
Anonymized data collection ........................................................................ 93
Contributors .............................................................................................. 93

Revisions .................................................................................................... 95

Notices ....................................................................................................... 97
Accelerate development and deployment of connected vehicle assets

Publication date: October 2023 (last update: February 2024)

Amazon Web Services (AWS) automotive customers have asked for ways to manage fleets with increased efficiency and reduced vehicle downtime through preventative maintenance, location tracking, improved fleet safety and security, and new software driven vehicle experiences.

The Connected Mobility Solution (CMS) on AWS addresses these needs, and provides various capabilities for vehicles and customers to interact with the AWS Cloud. This solution allows you to:

• Communicate between vehicles and the AWS Cloud.
• Manage and orchestrate CMS on AWS deployments from a centralized developer platform.
• Securely authenticate and authorize users and services.
• Onboard vehicles into AWS IoT Core.
• Create vehicle profiles for storing data about registered vehicles.
• Capture and store telemetry data emitted from registered vehicles.
• Query stored vehicle data.
• Create alerts and subscribe to notifications based on data thresholds.
• Visualize vehicle telemetry data through a provided dashboard.
• Simulate connected vehicle data emission.
• Add additional modules for your unique use cases.

CMS on AWS implements an opinionated deployment mechanism for managing a connected vehicle platform. Original equipment manufacturers (OEMs), tier one suppliers, and fleet operators can manage and configure all or a subset of these capabilities for an end-to-end connected vehicle solution.

Vehicle telemetry data, such as speed, oil temperature, tire pressure, and geolocation generated from car sensors can provide near real-time data ingestion for analytics and machine learning (ML) use cases. CMS on AWS makes this data available for external consumption to third parties. You can leverage charging mechanisms and role-based authorization for usage-based insurance scores, connected accident advisor, and package delivery services.
CMS on AWS provides a modular catalog approach that lets you independently enable workloads specific to your use cases. You can deploy and configure workloads independently or as a whole system to begin securely and economically deriving insights from your connected vehicle data.

This implementation guide provides an overview of CMS on AWS, its reference architecture and components, considerations for planning the deployment, and configuration steps for deploying CMS on AWS to the AWS Cloud.

This implementation guide is intended for solution architects, business decision makers, DevOps engineers, data scientists, and cloud professionals who want to implement CMS on AWS in their environment.

Use this navigation table to quickly find answers to these questions:

<table>
<thead>
<tr>
<th>If you want to . . .</th>
<th>Read . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Know the cost for running this solution.</td>
<td>Costs</td>
</tr>
<tr>
<td>The estimated cost for running this solution</td>
<td></td>
</tr>
<tr>
<td>in the us-east-1 Region is USD $21.60 per</td>
<td></td>
</tr>
<tr>
<td>month.</td>
<td></td>
</tr>
<tr>
<td>Understand the security considerations for</td>
<td>Security</td>
</tr>
<tr>
<td>this solution.</td>
<td></td>
</tr>
<tr>
<td>Know how to plan for quotas for this solution.</td>
<td>Quotas</td>
</tr>
<tr>
<td>Know which AWS Regions are supported for</td>
<td>Supported AWS Regions</td>
</tr>
<tr>
<td>this solution.</td>
<td></td>
</tr>
<tr>
<td>Access the source code and optionally use the</td>
<td>GitHub repository</td>
</tr>
<tr>
<td>AWS Cloud Development Kit (AWS CDK) to deploy</td>
<td></td>
</tr>
<tr>
<td>the solution.</td>
<td></td>
</tr>
</tbody>
</table>

### Features and benefits

The CMS on AWS provides the following features:
Modularity

CMS on AWS is designed to simplify maintaining and implementing the solution's modularity. Modularity's primary benefit is separating concerns and outlining clear boundaries between functional sets so that the solution flexibly meets connected vehicle system demands. This allows interchanging CMS on AWS modules with bespoke implementations, which meet the boundary definitions defined by CMS on AWS standards, as well as implementing your own custom modules into the solution.

Automotive Cloud Developer Portal

Deploying a complex modular system demands a well-defined and streamlined method for selecting, configuring, and deploying each module. This component uses Backstage for the presentation layer, AWS Proton for orchestration, and AWS CodeBuild along with AWS CloudFormation to deploy and manage specified modules.

Authentication

CMS on AWS provides a user pool through Amazon Cognito, which you can populate with trusted users. These users, in addition to CMS on AWS services, can then be authenticated by communicating with Amazon Cognito's provided OAuth endpoints, and retrieving the appropriate JSON Web Tokens (JWTs). CMS on AWS then provides means to validate the integrity of JWTs for authenticated users and services, and authorize the association with the appropriate user pool and app client.

Vehicle provisioning

CMS on AWS registers vehicles as AWS IoT Core things to help you to securely monitor vehicles, their certificates, and their policies. Vehicle provisioning begins by registering with a claim certificate that the solution generates during deployment. The claim certificate includes a provisioning template that configures AWS IoT Core thing, certificate, and policy creation. After certificate is registered, the solution provides provisioned vehicles an individual certificate and public/private key pair, which you can use to connect repeatedly in the future. This is the process defined by fleet provisioning.

Storage

This solution provides a simple storage mechanism for simulated and provisioned vehicle data with Amazon Simple Storage Service (Amazon S3). The solution ingests data from pre-defined Message Queuing Telemetry Transport (MQTT) topics, and stores it in both JSON and Parquet data formats.
API

CMS on AWS provides the ability to query vehicle data stored within the solution for use in other CMSs and customer-built modules. CMS on AWS uses an AWS AppSync GraphQL application programming interface (API) that builds and runs Amazon Athena queries to provide near real-time data directly from the CMS on AWS data lake (an Amazon S3 bucket).

Alerts

CMS on AWS provides the ability to send alerts based on customizable user subscriptions through AWS AppSync GraphQL API operations and Amazon Simple Notification Service (Amazon SNS). These alerts can be triggered from other CMS on AWS modules, and sent to the user through configurable notification settings.

Electric Vehicle battery health

CMS on AWS provides the ability to visualize EV battery telemetry data and configure alerts based on data thresholds. This is done through an Amazon Managed Grafana dashboard. Electric Vehicle (EV) battery telemetry data is obtained by running Amazon Athena queries through the CMS API module.

Simulation

For both developers and customers, it is important to have a method for generating simulated vehicle telemetry data. This allows for testing the solution in real time, while also generating datasets. CMS on AWS provides a method for simulating up to 10 vehicles at once, with configurations for how the data is generated (interval, amount, and duration) and a customizable schema to define the payload that is generated.

Integration with AWS Service Catalog AppRegistry and Application Manager, a capability of AWS Systems Manager

This solution includes a Service Catalog AppRegistry resource to register the solution's CloudFormation template and its underlying resources as an application in both Service Catalog AppRegistry and Application Manager. With this integration, you can centrally manage the solution's resources and enable application search, reporting, and management actions.

Use cases

Vehicle connection

Use cases
Vehicle manufacturers, fleet owners, and vehicle owners can securely register and receive unique credentials by using the fleet provisioning by claim process. You can use these credentials for future connections of the vehicle and securely manage them with AWS IoT Core. This facilitates disabling and replacing vehicle credentials at any time. Connected vehicles can then emit data to AWS IoT Core MQTT topics for the solution to ingest and store.

**Query vehicle data**

Vehicle manufacturers, fleet owners, and vehicle owners can leverage flexible queries using GraphQL for data analysis and visualization, ML applications, and stateful representation of individual vehicles. Data is available in near real-time as it is emitted from connected vehicles. Future modules and customer applications can use this feature to realize value from the collected data.

**Monitor vehicle data**

Vehicle manufacturers, owners, and fleet managers can monitor vehicle data through visualizations and alerts available through a configured Amazon Managed Grafana dashboard. Alerts can query vehicle data periodically and send notifications to users when alert thresholds are breached. Visualization is updated in near real-time and you can configure alerts to be evaluated at a desired interval. Furthermore, users can monitor data with Amazon SNS by using the CMS Alerts module, which you can configure with custom alert conditions and functionality from pre-built or custom CMS on AWS modules.

**Vehicle simulation**

Data and entity simulation is a valuable feature for the development and use of connectivity solutions. The CMS Vehicle Simulator module allows simulating vehicle provisioning and registration (onboarding), data generation, and data ingestion by publishing to MQTT topics. Connected vehicle engineers can use the CMS Vehicle Simulator module to test the solution in a variety of usage scenarios, as well as generate datasets. Fleet managers can use simulations to showcase functionality and ensure the effectiveness of their solution's configuration.

**Concepts and definitions**

This section describes key concepts and defines terminology within the context of CMS on AWS:

**alert**
Alerts refer to any notification sent to a user because of rules set up on stored telemetry data, or triggers from CMS on AWS modules. Alert functionality is currently provided through both the CMS Alerts module (Amazon SNS alerts) and CMS EV Battery Health module (Amazon Managed Grafana alerts).

**Backstage**

Backstage is an open-source project used by CMS on AWS. It provides a portal for managing deployments of CMS on AWS modules by integrating with AWS Proton.

**fleet provisioning by claim**

fleet provisioning by claim is a provisioning method that is used to authenticate and register vehicles the first time they connect to AWS IoT Core.

**Grafana**

Grafana is a popular open-source visualization and dashboarding platform that allows users to query, visualize, alert on, and understand data. Amazon Managed Grafana is a fully managed service for Grafana. CMS on AWS uses Amazon Managed Grafana to provide a visualization and alert dashboard for vehicle telemetry data.

**Message Queuing Telemetry Transport (MQTT)**

MQTT is an OASIS standard messaging protocol used by AWS IoT Core. It is designed as an extremely lightweight publish-subscribe messaging transport.

**provisioning**

Provisioning refers to the process of registering a vehicle in AWS IoT Core by using a recognized claim certificate. Registered vehicles receive a unique certificate and credentials to connect again in the future.

**simulation**

Simulation refers to the capability of the CMS Vehicle Simulator module to simulate vehicles emitting telemetry data. A single simulation can produce randomized data for multiple simulated vehicles at a time, and emit data at regular intervals for a specified duration.

**tier one supplier**
A tier one supplier manufactures and supplies original equipment manufacturer (OEM) companies with components. They supply parts and devices that an OEM needs in order to complete a product.

**Vehicle Signal Specification**

Vehicle Signal Specification (VSS) is a common data schema for standardizing vehicle data formats.

For a general reference of AWS terms, see the [AWS glossary](#) in the AWS General Reference.
Architecture overview

This section provides architecture diagrams with high level descriptions for the entire solution, the deployment of the Automotive Cloud Developer Portal, and the deployment of subsequent CMS on AWS modules. Outside of deployment resources, each installation of CMS on AWS is unique based on which modules you deploy. The deployment architecture described in this section is consistent across deployments of CMS on AWS.

CMS on AWS solution overview

The high-level architectural descriptions for the CMS on AWS modules displayed in this diagram are as follows:
Automotive Cloud Developer Portal (ACDP) and Backstage

The Automotive Cloud Developer Portal (ACDP) is the centralized platform for deploying subsequent CMS on AWS modules. The ACDP uses the CMS Backstage module as its presentation layer to provide a configurable developer platform for managing and monitoring the deployment of CMS on AWS modules. For more details, see Automotive Cloud Developer Portal and Backstage module.

Authentication

The CMS Authentication module allows for the authentication and authorization of users and services throughout the solution. The module provides a default identity provider option through Amazon Cognito. The solution provides two AWS Lambda functions, which allow users to run the authorization code flow to retrieve user tokens, also functions to validate and authorize JSON Web Tokens for both users and services. For more details, see Authentication module.

AWS IoT Core and MQTT

AWS IoT Core MQTT topics are the primary method for communicating events between the CMS on AWS modules. Messages published to MQTT from CMS on AWS modules can be consumed by, and invokerules configured by, other modules. AWS IoT Core is also used as the primary management and storage system for provisioned vehicles. For more details on the usage of AWS IoT Core, see Vehicle Provisioning module.

Vehicle Provisioning

The CMS Vehicle Provisioning module provides means to onboard and register vehicles with AWS IoT Core. Deploying the module checks for the existence of, and if not found creates, a claim certificate and private key pair for use with fleet provisioning by claim. This claim certificate is linked to a well-defined provisioning template, which controls how vehicles are provisioned and informs the AWS IoT policy that is given to newly provisioned vehicles.

Using the claim certificate, a vehicle can retrieve a unique certificate to allow for further communication with AWS IoT Core. Registering invokes AWS IoT rules linked to Lambda functions. These functions check for vehicle authorization and create and manage vehicle records in Amazon DynamoDB. At the end of the process, the solution registers an AWS IoT Core thing for the vehicle that is linked to credentials safely stored in AWS Secrets Manager. For more details, see Vehicle Provisioning module.
**Connect and Store**

A centralized Amazon S3 bucket deployed within the CMS Connect and Store module serves as the reservoir for all CMS on AWS data. Centralized data storage allows for querying of vehicle telemetry data and enabling alerts based on data insertion and thresholds. For more details, see Connect and Store module.

**API**

CMS on AWS users can interact with vehicle telemetry data stored in the CMS on AWS data lake through the CMS API module. API endpoints are provided through AWS AppSync, which expects GraphQL requests. AWS AppSync endpoints use Lambda functions to build and run Amazon Athena queries on vehicle data stored in Amazon S3.

**Alerts**

The CMS Alerts module allows you to receive notifications invoked by data stored in the CMS on AWS data lake. CMS modules can publish to Amazon SNS topics defined by the CMS Alerts module by utilizing an API provided through AWS AppSync. You can subscribe to these same topics to receive email notifications. For more details, see Alerts module.

**EV Battery Health**

For monitoring stored data, CMS on AWS users can use the CMS EV Battery Health module. This module provides a dashboard through Amazon Managed Grafana, which is authenticated by AWS IAM Identity Center. From the dashboard, users can visualize data and setup alerts based on configurable data thresholds. For more details, see EV Battery Health module.

**Vehicle Simulator**

The CMS Vehicle Simulator module provides a user interface (UI) and backend engine for creating, operating, and monitoring simulations of vehicle data emissions. Simulations are configurable by interval, number of vehicles, and overall durations. They also support either a custom payload schema, or the provided default VSS schema.

This solution runs simulations by using AWS Step Functions, backed by a series of AWS Lambda functions. The simulator handles scaling, AWS IoT Core provisioning and registration, telemetry data generation, generation intervals, and total emission quantity. For more details, see Vehicle Simulator module.
Automotive Cloud Developer Portal deployment

Deploying this solution with the default parameters deploys the following components of the ACDP.

Automotive Cloud Developer Portal deployment architecture

Note
CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.
The high-level process flow for the solution components deployed with AWS CDK is as follows:

1. **Automotive Cloud Developer Portal** – To aid in orchestrating deployments of CMS on AWS modules, you first deploy the ACDP. This deployment is handled with [CloudFormation](https://aws.amazon.com/cloudformation/) templates defined with AWS CDK. The ACDP creates an [AWS Proton](https://aws.amazon.com/proton/) environment template for each module and a CMS Backstage deployment pipeline. The pipeline then deploys the [CMS Backstage module](#).

2. **Proton Environments** – The ACDP supports the deployment of CMS on AWS modules through AWS Proton. Deploying via AWS Proton requires environment templates to define the shared environment to be used by the deployed resources. Deploying the ACDP manages the packaging of CMS on AWS environment source code and the creation of AWS Proton environment templates for future module deployments.

3. **Backstage Deployment Pipeline** – The ACDP deploys a CI/CD pipeline that configures all the steps necessary to deploy the CMS Backstage module. This is accomplished with [AWS CodeBuild pipeline projects](https://aws.amazon.com/codebuild/), which use build specification files to define their actions, and [Amazon Elastic Container Registry](https://aws.amazon.com/ecr/) (Amazon ECR) to store the Backstage [Docker](https://www.docker.com/) image. For details on the structure of the pipeline and each build step, see [Backstage module](#).

4. **Deploying CMS on AWS modules** – When the ACDP is configured, there are three different ways to deploy CMS on AWS modules (see [Deploy the solution](#) for details):
   
   - CMS Backstage deployment – By using the CMS Backstage portal, you can deploy CMS on AWS modules from AWS Proton service templates. Backstage integrates with AWS Proton to orchestrate the deployments.
   
   - AWS Proton deployment – Similar to Step 4a, but without using CMS Backstage, you can deploy the CMS on AWS modules directly from the AWS Proton console by creating and deploying service templates.
   
   - AWS CDK CLI – If you don't want to use AWS Proton, you can individually deploy the CMS on AWS modules directly from the repository by running CDK deploy commands.

5. **CMS Backstage module** – The CMS Backstage module is the presentation layer for the ACDP. An [Elastic Load Balancing](https://aws.amazon.com/elastice煲) Application Load Balancer connects with [Amazon Route 53](https://aws.amazon.com/route53/) and an [Amazon Elastic Container Service](https://aws.amazon.com/ecs/) (Amazon ECS) cluster group setup with [AWS Fargate](https://aws.amazon.com/fargate/) tasks. The CMS Backstage module allows for deploying CMS on AWS modules through a graphical user interface. For more information, see [Backstage module](#).
CMS on AWS module deployment

Deploying CMS on AWS modules uses the CMS Backstage module and AWS Proton configurations displayed in the following diagram.

**CMS on AWS module deployment architecture**

**Note**: AWS CloudFormation resources are created from AWS CDK constructs.

**Note**

The high-level process flow for the solution components deployed with AWS CDK as follows:

1. **Deploying CMS on AWS modules** – When the ACDP is configured, there are three different ways to deploy CMS on AWS modules: CMS Backstage, AWS Proton, and AWS CDK CLI. For more information, see Deploy the solution.
2. **CMS Backstage module** – The presentation layer for the ACDP, which integrates with AWS Proton to aid in the deployment of CMS on AWS modules. For more information, see [Backstage module](#).

3. **External deployment assets** – CMS Backstage and AWS Proton require external deployment assets to provide information about the deployment of CMS on AWS modules. These assets are available from an Amazon S3 assets bucket, which is populated during pre-deploy steps. The assets include the AWS Proton service template `.tar` files and `template.yaml` files to instruct CMS Backstage on its deployment configuration. The templates define parameters required for deployment and the subsequent steps to take to begin the AWS Proton deployment.

4. **AWS Proton** – Much of the deployment structure of CMS on AWS is driven by AWS Proton. CMS on AWS instructs AWS Proton on what to deploy and how to deploy through three `.yaml` configuration files (`spec`, `schema`, and `manifest`). Some manual setup is required to create, register, and deploy [AWS Proton service templates](#) through CMS Backstage (see Step 2: Bootstrap Proton).

### AWS Well-Architected design considerations

This solution uses the best practices from the [AWS Well-Architected Framework](#), which helps customers design and operate reliable, secure, efficient, and cost-effective workloads in the cloud.

This section describes how the design principles and best practices of the Well-Architected Framework were applied when building this solution.

**Operational excellence**

This section describes how we architected this solution using the principles and best practices of the [operational excellence pillar](#).

The built-in CI/CD pipeline enables a standardized deployment strategy for the ACDP and CMS Backstage module, as well as supporting the further managed deployment of CMS on AWS modules with AWS Proton. CMS on AWS sends logging and metrics to [Amazon CloudWatch](#) throughout the entire solution. The infrastructure is managed and operated by AWS CDK, with deployment assets stored in Amazon S3 for use with CMS Backstage.

**Security**

This section describes how we architected this solution using the principles and best practices of the [security pillar](#).
All internet accessible endpoints are protected by authentication (OAuth2) with Amazon Cognito. All databases are closed off from anything external to the AWS account where they are deployed and have rotating credentials where applicable. CMS on AWS data is encrypted at rest and in transit with rotating encryption keys. Permissions are locked down to zero-trust principles or least-privilege; the most restrictive choice is made where possible. Parameters that contain sensitive information are stored in AWS Secrets Manager and rotated, while remaining parameters are kept in Parameter Store, a capability of AWS Systems Manager.

**Reliability**

This section describes how we architected this solution using the principles and best practices of the reliability pillar.

CMS on AWS uses primarily serverless AWS services, which provides resiliency, uptime, and automatic scaling. All appropriate Amazon S3 buckets have versioning enabled and are backup protected. All DynamoDB tables have point-in-time recovery, and customer data is not deleted when you uninstall the solution.

**Performance efficiency**

This section describes how we architected this solution using the principles and best practices of the performance efficiency pillar.

All compute and performance efficiency relates to usage and not a base cost. Complex tasks are delegated to appropriate AWS services that provide built-in, efficient functionality. You can deploy in supported AWS Regions to keep your data closer to where it's being used and processed, minimizing delays.

**Cost optimization**

This section describes how we architected this solution using the principles and best practices of the cost optimization pillar.

AWS Billing and Cost Management provide cost observation and analysis. CMS on AWS follows a consumption model, so costs are driven by usage.

**Sustainability**

This section describes how we architected this solution using the principles and best practices of the sustainability pillar.
This solution uses managed and serverless services to minimize the environmental impact of the backend services.
Architecture details

This section describes the components and AWS services that make up this solution and the architecture details on how these components work together.

Automotive Cloud Developer Portal

Automotive Cloud Developer Portal architecture

The initial deployment of CMS on AWS includes the ACDP, which assists in managing the deployment of CMS on AWS modules. The ACDP provides a presentation layer through the
CMS Backstage module, which you can use to select, configure, and deploy individual modules. Backstage integrates with AWS Proton for deployment orchestration, which uses AWS CodeBuild to run the deployment. A powerful and flexible platform emerges that enables flexibility and quick insights into the deployed infrastructure.

Deploying the ACDP creates and configures AWS Proton environment templates for future module deployments. The solution builds these AWS Proton environments using source code, which is uploaded to Amazon S3. A custom resource Lambda function communicates with Amazon S3 and AWS Proton to create the necessary templates.

The deployment also creates and configures an AWS CodePipeline for deploying the CMS Backstage module. To configure the Backstage deployment, a variety of AWS CloudFormation parameters are required at deployment time. For details, see the CloudFormation Parameters section.

The pipeline handles the deployment of the CMS Backstage module, and runs four stages:

1. **Source Backstage** - Sources the Backstage source code from an Amazon S3 bucket.
2. **Backstage Environment Deploy** - Deploys the Backstage environment stack.
3. **Backstage Image Build** - Builds the Backstage docker image and pushes it to a private Amazon ECR repository.
4. **Backstage Deploy** - Deploys the CMS Backstage module stack.

Once deployed, you can then use the CMS Backstage module to [deploy CMS on AWS modules](#).
CMS Backstage module architecture

CMS on AWS uses Backstage as an optional approach for deploying its modules. The CMS Backstage module is deployed and configured to use the hosted zone specified in the deployment parameters as the host for the Backstage portal (see Deploy the solution). Using CMS Backstage, given the pre-requisite deployment setup is completed, you can register CMS on AWS module templates within Backstage. CMS on AWS modules can then be deployed directly from the Backstage portal (see Deploy CMS Modules with Backstage). When deploying the CMS Backstage module an Amazon Cognito user pool is created with an initial user to enable authorization to the Amazon Route 53 hosted zone.

The CMS Backstage module is separated into two parts: the CMS Backstage environment and the CMS Backstage main stack. The environment contains the Aurora PostgreSQL-Compatible Edition database, as well as an Amazon S3 bucket that acts as the Backstage catalog. The main stack contains the Amazon EC2 resources, as well as the Amazon Route 53 domain and Amazon Cognito authentication setup.
Backstage uses an Amazon VPC (Virtual Private Cloud) to create a private network to help protect some of its resources. Specifically, Backstage uses the following resources within its VPC:

- **Elastic Load Balancing** – The Application Load Balancer connects with the Amazon Route 53 domain, as well as the Backstage Fargate service, to help orchestrate and balance tasks.

- **Amazon ECS** – Creates a cluster, combined with AWS Fargate, to group task definitions and provide the container image through Amazon ECR.

- **Amazon ECR** – Stores the Backstage image to be supplied to the docker container associated with the AWS Fargate task definitions.

- **AWS Fargate** – Combined with Amazon ECS, allows for defining task definitions with associated containers, and running those containers without needing to manage Amazon EC2 instances.

- **Amazon Aurora PostgreSQL** – The relational database used by Backstage.

The CMS Backstage module is integrated with AWS Proton to enable its deployment functionality (see [Bootstrap AWS Proton](#)). The module creates components for each of the other CMS on AWS modules, which are associated with an AWS Proton service template. Deploying a CMS on AWS module through the CMS Backstage modules requires providing the location of an Amazon S3 bucket, which stores the appropriate module's AWS Proton template .tar, and the template.yaml file, which provides deployment instructions to CMS Backstage. When a CMS Backstage component is deployed, it uses the template.yaml file to instruct the process and performs the following steps:

1. Write a spec.yaml file for the module to the Amazon S3 CMS on AWS assets bucket.
2. Create an AWS Proton service for the module.
3. Write the module's catalog info to the Amazon S3 catalog bucket.
4. Register the module within the CMS Backstage module's catalog.

After creating the AWS Proton service for the module, the module's deployment begins through AWS Proton and AWS CodeBuild. Configuration details for the deployment are provided to AWS Proton through three YAML files. For more details, see the [AWS Proton documentation](#).
Connected Mobility Solution on AWS
Implementation Guide

Authentication module

CMS Authentication module architecture

The CMS Authentication module provides a means for CMS on AWS users and internal services to authenticate and authorize themselves for use with CMS on AWS APIs. Users can sign up and communicate with the provided identity provider to retrieve authorization tokens through the authorization code grant flow, while services can communicate with a separate app client through the client-credentials grant flow.

The Authentication module provides a default identity provider (IdP) option through Amazon Cognito.

Deploying this module creates an Amazon Cognito user pool with no users. This user pool is configured with both a service and user app client. The user app client supports the authorization code grant flow, which is recommended to prevent exposing the ID and access tokens to attack.
vectors. The service app client supports the client credentials grant flow with a client secret, so that internal CMS on AWS services can safely authenticate themselves.

There are two Lambda functions provided to support authentication and authorization functionality: the token validation Lambda function and token exchange Lambda function.

The token validation Lambda function implements a JWT validation flow that is defined by the standard OAuth2.0 protocol and Amazon Cognito validation procedure. This validation can be done for both user and service JWTs, and is performed as follows:

1. Validate the integrity of the JWT signature.
2. Check the token expiration.
3. Authorize the token's claims and scope against the user pool configuration and appropriate app client.

You can use the token exchange Lambda function to exchange an authorization code, retrieved from the authorization code grant flow and OAuth /token endpoint, for an ID and access token. These tokens are used to authorize the user against CMS on AWS API operations. The token exchange Lambda function communicates with the Amazon Cognito OAuth API to verify the integrity of the authorization code against the associated user app client.

The solution uses a Proof Key for Code Exchange (PKCE) code verifier to protect against injection attacks, which could intercept the user tokens. Once retrieved, user tokens are validated using the token validation Lambda described previously, before being returned to the user.

The CMS Authentication module also implements multiple app client management Lambda functions, which you can use to manage the state of the app clients associated with the Amazon Cognito use.
Vehicle Provisioning module architecture

The CMS Vehicle Provisioning module defines, creates, and manages the certificates, policies, and vehicle profiles for registered vehicles. This module allows vehicles to register with AWS IoT Core using a secure communication system encrypted with the TLS v1.2 protocol. On deployment, the module generates a unique claim certificate to allow vehicle registration. Registered vehicles receive credentials to allow connections to AWS IoT Core.

This module uses the fleet provisioning by claim workflow which is supported by AWS IoT Core. When the CloudFormation stack is created or updated, a custom resource Lambda function initiates, configuring AWS IoT Core to enable AWS IoT Core thing events for detecting vehicle registrations. The custom resource function retrieves an existing claim certificate and private key from AWS Secrets Manager. If not found, the claim certificate and private key are created, activated, and stored in AWS Secrets Manager. The claim certificate has an attached provisioning template that configures AWS IoT Core thing, certificate, and policy creation.
After set up, a user can connect to AWS IoT Core using the claim certificate, private key, and Amazon Root CA, which is used to sign the claim certificate. After connecting, the user can retrieve a new unique certificate and private key for registering the vehicle.

To register the vehicle, the solution calls for AWS IoT Core RegisterThing endpoint with credentials. Starting the registration process invokes the pre-provision Lambda function, which completes the following:

1. Finds existing certificates for this vehicle in the ProvisionedVehicles DynamoDB table. If certificates are found and not already INACTIVE, the function deactivates them and updates the ProvisionedVehicles records to reflect the change.
2. Creates a record in the ProvisionedVehicles DynamoDB table for the new combination of Vehicle Identification Number (VIN) and certificate in the PENDING_ACTIVATION status.
3. Searches for the vehicle in the AuthorizedVehicles DynamoDB table. If the vehicle is not found, the solution prevents registration by deleting the certificate.
4. Returns a registration-allowed Boolean.

If provisioning was allowed for this vehicle, AWS IoT Core creates or updates the thing for this vehicle. The certificate for this vehicle is activate, and creates and attaches a policy to the certificate as defined in the provisioning template.

Continuing the registration process creates an AWS IoT Core thing that invokes a post-provision Lambda function, which completes the following:

1. Updates the new certificate's status to ACTIVE in the ProvisionedVehicles DynamoDB table.
2. Deletes this vehicle's INACTIVE certificates from AWS IoT Core.
3. Deletes certificates' record in the ProvisionedVehicles DynamoDB table.

After a registered vehicle connects to AWS IoT Core, the vehicle can then publish to the vehicleactive AWS IoT MQTT topic to signal a successful connection to the solution. Messages to this topic invoke the initial-connection Lambda function, which flips a Boolean for the vehicle record to indicate that the vehicle has successfully connected with their certificate at least one time.
Note
Publishing to the vehicleactive topic is not an automatic part of registration, and is not implemented as a part of the solution.

Connect and Store module

Connect and Store module on AWS

AWS IOT MQTT Protocol
/cms/data/#

AWS IoT Core

CMS Connect & Store module

AWS IoT Core
Kinesis Rule

AWS IoT Core
S3 Rule

AWS IoT Core
Vehicle Alarm Rule

AWS Lambda
Vehicle Alarm Function

Amazon Kinesis Data Firehose
Write parquet

AWS Glue Schema Table

Amazon S3 Root Bucket

AWS Secrets Manager

Fetch client credentials

AWS AppSync Alerts Publish API

External Resources

Validate and authorize

Amazon Cognito OAuth API
Connect and Store module architecture

The CMS Connect and Store module is the primary data lake for the solution. This module provides the collection of telemetry data payloads from well-defined AWS IoT MQTT topics. Data is stored in an Amazon S3 bucket in both JSON and Apache Parquet format. Centralizing all CMS on AWS telemetry data into a single data lake enables CMS on AWS modules to retrieve data without needing to interface with AWS IoT Core directly.

A single Amazon S3 bucket acts as the data lake within the module. The Amazon S3 objects are prefixed with the timestamp, data format, and vehicle identifier.

This module uses three AWS IoT rules. Two of those rules subscribe to a broad AWS IoT MQTT topic invoked on vehicle data ingestion. The first rule invokes an action property that writes the JSON payload to the appropriate Amazon S3 bucket. The second rule invokes an action property which passes the payload to Amazon Data Firehose. A delivery stream validates and transforms the payload using AWS Glue. The stream then writes the Parquet-formatted payload to the appropriate Amazon S3 bucket. Using Firehose allows the CMS Connect and Store module to handle a high throughput of telemetry data payloads. The third rule subscribes to a broad MQTT topic, which can be invoked by vehicle emission. This rule delivers the received notification to the intended CMS Alerts API, which forwards the notification to the appropriate destination.
The CMS API module provides the ability for users and services to query data stored by the solution. This is done by leveraging an AWS AppSync GraphQL API that integrates with the default CMS on AWS data lake deployed from the Connect and Store module. You can integrate it with a data lake or resources external to the solution or pre-built modules by changing the module's configuration.

The AWS AppSync API uses Amazon Athena with a Lambda resolver data source, allowing the data to be queried the moment it is stored in the solution. This provides a near real-time representation of the state of vehicles.
By default, the CMS API module integrates with the CMS Connect and Store module's Amazon S3 bucket as its data source and the CMS Authentication module's IdP for authorization on API requests.

The following steps explain how a request to the API is handled:

1. The client makes a valid GraphQL request to the API endpoint with a bearer token provided in the authorization header. The bearer token should be an access token obtained from the token endpoint of the configured IdP.
2. The token must be validated and authorized against the chosen IdP's user pool (this functionality is available via the CMS Authentication module). Further authorization logic can determine whether the user has permission for the operations and fields selected.
3. Once authorized, the context of the GraphQL query is sent to a Lambda resolver. The resolver builds and invokes an Amazon Athena query using the selected fields and provided arguments.
4. Amazon Athena uses the configured AWS Glue table to query the vehicle data from the data lake, and store the results in a separate Amazon S3 bucket.
5. The Lambda resolver parses the results into JSON format and returns them to AWS AppSync.
6. The AWS AppSync API receives the results and returns them to the client.
The CMS Alerts module enables CMS on AWS and customer implemented modules to send alerts to subscribed users, and allows users to manage their alert subscriptions. This is done by leveraging two AWS AppSync GraphQL API operations; one for user subscription management and another for publishing messages to user subscribed Amazon SNS topics. The CMS Alerts module requires the CMS Authentication module as a prerequisite to authenticate API requests.

The user subscription AWS AppSync API uses DynamoDB with a Lambda resolver data source. This Lambda function stores user subscription information in a DynamoDB table as well as subscribes and unsubscribes users from an Amazon SNS topic.

The following steps explain how a request to the user subscription API is handled:

1. The client makes a valid GraphQL request to the API endpoint with a bearer token provided in the authorization header. The bearer token should be an access token obtained from the token endpoint of the configured IdP.
2. The token must be validated and authorized against the chosen IdP's user pool (this functionality is available via the CMS Authentication module). Further authorization logic can determine whether the user has permission for the operations and fields selected.
3. Once authorized, the context of the GraphQL query is sent to a Lambda resolver. The resolver updates the user’s subscription preferences on Amazon SNS and makes a corresponding update to the DynamoDB table.
4. The Lambda resolver parses the results into JSON format and returns them to AWS AppSync.
5. The AWS AppSync API receives the results and returns them to the client.

The publish API endpoint uses a Lambda function as its data source. When a CMS module sends a request, the Lambda function relays the message through the system.

The following steps explain how a request to the publish API is handled:

1. The client makes a valid GraphQL request to the API endpoint with a bearer token provided in the authorization header. The bearer token should be an access token obtained from the token endpoint of the configured IdP.
2. The token must be validated and authorized against the chosen IdP's user pool (this functionality is available via the CMS Authentication module). Further authorization logic can determine whether the user has permission for the operations and fields selected.
3. Once authorized, the context of the GraphQL query is sent to a Lambda resolver. The Lambda resolver publishes this message to the central Amazon SNS topic, which notifies the central Amazon Simple Queue Service (Amazon SQS) queue.

4. This queue triggers the create-alerts Lambda function, which stores this message in a notifications DynamoDB table.

5. This DynamoDB table has a DynamoDB stream enabled, which notifies a send-notifications Lambda function about the changes in the DynamoDB table.

6. The send-notifications Lambda function publishes all the notifications in the stream to their corresponding Amazon SNS topics.

7. The users subscribed to these Amazon SNS topics receive an email notification.
EV Battery Health module architecture

The CMS EV Battery Health module implements an Amazon Managed Grafana workspace to facilitate visualizing and alerting based on vehicle telemetry data. The workspace is created with AWS IAM Identity Center authentication. The workspace uses Amazon Athena as its query engine for panels and alerts, with queries written in ANSI SQL. The Amazon Managed Grafana console is used to add users to the workspace and assign users to Admin, Editor, or Viewer roles.
After deployment, the solution creates an Admin API key for the workspace, which allows creating and updating the dashboard panels and alert rules. Amazon Managed Grafana API keys expire in 30 days, so the API key is stored in Secrets Manager and rotated every 29 days. The workspace is provisioned with **unified alerting** to leverage the alerting capabilities of Amazon Managed Grafana.

The EV battery health dashboard contains gauge panels for Remaining Useful Life (%), Remaining Charge (%), Average Temperature (°C), and Current Voltage (V). The solution creates stat panels to visualize the remaining driving range from battery and hybrid energy sources. The solution also creates a time series graph panel, which shows the historical data for the Remaining Useful Life (%) of the battery. The dashboard can be parameterized by VIN, allowing data to be visualized for each vehicle by selecting its corresponding VIN from the dropdown at the top of the dashboard. The dashboard data model is stored in a versioned and encrypted Amazon S3 bucket, which uses an event-driven approach to invoke a Lambda function if the data model object is modified. This facilitates modifying the dashboard data model post deployment without having to update the deployment itself.

The solution creates alert rules to send an alert message whenever the remaining charge or remaining useful life of the battery drops below a configurable threshold. A time series query is run, and the results of the query are reduced to a single value by a reducer function. The reduced value is then compared with a threshold and evaluated for whether the queried data violates the alert rule.

You can configure alert rules as to how often they evaluate queries, and how long they should wait before firing an alert. If the alert rule is breached, the solution sends an alert message to an Amazon SNS topic. The Amazon SNS topic has a Lambda function subscriber that processes the alert messages. The solution sends messages to the alerts endpoint exposed by the CMS Alerts module. The API call to the CMS Alerts module is authenticated by an access token obtained through the client credentials grant flow, using the service app client provided by the CMS Authentication module.

The deployment of the CMS EV Battery Health module uses the AWS CDK library to create the CloudFormation template. CDK support for Amazon Managed Grafana is limited as it is a relatively new service offering, which has a heavy reliance on custom resources during deployment. Resource dependencies created outside the scope of the module are managed by Systems Manager parameters. These parameters are then used in IAM Identity Center policies to give appropriate permissions to other resources. The following steps are performed in the deployment:

- **Create Amazon Managed Grafana workspace** - An Amazon Managed Grafana workspace is created with the authentication provider set to AWS IAM Identity Center.
• **Create Amazon Managed Grafana API key** - An Admin API key is created for the Amazon Managed Grafana workspace and is stored in Secrets Manager.

• **Create Amazon Athena data source** – Amazon Athena is added as a data source to the workspace with the Amazon S3, AWS Glue, and Amazon Athena resources used for storing and indexing telemetry data taken as input to the module.

• **Create dashboard** – An Amazon S3 bucket to store Amazon Managed Grafana assets is created. A custom resource creates the dashboard JSON data model and uploads it to the bucket. The `PutObject` action in the bucket invokes a Lambda function, which calls the Amazon Managed Grafana HTTP API to create the dashboard in the workspace.

• **Provision alerting** – The Amazon Managed Grafana workspace does not have alerting capabilities by default. To support alerting, additional updates to the workspace configuration are made with custom resources post-deploy.

• **Create alert rules** - A custom resource creates the alert rules JSON data model and uploads it to the Amazon Managed Grafana assets bucket. The `PutObject` action in the bucket triggers a Lambda function, which calls the Amazon Managed Grafana HTTP API to create the alert rules in the workspace.

• **Create alert contact points and notification policy** – Alert rules must be configured with a contact point (where the alert messages are routed to) and a notification policy (which alert rules are routed to which contact points). The CMS EV Battery Health module uses Amazon SNS as the contact point for alerts.
Vehicle Simulator module

CMS Vehicle Simulator module architecture

The CMS Vehicle Simulator module helps customers test vehicle communication with AWS IoT Core without the need for physical vehicles. This module provides a web-based GUI (Graphical user interface) that allows customers to create and simulate fleets of connected vehicles.

Simulated fleets emit data payloads at configured intervals from either a user-defined template or the default provided VSS schema. Simulated vehicles also onboard themselves with AWS IoT Core. You can monitor the simulation and observe how CMS on AWS services are processing the data.
The GUI for the CMS Vehicle Simulator module is hosted as a static webpage in Amazon S3 and distributed through Amazon CloudFront. When deploying the module, an Amazon Cognito user pool is created with an initial user to enable authorization.

A REST API is implemented through Amazon API Gateway and backed by Lambda functions. This API provides functionality to create and store devices and simulations into DynamoDB tables.

Invoking the simulation API invokes the start of an AWS Step Function workflow, which does the following:

1. Maps device types to their corresponding payload configuration.
2. Retrieves payload configurations from a DynamoDB table.
3. Maps the total device count to Lambda function invocations.
4. For each Lambda function invocation, publishes the simulated payload to an AWS IoT MQTT topic.
5. Checks the overall duration and if elapsed, goes to step 7. If not elapsed, goes to step 6.
6. Waits the configured delay interval, then repeats steps 4 and 5.
7. Updates the DynamoDB table to represent the finished simulation state.

Note
This module is designed to simulate data for testing. It is not recommended for production environments.

AWS services in this solution

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon API Gateway</td>
<td>Core. Hosts the REST API endpoints in the CMS Vehicle Simulator module.</td>
</tr>
<tr>
<td>AWS AppSync</td>
<td>Core. Provides GraphQL API for querying vehicle data.</td>
</tr>
<tr>
<td>AWS service</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Amazon Athena</td>
<td><strong>Core.</strong> Performs queries on vehicle data stored in Amazon S3.</td>
</tr>
<tr>
<td>AWS Certificate Manager (ACM)</td>
<td><strong>Core.</strong> Generates certificates used for validation of HTTPS requests to the CMS Backstage module.</td>
</tr>
<tr>
<td>AWS Chalice</td>
<td><strong>Core.</strong> Framework used to define and deploy the CMS Vehicle Simulator module's API.</td>
</tr>
<tr>
<td>AWS CDK</td>
<td><strong>Core.</strong> Enables infrastructure as code for the entirety of CMS on AWS.</td>
</tr>
<tr>
<td>AWS CloudFormation</td>
<td><strong>Core.</strong> Manages deployments for the solution infrastructure.</td>
</tr>
<tr>
<td>Amazon CloudWatch</td>
<td><strong>Core.</strong> The solution's code and infrastructure emit logs to Amazon CloudWatch.</td>
</tr>
<tr>
<td>AWS CodeBuild</td>
<td><strong>Core.</strong> Defines build steps to aid in deploying and managing the solution.</td>
</tr>
<tr>
<td>AWS CodePipeline</td>
<td><strong>Core.</strong> Runs collections of AWS CodeBuild steps for deploying and managing the solution.</td>
</tr>
<tr>
<td>Amazon Cognito</td>
<td><strong>Core.</strong> Authenticates users and internal services across the solution.</td>
</tr>
<tr>
<td>Amazon DynamoDB</td>
<td><strong>Core.</strong> Primary non-relational data storage used in the CMS Vehicle Provisioning module for vehicle records.</td>
</tr>
<tr>
<td>AWS Glue</td>
<td><strong>Core.</strong> Validates and transforms JSON formatted payloads of a specified schema into Parquet format.</td>
</tr>
<tr>
<td>AWS service</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>AWS IAM</strong></td>
<td><strong>Core.</strong> Authorizes CMS on AWS resources and users through associated least-privilege roles and policies.</td>
</tr>
<tr>
<td><strong>AWS IoT Core</strong></td>
<td><strong>Core.</strong> Primary service for onboarding and connecting vehicles, as well as management of those vehicles (things), their certificates, and policies. Also used for data ingestion through MQTT topics.</td>
</tr>
<tr>
<td><strong>AWS KMS</strong></td>
<td><strong>Core.</strong> Encrypts data in transit and at rest throughout the solution.</td>
</tr>
<tr>
<td><strong>Amazon Data Firehose</strong></td>
<td><strong>Core.</strong> High throughput writing and transformation of MQTT topic payloads to Amazon S3.</td>
</tr>
<tr>
<td><strong>AWS Lambda</strong></td>
<td><strong>Core.</strong> A variety of runtime logic with serverless code throughout CMS on AWS.</td>
</tr>
<tr>
<td><strong>Amazon Managed Grafana</strong></td>
<td><strong>Core.</strong> Build dashboards and configure alerts based on vehicle data stored in Amazon S3.</td>
</tr>
<tr>
<td><strong>Parameter Store, a capability of AWS Systems Manager</strong></td>
<td><strong>Core.</strong> Stores configuration level information used throughout the solution.</td>
</tr>
<tr>
<td><strong>AWS Secrets Manager</strong></td>
<td><strong>Core.</strong> Stores and rotates secrets used throughout the solution.</td>
</tr>
<tr>
<td><strong>Amazon S3</strong></td>
<td><strong>Core.</strong> General purpose shared data storage used throughout CMS on AWS for vehicle data, logs, and frontend assets.</td>
</tr>
<tr>
<td><strong>Amazon SNS</strong></td>
<td><strong>Core.</strong> Used to publish and subscribe to messages.</td>
</tr>
<tr>
<td><strong>Amazon SQS</strong></td>
<td><strong>Core.</strong> Used to deliver messages to Lambda functions.</td>
</tr>
<tr>
<td>AWS service</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Aurora PostgreSQL-Compatible</strong></td>
<td><strong>Supporting.</strong> Database used by the CMS Backstage module to aid with deployment.</td>
</tr>
<tr>
<td><strong>Amazon CloudFront</strong></td>
<td><strong>Supporting.</strong> Provides a domain name to serve static web content for the CMS Vehicle Simulator module.</td>
</tr>
<tr>
<td><strong>Amazon ECS</strong></td>
<td><strong>Supporting.</strong> Simplifies the deployment, management, and scaling of the CMS Backstage module.</td>
</tr>
<tr>
<td><strong>Amazon ECR</strong></td>
<td><strong>Supporting.</strong> Image repository for Dockerized containers used by the CMS Backstage module.</td>
</tr>
<tr>
<td><strong>Elastic Load Balancing (ELB) - Application Load Balancer</strong></td>
<td><strong>Supporting.</strong> Provides network connections to Amazon ECS tasks running CMS Backstage to aid with deployment.</td>
</tr>
<tr>
<td><strong>AWS Fargate</strong></td>
<td><strong>Supporting.</strong> Manages CMS Backstage ECS containers without the need for managing Amazon EC2 instances.</td>
</tr>
<tr>
<td><strong>Amazon IAM Identity Center</strong></td>
<td><strong>Supporting.</strong> Service that provides authentication for the Amazon Managed Grafana workspace.</td>
</tr>
<tr>
<td><strong>AWS Proton</strong></td>
<td><strong>Supporting.</strong> Orchestrates deployments for all CMS on AWS modules.</td>
</tr>
<tr>
<td><strong>Amazon Route 53</strong></td>
<td><strong>Supporting.</strong> Domain hosting integration used by the CMS Backstage module.</td>
</tr>
<tr>
<td><strong>AWS Step Functions</strong></td>
<td><strong>Supporting.</strong> Orchestrates and manages Lambda functions used in the CMS Vehicle Simulator module.</td>
</tr>
<tr>
<td>AWS service</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>Amazon VPC</strong></td>
<td><strong>Supporting.</strong> Networking boundary used by the CMS Backstage module.</td>
</tr>
<tr>
<td><strong>AWS X-Ray</strong></td>
<td><strong>Supporting.</strong> Traces runs of the AWS Step Functions in the CMS Vehicle Simulator module.</td>
</tr>
<tr>
<td><strong>Amazon Location Service</strong></td>
<td><strong>Optional.</strong> Geographical functionality for the CMS Vehicle Simulator module.</td>
</tr>
</tbody>
</table>
Plan your deployment

This section describes the cost, security, Regions, and other considerations prior to deploying the solution.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of this revision, the cost for running this solution with the default settings in the US East (N. Virginia) is approximately **$38.40 a month**.

See the pricing webpage for each AWS service used in this solution.

We recommend creating a budget through AWS Cost Explorer to help manage costs. Prices are subject to change. For full details, see the pricing webpage for each AWS service used in this solution.

**CMS on AWS static cost tables**

**CMS on AWS total**

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS KMS</td>
<td>27</td>
<td>Customer managed keys</td>
<td>$27.00</td>
</tr>
<tr>
<td>AWS Secrets Manager</td>
<td>6</td>
<td>Sensitive data stored in secrets</td>
<td>$2.40</td>
</tr>
<tr>
<td>Amazon Managed Grafana</td>
<td>1</td>
<td>1 Workspace x API license</td>
<td>$9.00</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td><strong>$38.40 / month</strong></td>
</tr>
</tbody>
</table>

Automotive Cloud Developer Portal
<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS KMS</td>
<td>6</td>
<td>Customer managed keys</td>
<td>$6.00</td>
</tr>
<tr>
<td>AWS Secrets Manager</td>
<td>1</td>
<td>Backend secret</td>
<td>$0.40</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$6.40 / month</strong></td>
</tr>
</tbody>
</table>

**Backstage module**

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS KMS</td>
<td>2</td>
<td>Customer managed keys</td>
<td>$2.00</td>
</tr>
<tr>
<td>AWS Secrets Manager</td>
<td>1</td>
<td>PostgreSQL secret</td>
<td>$0.40</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$2.40 / month</strong></td>
</tr>
</tbody>
</table>

**Authentication module**

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Secrets Manager</td>
<td>2</td>
<td>Client secrets for Amazon Cognito app clients</td>
<td>$0.80</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$0.80 / month</strong></td>
</tr>
</tbody>
</table>

**Vehicle Provisioning module**
### Connected Mobility Solution on AWS

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS KMS</td>
<td>2</td>
<td>Customer managed keys</td>
<td>$2.00</td>
</tr>
<tr>
<td>AWS Secrets Manager</td>
<td>1</td>
<td>Claim certificate created on deploy</td>
<td>$0.40</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$2.40 / month</strong></td>
</tr>
</tbody>
</table>

#### Connect and Store module

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS KMS</td>
<td>4</td>
<td>Customer managed keys</td>
<td>$4.00</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$4.00</strong></td>
</tr>
</tbody>
</table>

#### API module

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS KMS</td>
<td>1</td>
<td>Customer managed keys</td>
<td>$1.00</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>$1.00 / month</strong></td>
</tr>
</tbody>
</table>

#### EV Battery Health module

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS KMS</td>
<td>3</td>
<td>Customer managed keys</td>
<td>$3.00</td>
</tr>
</tbody>
</table>
### Connected Mobility Solution on AWS Implementation Guide

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Secrets Manager</td>
<td>1</td>
<td>Amazon Managed Grafana API key secret</td>
<td>$0.40</td>
</tr>
<tr>
<td>AWS Grafana</td>
<td>1</td>
<td>1 Workspace x API license</td>
<td>$9.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Total:</strong></td>
<td><strong>$12.40 / month</strong></td>
</tr>
</tbody>
</table>

**Alerts module static**

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS KMS</td>
<td>6</td>
<td>Customer managed keys</td>
<td>$6.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Total:</strong></td>
<td><strong>$6.00 / month</strong></td>
</tr>
</tbody>
</table>

**Vehicle Simulator module**

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Amount</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS KMS</td>
<td>3</td>
<td>Customer managed keys</td>
<td>$3.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Total:</strong></td>
<td><strong>$3.00 / month</strong></td>
</tr>
</tbody>
</table>

**Example costs**

This solution provides zero-cost-on-no-use. The following examples show a variety of use cases.

**Use case 1 - Solution deployed for 100 devices connected with little to no analytics run.**
<table>
<thead>
<tr>
<th>Service</th>
<th>Configuration summary</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQTT</td>
<td>• Number of devices (MQTT) (100)</td>
<td>$2.59</td>
</tr>
<tr>
<td></td>
<td>• Average size of each message (5 KB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average size of each record (1 KB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average size of each record (1 KB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average number of actions applied per rule (2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average size of each message (5 KB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average number of connection minutes for a device (24,000)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of messages for a device (24,000)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of rules invoked (5)</td>
<td></td>
</tr>
<tr>
<td>S3 Standard</td>
<td>• S3 Standard storage (1 GB per month)</td>
<td>$0.04</td>
</tr>
<tr>
<td></td>
<td>• PUT, COPY, POST, LIST requests to S3 Standard (720)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GET, SELECT and all other requests from S3 Standard (720)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Data returned by S3 Select (1 GB per month)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Data scanned by S3 Select (4 GB per month)</td>
<td></td>
</tr>
</tbody>
</table>
## Configuration summary

### Amazon Data Firehose
- Source type (Direct PUT or Kinesis Data Stream)
- Data records units (exact number)
- Record size (5 KB)
- Dynamic Partitioning (Add On) (Enabled)
- Data format conversion (optional) (Enabled)
- Average ratio of data processed to VPC vs data ingested (1.3)
- Number of records for data ingestion (1,200 per day)
- Number of subnets for VPC delivery (0)
- Average size objects delivered (64 MB)
- JQ Processing (optional) (Enabled)
- Average JQ expected processing hours (70)

<table>
<thead>
<tr>
<th>Service</th>
<th>Configuration summary</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Data Firehose</td>
<td>• Source type (Direct PUT or Kinesis Data Stream)</td>
<td>$4.91</td>
</tr>
<tr>
<td></td>
<td>• Data records units (exact number)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Record size (5 KB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Dynamic Partitioning (Add On) (Enabled)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Data format conversion (optional) (Enabled)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average ratio of data processed to VPC vs data ingested (1.3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of records for data ingestion (1,200 per day)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of subnets for VPC delivery (0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average size objects delivered (64 MB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• JQ Processing (optional) (Enabled)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average JQ expected processing hours (70)</td>
<td></td>
</tr>
</tbody>
</table>

### Amazon Athena
- Amount of data scanned per query (1 GB)
- Total number of queries (100 per day)

<table>
<thead>
<tr>
<th>Service</th>
<th>Configuration summary</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Athena</td>
<td>• Amount of data scanned per query (1 GB)</td>
<td>$14.85</td>
</tr>
<tr>
<td></td>
<td>• Total number of queries (100 per day)</td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td>Configuration summary</td>
<td>Cost [USD]</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>DynamoDB on-demand capacity</td>
<td>• Table class (Standard), Average item size (all attributes) (3 KB)</td>
<td>$0.05</td>
</tr>
<tr>
<td></td>
<td>• Data storage size (0.1 GB)</td>
<td></td>
</tr>
<tr>
<td>Amazon CloudFront</td>
<td>• Data transfer out to internet (1 GB per month)</td>
<td>$0.09</td>
</tr>
<tr>
<td></td>
<td>• Number of requests (HTTPS) (1,000 per month)</td>
<td></td>
</tr>
<tr>
<td>Amazon CloudWatch</td>
<td>• Standard Logs: Data Ingested (0.1 GB)</td>
<td>$48.13</td>
</tr>
<tr>
<td></td>
<td>• Logs Delivered to CloudWatch Logs: Data Ingested (0.1 GB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Logs Delivered to Amazon S3: Data Ingested (0.1 GB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of Lambda functions (20), Number of requests per function (10 per day)</td>
<td></td>
</tr>
<tr>
<td>Amazon Cognito</td>
<td>• Number of monthly active users (MAU) (100)</td>
<td>$5.00</td>
</tr>
<tr>
<td></td>
<td>• Advanced security features (Enabled)</td>
<td></td>
</tr>
<tr>
<td>Amazon Managed Grafana</td>
<td>• Number of active editors/administrators (1 per workspace per month)</td>
<td>$34.00</td>
</tr>
<tr>
<td></td>
<td>• Number of active viewers (5 per workspace per month)</td>
<td></td>
</tr>
</tbody>
</table>
## Service
### Configuration summary
### Cost [USD]

| Standard topics | • DT Inbound: Not selected (0 TB per month) |
|                 | • DT Outbound: Not selected (0 TB per month) |
|                 | • Requests (10,000 per month) |
|                 | • EMAIL/EMAIL-JSON Notifications (10,000 per month) |
|                 | • Mobile Push Notifications (20,000 per month) |
| AWS AppSync API Request | • Number of API Requests (10 thousand per month) |

| Total | $109.89 / month |

### Use case 2 - Solution deployed with the following parameters.

1. Cost estimated for 1 million vehicles.
2. Certificates are rotated annually. Consider exploring non-AWS options to host your own Certificate Authority (CA).
3. A baseline of 10 alerts per vehicle per day.
4. Data ingestion to the cloud, approximately 10 MB per vehicle per day.
5. 50 signals per second (2,000) per vehicle for 5x20 hours a month – 5KB of data per minute; maintaining the message size at or below 5KB is recommended to save costs.
6. Monitoring of 100 Diagnostic Trouble Codes (DTC) per vehicle per month for vehicle health monitoring.
7. 10-15 trips per vehicle per day, with 1 MB of vehicle data stored in DynamoDB storage. Historical data is exported to Amazon S3. Vehicle profile data resides in DynamoDB, while all vehicle-generated data is stored in Amazon S3.
8. 100 Amazon SNS notification requests per vehicle per day, alongside 20 mobile push notifications per vehicle per day.


10. In the context of DevSecOps, we anticipate retaining logs for a period of three months. The cost of CloudWatch will increase because of the obligation to retain logs for more than one month, with a minimum retention period of three months.

11. The cost of Amazon S3 includes the monthly cost of Standard S3, which amounts to $4,000.00 every month.

<table>
<thead>
<tr>
<th>AWS service</th>
<th>Dimensions</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQTT</td>
<td>• Number of devices (MQTT) (1,200,000)</td>
<td>$26,910.70</td>
</tr>
<tr>
<td></td>
<td>• Average size of each message (5 KB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average size of each record (1 KB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average number of actions applied per rule (2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average size of each message (5 KB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Average number of connection minutes for a device (28,000)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of messages for a device (28,000)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of rules invoked (5)</td>
<td></td>
</tr>
<tr>
<td>S3 Standard</td>
<td>• S3 Standard storage (10 TB per month)</td>
<td>$5,249.09</td>
</tr>
<tr>
<td></td>
<td>• PUT, COPY, POST, LIST requests to S3 Standard (920,000,000)</td>
<td></td>
</tr>
<tr>
<td>AWS service</td>
<td>Dimensions</td>
<td>Cost [USD]</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>GET, SELECT, and all other requests from S3 Standard (920,000,000)</td>
<td>$11,679.87</td>
<td></td>
</tr>
<tr>
<td>Data returned by S3 Select (1 TB per month)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data scanned by S3 Select (8 TB per month)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon Data Firehose</td>
<td>Source type (Direct PUT or Kinesis Data Stream)</td>
<td>$11,679.87</td>
</tr>
<tr>
<td>Data records units (exact number)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record size (5 KB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Partitioning (Add On) (Enabled)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data format conversion (optional) (Enabled)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average ratio of data processed to VPC vs data ingested (1.3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of records for data ingestion (1,200,000,000 per day)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of subnets for VPC delivery (0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average size objects delivered (64 MB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JQ Processing (optional) (Enabled)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average JQ expected processing hours (70)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AWS service</td>
<td>Dimensions</td>
<td>Cost [USD]</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Amazon Athena</td>
<td>• Amount of data scanned per query (100 GB)</td>
<td>$53,710.94</td>
</tr>
<tr>
<td></td>
<td>• Total number of queries (110,000 per month)</td>
<td></td>
</tr>
<tr>
<td>DynamoDB on-demand capacity</td>
<td>• Table class (Standard)</td>
<td>$185.00</td>
</tr>
<tr>
<td></td>
<td>• Average item size (all attributes) (3 KB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Data storage size (10 GB)</td>
<td></td>
</tr>
<tr>
<td>Amazon CloudFront</td>
<td>• Data transfer out to internet (1 TB per month)</td>
<td>$107.58</td>
</tr>
<tr>
<td></td>
<td>• Number of requests (HTTPS) (60,000 per month)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Data transfer out to origin (1 TB per month)</td>
<td></td>
</tr>
<tr>
<td>Amazon CloudWatch</td>
<td>• Standard Logs: Data Ingested (1 TB)</td>
<td>$575.64</td>
</tr>
<tr>
<td></td>
<td>• Logs Delivered to CloudWatch Logs: Data Ingested (10 GB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Logs Delivered to Amazon S3: Data Ingested (10 GB)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of Lambda functions (20)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Number of requests per function (10,000 per day)</td>
<td></td>
</tr>
<tr>
<td>AWS service</td>
<td>Dimensions</td>
<td>Cost [USD]</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Amazon Cognito</td>
<td>• Number of monthly active users (MAU) (100)</td>
<td>$5.00</td>
</tr>
<tr>
<td></td>
<td>• Advanced security features (Enabled)</td>
<td></td>
</tr>
<tr>
<td>Amazon Managed Grafana</td>
<td>• Number of active editors/administrators (5 per workspace per month)</td>
<td>$295.00</td>
</tr>
<tr>
<td></td>
<td>• Number of active viewers (50 per workspace per month)</td>
<td></td>
</tr>
<tr>
<td>Standard topics</td>
<td>• Requests (2,000,000 per month)</td>
<td>$4,109.48</td>
</tr>
<tr>
<td></td>
<td>• EMAIL/EMAIL-JSON Notifications (200,000,000 per month)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Mobile Push Notifications (20,000,000 per month)</td>
<td></td>
</tr>
<tr>
<td>AWS AppSync API Request</td>
<td>• Number of API Requests (300,000,000 per month)</td>
<td>$1,200.00</td>
</tr>
<tr>
<td>Amazon Elastic Container Registry</td>
<td>• DT Inbound: All other Regions (1 TB per month)</td>
<td>$28.00</td>
</tr>
<tr>
<td></td>
<td>• DT Outbound: Not selected (0 TB per month)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Amount of data stored (280 GB per month)</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>$104,056.30 / month</strong></td>
</tr>
</tbody>
</table>
Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared responsibility model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about AWS security, visit AWS Cloud Security.

⚠️ Important

This solution is not designed to handle personally identifiable information (PII). See PII data for more information.

Authentication and authorization

All CMS on AWS API operations are protected through authentication requirements for both users and services. Both users or services must provide a valid access token associated with the IdP configured within the solution.

All JWTs used for authentication and authorization are validated through protocol defined by OAuth2.0 standards and Amazon Cognito. For more details, see Authentication module.

The authentication flow is protected against security risks and attacks by implementing a variety of safety procedures. These include the use of client secrets for both the user and service app client, a PKCE code verifier for user authentication, and the use of the authorization code grant flow for user authentication.

IAM roles and policies

IAM roles allow customers to assign granular access policies and permissions to services and users on the AWS Cloud.

The CMS Vehicle Simulator module uses an IAM role to give permissions to an authenticated user. These permissions include accessing the API, accessing location services, and communicating with AWS IoT Core MQTT topics.

The claim certificate's provisioning template defined in the CMS Vehicle Provisioning module uses IAM roles to give users permission to register vehicles and perform the necessary related AWS IoT Core thing, certificate, and policy actions.
CMS on AWS uses IAM roles to provide least-privilege access across all resources, and provide necessary permissions to authenticated Amazon Cognito users in the CMS Vehicle Simulator module and CMS Backstage module.

Specific instances of IAM role usage for individual services include the following:

- Provides Amazon CloudWatch logging permissions to a variety of services.
- Provides AWS KMS (Key Management Service) encrypt and decrypt permissions to a variety of services.
- Allows Lambda functions to perform their runtime operations, such as permissions for DynamoDB access, AWS IoT Core, AWS Secrets Manager, Amazon Athena, and Amazon SNS.
- Allows AWS IoT Core to write to specified Amazon S3 buckets.
- Allows Kinesis Data Streams to write to specified Amazon S3 buckets and access AWS Glue.
- Allows Amazon Athena to use Lambda function resolver.
- Allows Amazon Managed Grafana to access Amazon S3, AWS Glue, Athena, and Amazon SNS resources as needed.
- Grant permissions across the solution to execute the CMS Authentication module's token validation Lambda function.
- Grants a variety of permissions to the CMS Backstage module.
- Grants a variety of permissions to AWS CodeBuild.
- Grants a variety of permissions to the CMS Vehicle Simulator module state machine.
- Grants a variety of Amazon SNS and DynamoDB permissions to the CMS Alerts module.

**Amazon CloudFront**

This solution deploys a web console hosted in an Amazon S3 bucket. To help reduce latency and improve security, this solution includes a CloudFront distribution with an origin access identity, which is a CloudFront user that provides public access to the solution's website bucket contents. For more information, see [Restricting access to an Amazon S3 origin](https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/origin-access-identity-oidc-access-control.html) in the Amazon CloudFront Developer Guide.

Amazon CloudFront is deployed using the default CloudFront domain name and TLS certificate. The default CloudFront SSL certificate only supports TLSv1. To use a later TLS version, use your
own custom domain name and custom SSL certificate. For more information, refer to Using alternate domain names and HTTPS in the Amazon CloudFront Developer Guide.

**Amazon API Gateway**

This solution deploys an Amazon API Gateway REST API and uses the default API endpoint and SSL certificate. The default API endpoint only supports TLSv1. To use a later version of TLS, use your own domain name and custom SSL certificate. For more information, refer to Choosing a minimum TLS version for a custom domain in API Gateway in the Amazon API Gateway Developer Guide.

**AWS AppSync**

This solution deploys AWS AppSync GraphQL API operations and uses the default API endpoints and SSL certificates. The default API endpoints only support TLSv1. To use a later version of TLS, use your own domain name and custom SSL certificate. For more information, refer to Configuring custom domain names in the AWS AppSync Developer Guide.

**Backstage**

This solution deploys a customized instance of Backstage. This instance is a reference implementation and is not optimized fully for production workloads. To use backstage for production workloads, we recommend hosting your own instance. The default deployment of backstage runs on express and allows the unsafe-eval header. Productionized deployments should compile the backstage frontend into a static webpage and disable this header for optimal security.

**Security groups**

The security groups created in this solution are designed to control and isolate network traffic between the Amazon ECS tasks, an Amazon Aurora PostgreSQL database, and client requests. We recommend that you review the security groups and further restrict access as needed after the deployment is complete.

**PII data**

This solution is not designed with the advanced security protocols necessary to store, process, or handle PII. All data is encrypted in-transit and at rest; however, this solution doesn't vet or filter incoming data for PII elements. As a result, you must ensure that no PII is included in the data transmitted.
Customer managed AWS KMS keys

This solution uses encryption at rest for securing data and employs customer managed keys for customer data and AWS managed keys for AWS service data. These keys are used to automatically and transparently encrypt your data before it is written to storage layers. Some users might prefer to have more control over their data encryption processes. This approach allows you to administer your own security credentials, offering a greater level of control and visibility.

AWS WAF

This solution's default configuration doesn't deploy a web application firewall (WAF) in front of the API endpoints. To enhance your API security by setting up a WAF, you must do so manually. AWS provides an in-depth guide on how you can control access to your API Gateway with AWS WAF. For instructions on how to implement AWS WAF in front of your API and increase distributed denial of service (DDoS) protection for your web applications, see Using AWS WAF to protect your APIs.

Supported AWS Regions

This solution uses the AWS Proton service, which is not currently available in all AWS Regions. You must launch this solution in an AWS Region where AWS Proton is available. For the most current availability of AWS services by Region, see the AWS Regional Services List.

<table>
<thead>
<tr>
<th>Region name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (Ohio)</td>
<td>Asia Pacific (Tokyo)</td>
</tr>
<tr>
<td>US East (N. Virginia)</td>
<td>Canada (Central)</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>Europe (Frankfurt)</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>Europe (Ireland)</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>Europe (London)</td>
</tr>
</tbody>
</table>
Quotas

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

Quotas for AWS services in this solution

Make sure you have sufficient quota for each of the AWS services in this solution. For more information, see AWS Service Quotas.

Use the following links to go to the page for that service. To view the Service Quotas for all AWS services in the documentation without switching pages, view the information in the Service endpoints and quotas page in the PDF instead.

AWS CloudFormation quotas

Your AWS account has CloudFormation quotas that you should be aware of when launching the stack in this solution. By understanding these quotas, you can avoid limitation errors that would prevent you from deploying this solution successfully. For more information, see AWS CloudFormation quotas in the AWS CloudFormation User's Guide.
Deployment Prerequisites

This solution uses AWS CloudFormation templates and stacks to automate its deployment. The CloudFormation template specifies the AWS resources included in this solution and their properties. The CloudFormation stack provisions the resources that are described in the template. AWS Proton is used to orchestrate the CloudFormation deployments of CMS on AWS modules with AWS Proton service templates. AWS CodeBuild is used by AWS Proton to run the deployments.

Prerequisite tools

The solution requires the following development environment dependencies:

1. nvm
2. pyenv
3. pipenv
4. NVM
5. Node
6. NPM
7. Yarn
8. Pyenv
9. Python
10. Pip
11. Pipenv
12. AWS CLI
13. AWS CDK Toolkit

Required tool versions

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPM</td>
<td>9.6.7</td>
</tr>
<tr>
<td>NodeJS</td>
<td>18.17.1</td>
</tr>
<tr>
<td>Python</td>
<td>3.10.9</td>
</tr>
</tbody>
</table>
Install prerequisites

Clone the repository

Use the following command to clone the repository:

```bash
git clone https://github.com/aws-solutions/connected-mobility-solution-on-aws.git
cd connected-mobility-solution-on-aws
```

⚠️ Warning

If after a successful installation, a command is not found, you may need to restart your terminal.

NVM

Follow the [nvm installation guide](#) to install NVM. Ensure your installation properly set your path by running the script below.

```bash
nvm --version
# Expected Output: x.xx.x
```

Node / NPM

```bash
nvm install
nvm use
```

For more information see the [nvm usage guide](#) for installing the correction version of Node. Manually installing Node without the user of nvm is not recommended.

Yarn

Follow the [yarn installation guide](#) to install yarn. Ensure your installation properly set your path by running the script below.

```bash
yarn --version
```
Pyenv

Follow the [pyenv installation guide](#) to install Pyenv. You will likely need to manually add Pyenv to your PATH by following the provided instructions. Ensure your installation properly set your path by running the script below.

```
pyenv --version
# Expected Output: pyenv x.x.xx
```

Python / Pip

```
pyenv --install -s
```

For more information see the [pyenv usage guide](#) for installing the correct version of Python. Manually installing Python without the use of pyenv is not recommended.

Pipenv

Follow the [pipenv installation guide](#) to install Pipenv. You will likely need to manually add Pipenv to your PATH by following the provided instructions. Ensure your installation properly set your PATH by running the script below.

```
pipenv --version
# Expected Output: pipenv, version xxxx.xx.x
```

AWS CLI

Follow the installation instructions laid out in the [AWS CLI install page](#). This install is OS specific, and includes multiple options for both a system wide, and user specific install. Follow the install instructions most appropriate to you. Ensure your installation properly set your PATH by running the script below.

```
aws --version
# Expected Output: aws-cli/x.xx.xx ...
```

For more information see the [nvm usage guide](#) for installing the correction version of Node. Manually installing Node without the user of nvm is not recommended.
AWS CDK Toolkit

Follow the installation guide to install the AWS CDK toolkit. To ensure your installation, properly set your PATH by running the script below.

```bash
cdk --version
# Expected Output: x.xxx.x (build …)
```

Install solution dependencies

Now that you have the correct tools, you can install the dependencies used by the solution using make. After installing, we will activate the environment which contains the dependencies.

```bash
make install
source ./venv/bin/activate
```

Create an Amazon 53 hosted zone

To deploy the solution, an Amazon Route 53 Hosted Zone is required. This should be setup in the AWS account in which your installing the solution. You will provide the domain for this hosted zone in the following step when you setup your environment variable. This is a manual step. For more details, see Working with hosted zones.

Set up environment variables

To deploy the solution, a variety of environment variables are required. These environment variables will be used to provide the context to your CDK deployment.

1. ROUTE53_BASE_DOMAIN is optional. If unset, the base domain is assumed to be the same as the ROUTE53_ZONE_NAME variable. This must be set to a superset of the ROUTE53_ZONE_NAME (.*${ROUTE53_ZONE_NAME}).
2. BACKSTAGE_TEMPLATE_S3_UPDATE_REFRESH_MINS: should be set to something small such as 1 min for development. It is recommended to have longer refresh intervals for cost savings in production environments.

Create .env file (preferred option)
**Note**

Do not use quotes around values in the .env file, or the make commands will fail.

Use the following command to create a .env file:

```bash
cat > .env <<EOL
USER_EMAIL=you@domain.com
ROUTE53_ZONE_NAME=domain.com
ROUTE53_BASE_DOMAIN=subdomain.domain.com
LOG_LEVEL=info
CMS_SOLUTION_VERSION=v0.0.0
BACKSTAGE_TEMPLATE_S3_UPDATE_REFRESH_MINS=30
BACKSTAGE_NAME=Default Name
BACKSTAGE_ORG=Default Org
EOL
```

**Set environment variables (secondary option)**

Use the following command:

```bash
export USER_EMAIL="you@domain.com"
export ROUTE53_ZONE_NAME="domain.com"
export ROUTE53_BASE_DOMAIN="subdomain.domain.com"
export LOG_LEVEL="info"
export CMS_SOLUTION_VERSION="v0.0.0"
export BACKSTAGE_TEMPLATE_S3_UPDATE_REFRESH_MINS="30"
export BACKSTAGE_NAME="Default Name"
export BACKSTAGE_ORG="Default Org"
```

**Set environment variables (secondary option)**

Before continuing, run the following command:

```bash
make cdk-context
```

This will display your current CDK context which will be used for your deployment. Ensure the environment variables look correct and that all the expected values are displayed. If not, please retry setting your environment variables.
Deploy the solution

This section includes instructions for deploying CMS on AWS.

Before you deploy

Ensure you've followed the steps in the deployment prerequisites section.

• Prerequisite tools are installed.
• Solution dependencies installed.
• An Amazon Route 53 hosted zone in the deployment account.
• Environment variables set.

Deployment process overview

Follow the step-by-step instructions in this section to configure and deploy CMS on AWS into your account.

Before you launch the solution, review the cost, architecture, network security, and other considerations discussed earlier in this guide.

Time to deploy: Approximately 30–45 minutes

Step 1: Deploy

Step 2: Bootstrap AWS Proton

Step 3: Deploy CMS Modules via Backstage

⚠️ Important

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. AWS owns the data gathered through this survey. Data collection is subject to the AWS Privacy Notice.
Step 1: Deploy

Follow the configuration steps before starting this section. Ensure that the CDK CLI can find AWS credentials. For details, refer to Getting started in the AWS CDK v2 Developer Guide.

Run CDK bootstrap

If the CDK has not been used in the Region of the AWS account you are deploying to, you must first bootstrap it in order for deployments to work.

```
made bootstrap
```

Upload Amazon S3 deployment assets (Backstage template.yaml and AWS Proton service template.tar)

```
made upload-s3-deployment-assets
```

Deploy the Automotive Cloud Developer Portal

```
made deploy
```

After the AWS CDK deployment is completed, navigate to CodePipeline in the AWS Management Console and verify that the Backstage-Pipeline ran successfully.

![Pipeline Execution](image)

After the pipeline has completed, the deployment is successfully complete and Backstage is ready for use.

**Note**

It can take up to 10 minutes after the Backstage pipeline completes for Amazon Cognito's auth domain to become available for use with Backstage.
Step 2: Bootstrap AWS Proton

Note
The Amazon S3 location where deployment assets were uploaded is in your AWS account, and it should have a name with the format <AWS_ACCOUNT>-cms-resources-<AWS_REGION>.

1. Sign in to the AWS Management Console, select your Region, and navigate to the AWS Proton Service Templates page.

2. Choose Create service template for each module template that you plan to register.

3. Fill in the required fields. The following instructions detail how to register the Vehicle Simulator module template. You can apply the same steps to other modules by selecting the proper Amazon S3 path.
   - Select Use your own template bundle in Amazon S3.
   - Choose Browse Amazon S3 and locate the bucket where the templates were uploaded.
   - Locate the latest .tar for the vehicle_simulator module template. The Amazon S3 path should be in the format ../<SOLUTION_VERSION>/modules/<MODULE_NAME>/proton/. Select Choose.
• In the repository, locate the Vehicle Simulator on AWS Proton Template YAML file. Find the template name under the metadataName property located under the aws:proton:create-service action.

```
- id: create-proton-service
  name: Create AWS Proton Service
  action: aws:proton:create-service
  input:
    serviceName: ${{ parameters.component_id }}
    serviceSpecPath: ./spec.yaml
    # Update the following fields to match the resource
    region: ${{ parameters.aws_region }}
    templateName: cms_vehicle_simulator_on_aws
    templateMajorVersion: '1'
```

• Use the template name to populate the Service template name field.

• Under Compatible environment templates, select cms_environment.

---

Step 2: Bootstrap AWS Proton

---
• Leave the remaining settings as default and choose **Create Service Template**.

• After you receive a message stating *Successfully created service template* `cms_vehicle_simulator_on_aws`, select template version **1.0** and choose **Publish** to make it available for use by Backstage.

### Step 3: Deploy CMS modules by using Backstage

Follow the instructions in this section to deploy CMS modules by using Backstage.

**CMS module deployment order**

Some CMS on AWS modules have dependencies on other modules and must be deployed in order. Others do not have dependencies on other modules and can be deployed in any order, as long as the ACDP has been deployed first. The deployment order that must be observed is as follows:

**Deployment order of modules with dependencies**

1. CMS Authentication
2. CMS Alerts
3. CMS Connect and Store
4. CMS API
5. CMS EV Battery Health

Modules without dependencies

- CMS Vehicle Provisioning
- CMS Vehicle Simulator

**Example module deployment by using Backstage**

The following instructions detail how to deploy the CMS Vehicle Simulator module. The same steps can be applied to other modules by replacing the URLs and names.

1. Navigate to the CMS Backstage URL in a web browser (ROUTE53_BASE_DOMAIN that was specified during deployment).
2. Sign in to CMS Backstage using the credentials that were emailed to the user email specified during deployment.
3. Follow the prompts to create a new password and set up multi-factor authentication (MFA).
4. On Backstage, navigate to the **Create** page available from the **Catalog** menu in the side bar. Select **CHOOSE** on the **CMS Vehicle Simulator on AWS** card.
5. Fill in the form as required by the CMS Vehicle Simulator module's template and choose Next Step.
CMS Vehicle Simulator on AWS

1. Provide the required information

   Name*
   vehicle-simulator
   Unique name of the component

   Description
   Vehicle Simulator Module
   Help others understand what this component is for.

   Owner*
   auto-dev
   Owner of the component

   AWS Account ID*
   111111111111
   AWS Account ID to which the module should be deployed.

   AWS region*
   us-east-1
   AWS Region to which the module should be deployed.

   Admin email used to create the initial user*
   youremail@domain.com

BACK NEXT STEP

6. Choose Create.
CMS Vehicle Simulator on AWS

Provide the required information

Review and create

<table>
<thead>
<tr>
<th>Component Id</th>
<th>vehicle-simulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Vehicle Simulator Module</td>
</tr>
<tr>
<td>Owner</td>
<td>group:default/auto-dev</td>
</tr>
<tr>
<td>Aws Account Id</td>
<td>111111111111</td>
</tr>
<tr>
<td>Aws Region</td>
<td>us-east-1</td>
</tr>
<tr>
<td>User Email</td>
<td><a href="mailto:youremail@domain.com">youremail@domain.com</a></td>
</tr>
</tbody>
</table>

BACK      RESET  CREATE

7. Monitor the deployment and ensure that the CMS Vehicle Simulator module deploys successfully.
Example module deployment by using Backstage

- Fetch Skeleton + Template: 10 seconds
- Create AWS Proton Service: 0 seconds
- S3 Catalog Write: 0 seconds
- Register: 0 seconds

Open in catalog

START OVER  CANCEL
Monitoring the solution with Service Catalog AppRegistry

The Connected Mobility Solution on AWS includes a Service Catalog AppRegistry resource to register the CloudFormation template and underlying resources as an application in both Service Catalog AppRegistry and AWS Systems Manager Application Manager.

AWS Systems Manager Application Manager gives you an application-level view into this solution and its resources so that you can:

- Monitor its resources, costs for the deployed resources across stacks and AWS accounts, and logs associated with this solution from a central location.
- View operations data for the resources of this solution in the context of an application. For example, deployment status, CloudWatch alarms, resource configurations, and operational issues.

The following figure depicts an example of the application view for the Connected Mobility Solution on AWS stack in Application Manager.
Activate CloudWatch Application Insights

1. Sign in to the Systems Manager console.
2. In the navigation pane, choose Application Manager.
3. In Applications, choose AppRegistry applications.
4. In AppRegistry applications, search for the application name for this solution and select it.

Note

The next time you open Application Manager, you can find the new application for your solution in the AppRegistry application category.

5. In the Components tree, choose the application stack you want to activate.

Monitoring for your applications is now activated and the following status box appears:
Application insights setup complete

**Activate AWS Cost Explorer**

You can see the overview of the costs associated with the application and application components within the Application Manager console through integration with AWS Cost Explorer, which must be first activated. Cost Explorer helps you manage costs by providing a view of your AWS resource costs and usage over time. To activate Cost Explorer for the solution:

1. Sign in to the [AWS Cost AWS Management Console](#).
2. In the navigation pane, select **Cost Explorer**.
3. On the Welcome to Cost Explorer page, choose **Launch Cost Explorer**.

The activation process can take up to 24 hours to complete. After activation, you can open the Cost Explorer user interface to further analyze cost data for the solution.

**Confirm cost tags associated with the solution**

After you activate cost allocation tags associated with the solution, you must confirm the cost allocation tags to see the costs for this solution. To confirm cost allocation tags:

1. Sign in to the [Systems Manager console](#).
2. In the navigation pane, choose **Application Manager**.

3. In **Applications**, choose the application name for this solution and select it.

4. In the **Overview** tab, in **Cost**, select **Add user tag**.

5. On the **Add user tag** page, enter **confirm**, then select **Add user tag**.

The activation process can take up to 24 hours to complete and the tag data to appear.

### Activate cost allocation tags associated with the solution

After you activate Cost Explorer, you must activate the cost allocation tags associated with this solution to see the costs for this solution. The cost allocation tags can only be activated from the management account for the organization. To activate cost allocation tags:

1. Sign in to the [AWS Billing and Cost Management console](https://aws.amazon.com/billing/

2. In the navigation pane, select **Cost Allocation Tags**.

3. On the **Cost allocation tags** page, filter for the AppManagerCFNStackKey tag, then select the tag from the results shown.

4. Choose **Activate**.
The activation process can take up to 24 hours to complete and the tag data to appear.
Securing the solution

Network Access Control

EV Battery Health module

To configure network access control for the Amazon Managed Grafana workspace deployed when using the EV Battery Health module, follow the instructions provided in Managing network access to your workspace.
Troubleshooting

This section provides known issue resolution when deploying the solution.

Problem: Lambda Runtime not supported

Occasionally, an error could appear similar to Lambda Runtime (…) not supported. Supported list [...]. This is due to AWS CDK or third-party library updates that might create Lambda functions with non-supported runtime versions. To resolve this issue, you must add the necessary runtime version to the Lambda function's configuration and Lambda function aspect supported list.

Resolution

Add the necessary runtime version to the Lambda functions supported list:

1. Explicitly define the runtimes for a Lambda function in its configuration.
2. Find the Lambda function aspect for the appropriate module, typically located in source / infrastructure/spects/validations.py, and add the new runtime version to the supported runtime list.

Problem: Multiple ProvisionedVehicles active certificates

Attempting to concurrently provision multiple vehicles of the same VIN can lead to a non-valid database step for the ProvisionedVehicles DynamoDB table. Specifically, the table will have multiple ActiveCertificates for a single VIN, meaning multiple records exist in the table. To resolve this issue, delete all records for that VIN and reprovision.

Resolution

Delete all ProvisionedVehicles table records for the invalid VIN, and reprovision the vehicle.

1. Delete all entries for the ProvisionedVehicles DynamoDB table with the specified VIN, either through the console or the AWS Command Line Interface.
2. Reprovision the vehicle with one request. Reprovisioning the vehicle generates a new valid certificate, registers the vehicle with the new certificate, and creates a new record for the vehicle.
Uninstall the solution

This solution creates multiple CloudFormation deployments; both from the top level `cdk deploy` as well as additional stacks from AWS Proton and CodePipeline executions. Some resources cannot be uninstalled directly with CloudFormation and must be deleted by using the AWS Management Console or the AWS CLI.

The following commands assume that the stage is `dev`. For other stages, replace `dev` with the appropriate value.

Capture the deployment UUID

Capture and store the deployment UUID (Universally Unique ID) of the solution. This is used to look for any resources not destroyed by CloudFormation after teardown completes.

```
make get-deployment-uuid
```

The output will be a uuidv4 string:

```
XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Delete CMS modules in AWS Proton and CloudFormation

Use the following the instructions to delete the CMS on AWS modules in AWS Proton and CloudFormation that were deployed by using Backstage.

In AWS Proton, navigate to the Services view and delete any service attached to the cms-environment Environment. Wait until all services successfully delete.

**Note**

You must choose the link into the service to be able to delete it with the Actions dropdown.
If the delete fails in AWS Proton, and the CodeBuild cdk destroy task shows an error in the CodeBuild logs, most likely the module attempting to be deleted has a dependency blocking the deletion. Continue uninstalling the rest of the modules and try again. Refer to the AWS Proton CodeBuild logs and CloudFormation console output for additional information.

If the delete fails in AWS Proton, but the CloudFormation stack for the module is deleted successfully, most likely AWS Proton needs an additional role for account level CodeBuild. You will observe the following error:

```
AccountSettings.pipelineCodeBuildRoleArn has not been configured.
```

In this case, go to **Account Settings** and configure a role.

⚠️ **Warning**

AWS Proton's UI requires a GitHub repository connection to configure roles. To get around this, set the roles using the following AWS CLI command.

```
aws proton update-account-settings \
--pipeline-codebuild-role-arn arn:aws:iam::11111111111:role/cms-dev-cmsprotonenvironmentprotoncodebuildroleXXX-XXXXXX \
--pipeline-service-role-arn arn:aws:iam::11111111111:role/service-role/proton_role
```

After AWS Proton shows that all services have been deleted, verify in the CloudFormation console that all CMS on AWS modules have been deleted, and if not, delete them.
Delete AWS Proton Service Templates, Environment, and Environment Templates

Note
If you wish to keep services that have been deployed by using Backstage, skip these steps.

1. Navigate to the Service Templates view in AWS Proton and delete any CMS on AWS service templates.

   ![AWS Proton Service Templates](image)

   **Warning**
   If you receive the following Validation Exception: Service template has major versions that must first be deleted, then you must run the delete command multiple times until all of the major versions have been deleted for the service template.

2. Navigate to the Environments view and delete the cms_environment Environment.
**Warning**

If you receive the following Validation Exception: Environment template has major versions that must first be deleted, then you must run the delete command multiple times until all of the major versions have been deleted for the environment template.

2. Navigate to the **Environment Templates** view and delete the `cms_environment` Environment Template.

3. Validate on the AWS Proton dashboard that all resources have been deleted.
4. Navigate to CloudFormation and delete the AWS Proton CodeBuild stack (AWSProton-Codebuild-########).

5. Verify that the cms-environment stack was removed when uninstalling AWS Proton, and if not, delete it.

Delete the Backstage CloudFormation Stacks

1. Navigate to CloudFormation and delete the following stacks: cms-backstage-dev and cms-backstage-env-dev.
Delete the Backstage CloudFormation Stacks

84

Connected Mobility Solution on AWS
Implementation Guide

Note
The cms-backstage-dev stack might fail to delete due to the ACM certificate creation custom resource. After delete fails, navigate to DELETE AGAIN and select RETAIN on the custom resource. This will remove all resources from the account.
2. To delete the CMS Backstage module's Amazon ECR repository, navigate to Amazon ECR and delete the repository called backstage.
Delete the CMS on AWS CloudFormation stacks

Navigate to CloudFormation and delete the cms-environment and cms-dev stacks.

This is your last opportunity to capture the deployment UUID. Make sure you have captured it using the make command specified in Capture the deployment UUID.
Manually cleanup resources

Locate leftover resources using the following command, which first requires you to export the DEPLOYMENT_UUID variable using the value previously acquired from AWS Systems Manager. If you uninstalled the CMS on AWS stack without capturing the UUID, you can run the following command by removing the Solutions:DeploymentUUID Key filter. However the results will include other CMS on AWS stacks if they exist, so use this method with caution.

```bash
export DEPLOYMENT_UUID=<DEPLOYMENT_UUID_VALUE_FROM_SSM>
aws resourcegroupstaggingapi get-resources --tag-filters \
  Key=Solutions:SolutionID,Values=SO0241 \
  Key=Solutions:DeploymentUUID,Values=$DEPLOYMENT_UUID \
  --query "ResourceTagMappingList[*.ResourceARN"
```

This query results in a list of ARNs to assist you with locating the resources in the AWS Management Console. Resources can then be manually deleted, or deleted via a script, using the resource ARNs where appropriate.
**Warning**

Some resources may take some time to clean up after CloudFormation finishes tearing down, and could show in the output even if they no longer exist. For example, Amazon VPC, Fargate, and Amazon ECS resources can remain queryable for up to 30 minutes after deletion.

---

**Deleting the Amazon S3 buckets**

This solution is configured to retain the solution-created Amazon S3 buckets if you decide to delete the AWS CloudFormation stack to prevent accidental data loss. After uninstalling the solution, you can manually delete these Amazon S3 buckets if you don’t need to retain the data. Follow these steps to delete an Amazon S3 bucket.

1. Sign in to the Amazon S3 console.
2. Choose **Buckets** from the left navigation pane.
3. Locate the buckets created by the solution.
4. Select a bucket.
5. Choose **Delete**.

To delete the Amazon S3 bucket using AWS CLI, run the following command:

```
$ aws s3 rb s3://<bucket-name> --force
```

---

**Deleting the Amazon DynamoDB tables**

This solution is configured to retain the DynamoDB tables if you decide to delete the CloudFormation stack to prevent accidental data loss. After uninstalling the solution, you can manually delete the DynamoDB tables if you do not need to retain the data. Follow these steps:

1. Sign in to the [Amazon DynamoDB console](#).
2. Choose **Tables** from the left navigation pane.
3. Locate the tables created by the solution.
4. Select a CMS on AWS table.
5. Choose **Delete**.

6. Repeat the steps until you have deleted all of the solution's tables.

To delete the DynamoDB tables using AWS CLI, run the following command:

```bash
$ aws dynamodb delete-table <table-name>
```

### Deleting the Amazon CloudWatch logs

This solution retains the CloudWatch Logs if you decide to delete the CloudFormation stack to prevent against accidental data loss. After uninstalling the solution, you can manually delete the logs if you do not need to retain the data. Follow these steps to delete the CloudWatch Logs.

1. Sign in to the [Amazon CloudWatch console](https://console.aws.amazon.com/cloudwatch/home).
2. Choose **Log Groups** from the left navigation pane.
3. Locate the log groups created by the solution.
4. Select one of the log groups.
5. Choose **Actions** - **Delete**.

Repeat the steps until you have deleted all the solution's log groups.

To delete the CloudWatch log groups using AWS CLI, run the following command:

```bash
$ aws logs delete-log-group --log-group-name <log-group-name>
```

### Deleting the AWS KMS customer managed keys

This solution retains the AWS KMS customer managed keys if you decide to delete the CloudFormation stack to prevent against accidental encrypted data loss. After uninstalling the solution, you can manually delete the keys if you do not need to use them again. Follow these steps to delete the AWS KMS keys.

2. Choose **Customer managed keys** from the left navigation pane.
3. Locate the keys created by the solution.
4. Select one of the keys.
5. Choose Key actions - Schedule key deletion.
6. Optionally edit the **Waiting period (in days)** value.
7. Select Confirmation.
8. Choose Schedule deletion.

Repeat the steps until you have deleted all the solution's customer managed keys.

To delete the AWS KMS customer managed keys using AWS CLI, run the following command:

```
$ aws kms schedule-key-deletion --key-id <key-id-or-arn>
```

### Deleting the Amazon Cognito user pools

This solution retains the Amazon Cognito user pools if you decide to delete the CloudFormation stack to prevent against accidental user data loss. After uninstalling the solution, you can manually delete the user pools if you do not need to retain the users. Follow these steps to delete the user pools.

1. Sign in to the [Amazon Cognito console](https://console.aws.amazon.com/cognito/home).
2. Choose **User pools** from the left navigation pane.
3. Locate the user pools created by the solution.
4. Select one of the user pools.
5. Choose **Delete**.
6. Select Deactivate deletion protection.
7. Enter the user pool name in the second field.
8. Choose **Delete**.

Repeat the steps until you have deleted all the solution's user pools.

To delete the Amazon Cognito user pool using AWS CLI, run the following command:

```
$ aws cognito-idp delete-user-pool --user-pool-id <user-pool-id>
```
Dealing the Amazon Relational Database Service Snapshots

This solution retains the Amazon Relational Database Service (Amazon RDS) snapshots if you decide to delete the AWS CloudFormation stack to prevent against accidental data loss. After uninstalling the solution, you can manually delete the snapshots if you do not need to retain the data. Follow these steps to delete the snapshots.

1. Sign in to the Amazon RDS console.
2. Choose Snapshots from the left navigation pane.
3. Locate the snapshots created by the solution.
4. Select one of the snapshots.
5. Choose Actions – Delete snapshot.
6. Choose Delete.

Repeat the steps until you have deleted all the solution's snapshots.

To delete the Amazon RDS snapshot using AWS CLI, run the following command:

```
$ aws rds delete-db-snapshot --db-snapshot-identifier <db-snapshot-identifier>
```
Developer guide

Source code

Visit our GitHub repository to download the source files for this solution and to share your customizations with others.

The AWS Cloud Development Kit (AWS CDK) generates the Connected Mobility Solution on AWS templates. See the README.md file for additional information.
Reference

This section includes a list of builders who contributed to this solution.

Anonymized data collection

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When activated, the following information is collected and sent to AWS:

- Solution ID - The AWS solution identifier
- Unique ID (UUID) - Randomly generated, unique identifier for each deployment
- Resource Utilization - Total storage utilized by the solution
- API Usage - Usage amount of API calls to the solution's public facing API

AWS owns the data gathered through this survey. Data collection is subject to the Privacy Notice. To opt out of this feature, complete the following steps before launching the AWS CloudFormation template.

1. In the source/infrastructure/stacks/cms_stack.py file, replace the send_anonymous_usage_condition value with “No”.
2. Continue with the deployment as specified in the Deploy section.

Contributors

- Narmeen Ali
- Kevin Hargita
- Guru Koushik Senthil Kumar
- Anthony McIntosh
- Harshit Patel
- Alex Sansone
- Saif Shaikh
• Matt Wise
• Dmitry Zavyalov
# Revisions

<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 2023</td>
<td>Initial release</td>
</tr>
<tr>
<td>December 2023</td>
<td>v1.0.1:</td>
</tr>
<tr>
<td></td>
<td>• Resolved an issue where the Amazon Aurora PostgreSQL cluster’s version defaulted to 11 instead of 13 in some Regions.</td>
</tr>
<tr>
<td></td>
<td>• Pinned Node and Python versions in the Proton manifest.yml file for every module.</td>
</tr>
<tr>
<td></td>
<td>• Updated cost table for Use case 2.</td>
</tr>
<tr>
<td></td>
<td>• Corrected URLs in the README.md file.</td>
</tr>
<tr>
<td></td>
<td>For more information, refer to the CHANGELOG.md file in the GitHub repository.</td>
</tr>
<tr>
<td>January 2024</td>
<td>v1.0.2:</td>
</tr>
<tr>
<td></td>
<td>• Updated Grafana workspace in EV Battery Health module to include plugin management and install Amazon Athena plugin.</td>
</tr>
<tr>
<td></td>
<td>• Removed yarn tsc:full from backstage image build.</td>
</tr>
<tr>
<td></td>
<td>• Added octokit and resolution for follow-re directs to mitigate vulnerabilities.</td>
</tr>
<tr>
<td></td>
<td>• Added ignore pattern for Axios in vehicle simulator.</td>
</tr>
<tr>
<td></td>
<td>For more information, refer to the CHANGELOG.md file in the GitHub repository.</td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>February 2024</td>
<td>v1.0.3:</td>
</tr>
<tr>
<td></td>
<td>• Added resolutions to ECDSA, cryptography, and node-ip packages to mitigate vulnerabilities.</td>
</tr>
<tr>
<td></td>
<td>For more information, refer to the <a href="#">CHANGELOG.md</a> file in the GitHub repository.</td>
</tr>
<tr>
<td>February 2024</td>
<td>v1.0.4:</td>
</tr>
<tr>
<td></td>
<td>• Upgraded Backstage to 1.23.3 to mitigate vulnerability.</td>
</tr>
<tr>
<td></td>
<td>• Fixed a bug that could occur if the S3 version of the Backstage source was prefixed with a special character.</td>
</tr>
<tr>
<td></td>
<td>For more information, refer to the <a href="#">CHANGELOG.md</a> file in the GitHub repository.</td>
</tr>
</tbody>
</table>
Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. AWS responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Connected Mobility Solution on AWS is licensed under the terms of the Apache License Version 2.0 available at The Apache Software Foundation.