Implementation Guide

Connected Mobility Solution on AWS

Connected Mobility Solution on AWS: Implementation Guide

Table of Contents

Accelerate development and deployment of connected vehicle assets

Publication date: *October 2023 (last update: April 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 take advantage of provided features as modules, deploy and manage these modules from a centralized platform, and implement and integrate your own custom modules into the solution.

Provided features allow 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, creating profiles for provisioned vehicles.
- Capture, store, and query telemetry data emitted from provisioned vehicles.
- Create alerts for this data, and subscribe to notifications based on data thresholds.
- Visualize vehicle telemetry data through an Amazon Managed Grafana 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, as well as integrate custom module implementations.

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.

The intended audience for using this solution's features and capabilities in their environment includes solution architects, business decision makers, DevOps engineers, data scientists, and cloud professionals.

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

If you want to . . .	Read . . .
Know the cost for running this solution. The estimated cost for running this solution in the us-east-1 Region is USD $352.80 per month.	Cost
Understand the security considerations for this solution.	Security
Know how to plan for quotas for this solution.	Quotas
Know which AWS Regions are supported for this solution.	Supported AWS Regions
View or download the AWS CloudForm ation template included in this solution to automatically deploy the infrastructure resources (the "stack") for this solution.	AWS CloudFormation templates

If you want to . . .	Read . . .
Access the source code and optionally use the AWS Cloud Development Kit (AWS CDK) (AWS CDK) to deploy the solution.	GitHub repository

Features and benefits

The solution provides the following features:

Modularity

CMS on AWS simplifies maintaining and implementing the solution by utilizing a modular design. The first of two significant benefits of modularity is separating concerns and outlining clear boundaries between functional sets so that the solution flexibly meets connected vehicle system demands. The second benefit is allowing interchanging CMS on AWS modules with bespoke implementations, which meet the boundary definitions defined by CMS on AWS standards, as well as implementing additional modules into the solution. These custom modules can then be deployed from the same portal (ACDP / Backstage) as CMS on AWS provided modules.

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, AWS CodeBuild, and AWS CloudFormation to deploy and manage CMS on AWS modules. It uses well-defined templates to allow for implementations of custom modules to be integrated into the solution and deployed from the same portal.

Configuration

CMS on AWS provides a mechanism to associate deployments of CMS on AWS modules with a unique ID. Modules sharing the same unique ID also share infrastructure such as VPC and identity provider (IdP). Other CMS on AWS modules deployed with the unique ID can use the configuration module to lookup VPC and IdP configurations. Anonymized metrics collection is handled by the configuration module which reports metrics based on Amazon Simple Storage Service (Amazon S3) and AWS Timestream usage.

OpenID Connect authentication

CMS on AWS provides a default deployment of authentication infrastructure through Amazon Cognito or the ability to integrate with customer's own IdP services. The IdP chosen can be used for both users and CMS on AWS services to authenticate by retrieving access tokens and JSON Web Tokens (JWTs). CMS on AWS then provides the means to validate the integrity of access tokens with the chosen IdP.

Network security

CMS on AWS provides a default VPC or the ability to bring your own VPC to secure the AWS resources deployed in your account. The default VPC provides public, private and isolated subnets in two Availability Zones (AZs). This enables a highly available and secure cloud network. Modules then use these subnets to manage security accordingly.

Vehicle provisioning

CMS on AWS registers vehicles as AWS IoT Core things to help you 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 the 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 S3. The solution ingests data from pre-defined Message Queuing Telemetry Transport (MQTT) topics, and stores it in both JSON and Parquet data formats. The storage is integrated with an alerts mechanism which utilizes Amazon Simple Notification Service (Amazon SNS).

API

CMS on AWS provides the ability to query vehicle data stored within the solution for use in other CMS 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 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 Electric Vehicle (EV) battery telemetry data and configure alerts based on data thresholds. This is done through an Amazon Managed Grafana. 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 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

Deployment and management

Fleet owners and deployment managers can select, configure, and deploy CMS on AWS modules through the centralized Automotive Cloud Developer Portal (ACDP). From here, deployments can also be monitored and tore down. Custom module implementations can be integrated into Backstage, the front-end component of the ACDP, and managed in the same manner. A centralized platform allows for managing and monitoring your deployment easily and effectively, as well as separate ownership and permissions between members of your organization.

Vehicle connection

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 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 vehicles

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 specific to this solution:

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 via the ACDP. It also allows for integration with bespoke customer modules for a centralized deployment mechanism.

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.

identity provider

CMS on AWS provides the ability to integrate the authentication system with any third-party identity provider which is OAuth2.0 OpenID Connect compliant. This identity provider should provide some concept of users, a resource server (audience), domain (issuer), and clients for users and services.

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.

VPC / CMS on AWS VPC

A virtual private cloud (VPC) is an AWS service. Within the context of our solution, VPC is often referring to the specific VPC instance chosen to be deployed with the solution. This is either the default CMS on AWS VPC, a preconfigured VPC provided by CMS on AWS, or a customer provided VPC. Any of these can then be attached to this solution's configuration. In all cases, the VPC is deployed prior to CMS on AWS and is provided to subsequent module deployments for network security. All modules which can use a VPC utilize the same configured VPC.

> **Note**
>
> For a general reference of AWS terms, see the AWS Glossary.

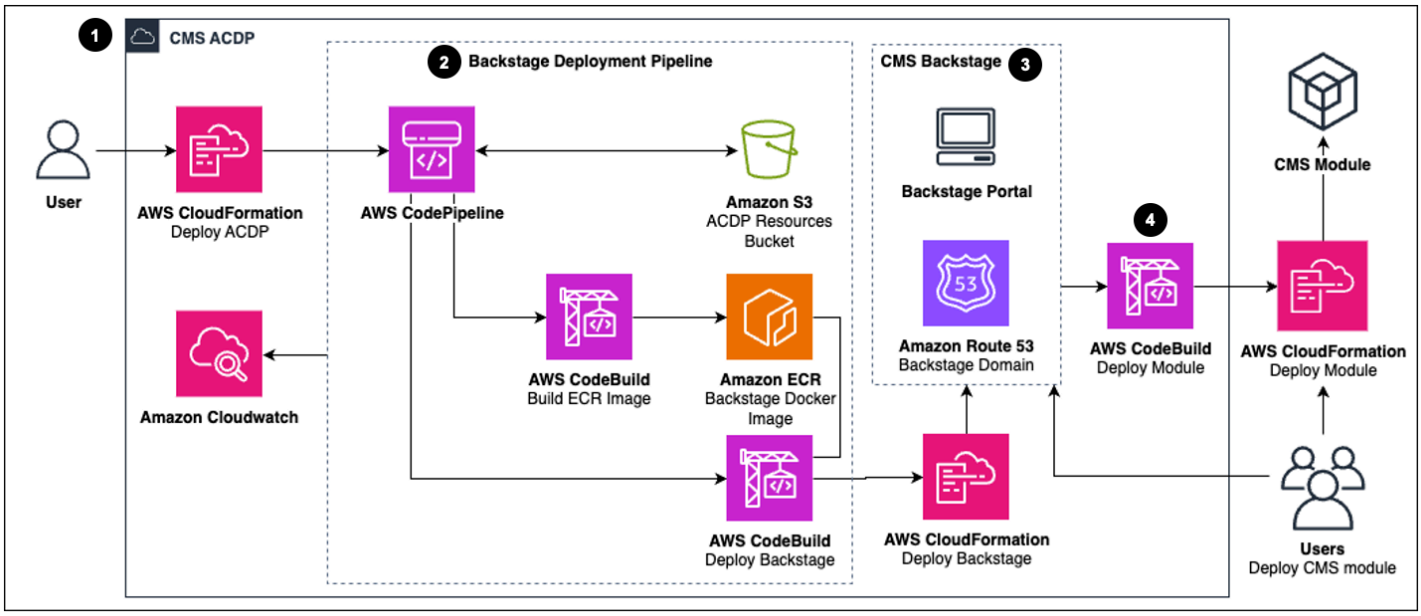
Architecture overview

This section provides a high-level description of all CMS on AWS provided modules. It also includes architecture diagrams for the deployment of the Automotive Cloud Developer Portal and the deployment of subsequent CMS on AWS modules.

Outside of deployment resources and necessary configuration modules, each installation of CMS on AWS is unique based on which provided modules you deploy, and which bespoke module implementations you integrate. The deployment architecture described in this section is consistent across deployments of CMS on AWS.

Automotive Cloud Developer Portal deployment

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



Automotive Cloud Developer Portal deployment architecture

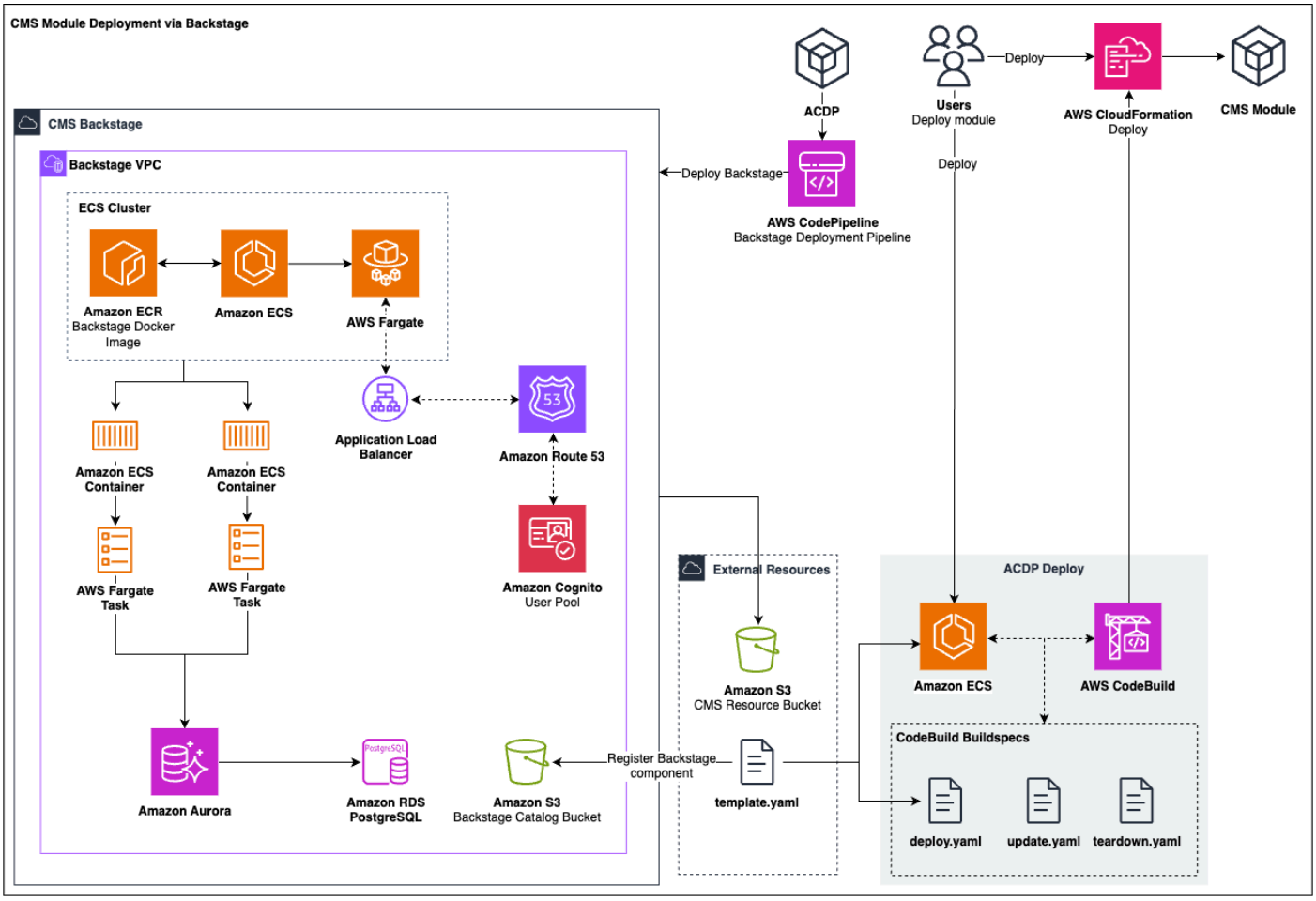
> **Note**
>
> CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.

The high-level process flow for the solution components deployed with AWS CDK is as follows (from left to right in the diagram):

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 templates created by AWS CDK. The ACDP creates a Backstage deployment pipeline. The pipeline then deploys the Backstage module.
2. **Backstage Deployment Pipeline** – The ACDP deploys a CI/CD pipeline through AWS CodePipeline that configures all the steps necessary to deploy the Backstage module. This is accomplished with AWS CodeBuild pipeline projects, which use build specification files to define their actions. Amazon Elastic Container Registry (Amazon ECR) is used to store the Backstage Docker image. For details on the structure of the pipeline and each build step, see the Backstage module.
3. **Backstage** – The Backstage module is the presentation layer for the ACDP. An Elastic Load Balancing (ELB) Application Load Balancer connects with Amazon Route 53 and an Amazon Elastic Container Service (Amazon ECS) cluster group setup with AWS Fargate tasks. The Backstage module allows deploying CMS on AWS modules through a graphical user interface. For more information, see Backstage module.
4. **Deploying CMS on AWS modules via Backstage** – When the ACDP is configured, there are two ways to deploy CMS on AWS modules (see Deploy the solution for details):
   - Backstage – By using ACDP and Backstage, you can deploy CMS on AWS modules using module templates configured for Backstage, powered by AWS CodeBuild.
   - AWS CDK CLI – Without Backstage, you can individually deploy the CMS on AWS modules directly from the repository by utilizing the `make deploy` target.

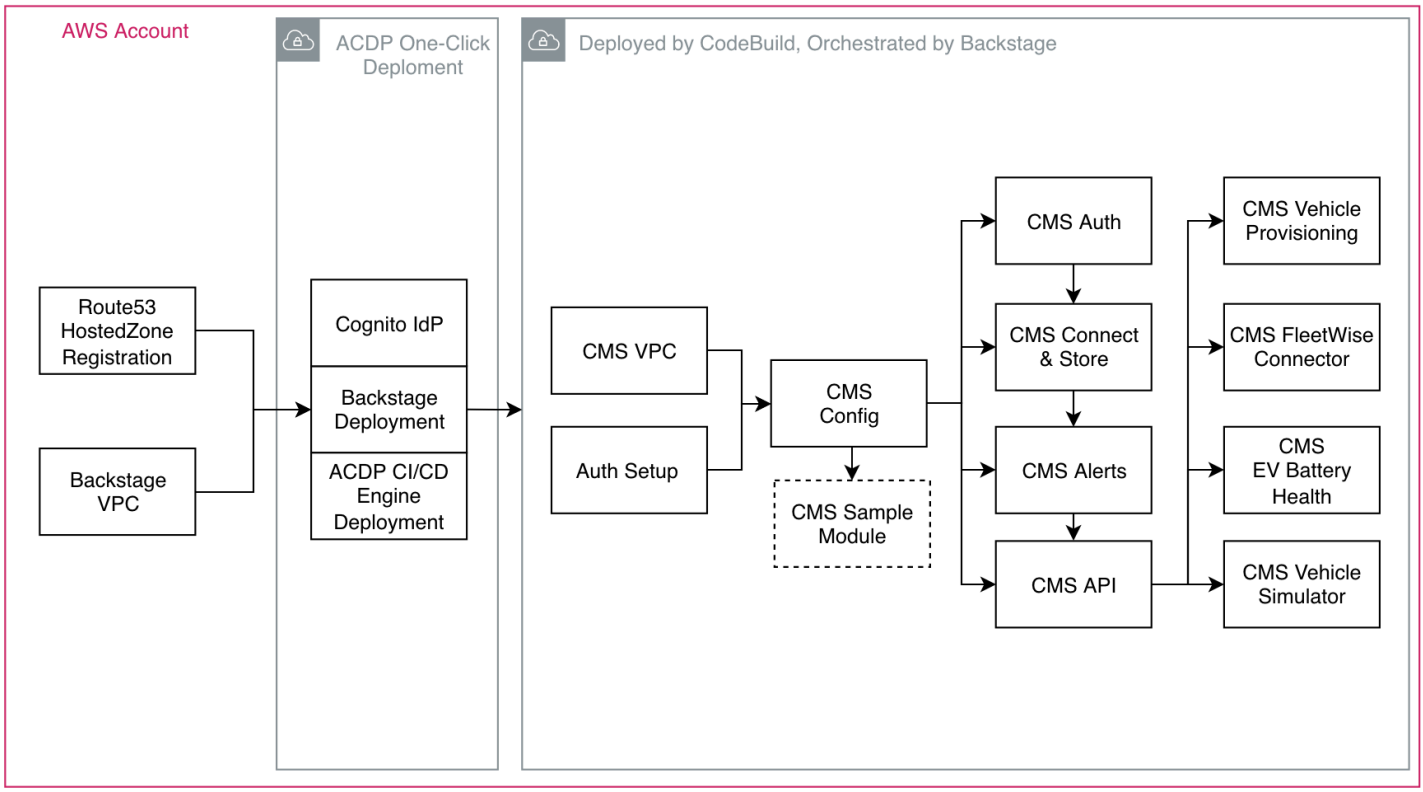
Deploying CMS on AWS modules via Backstage

The AWS CodePipeline deployment of Backstage initiates from the ACDP deployment described above allowing for Backstage to deploy additional CMS on AWS modules. Deploying CMS on AWS modules using Backstage is represented by the following diagram:



Deploying CMS on AWS modules via Backstage

All CMS on AWS Modules are deployable via Backstage templates. These deployments are powered by AWS CodeBuild. Some modules have deploy-time dependency on other modules and therefore should be deployed in the order of those relationships (shown below).



CMS on AWS module deployment order

CMS on AWS modules and services

The high-level architectural descriptions for the CMS on AWS modules and services are as follows:

Amazon Virtual Private Cloud (Amazon VPC)

Amazon Virtual Private Cloud (Amazon VPC) is an AWS service that allows you to launch AWS resources inside a logically isolated virtual network. CMS on AWS provides a VPC module that deploys an opinionated network configuration. For more details, see Virtual Private Cloud and Amazon VPC.

Auth Setup

The Auth Setup module provides the means to configure a third-party OpenID Connect (OIDC) compliant IdP of your choice for use with CMS on AWS. A default deployment of AWS Cognito infrastructure to serve as the IdP is also provided. The Auth Setup module will either deploy configurations with an expected JSON structure with the required IdP values, or the option is given

to use existing Secrets Manager secrets. Either will fully configure the authentication parameters required for CMS on AWS deployment's authentication. For more information, see Auth Setup.

Config

The CMS Config module uses the AWS Systems Manager Parameter Store to register a unique ID which serves as a namespace to deploy other CMS on AWS modules. The CMS Config module takes the VPC name and Identity Provider ID as additional inputs which are shared with the other CMS on AWS modules deployed with the same unique ID as the CMS Config module. The module uses an AWS Lambda function to send anonymized metrics about AWS S3 and AWS Timestream resource usage. The module also implements an AWS Lambda function for AWS SSM Parameter resource lookup based on the unique ID.

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 Backstage module as its presentation layer to provide a configurable developer platform for managing and monitoring the deployment of CMS on AWS modules and customer provided modules. For more details, see Automotive Cloud Developer Portal and Backstage module.

Auth

The CMS Auth module allows for the authentication and authorization of users and services throughout the solution. The module provides two AWS Lambda functions which can integrate with any third-party identity provider (IdP) that is OAuth2.0 OIDC compliant. This is done by communicating with the Auth Setup module's IdP configurations, exposed as Secrets Manager secrets. Of these two lambda functions, one facilitates exchanging an authorization code for an access token via the authorization code flow, and the other validates and authorizes access tokens. For more details, see Auth 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 invoke rules 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.

FleetWise Connector

The CMS FleetWise Connector module allows you to consume data that is captured by AWS IoT FleetWise campaigns. This is done by querying Amazon Timestream to migrate data into the CMS on AWS Connect & Store module's telemetry bucket. The data is then indexed using AWS Glue, and made accessible via Amazon Athena.

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.

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 Backstage module, as well as supporting the further managed deployment of CMS on AWS modules via AWS CodeBuild. CMS on AWS sends logging and metrics to Amazon CloudWatch throughout the entire solution. A default log retention of three months is used in most places; this can be customized by altering the CDK (look for `aws_logs.RetentionDays`) and rebuilding the solution. The infrastructure is managed and operated by AWS CDK, with deployment assets stored in Amazon S3 for use with Backstage.

Security

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

To ensure network security, CMS on AWS network traffic only flows through the internet when necessary. The traffic flows between AWS services via the VPC network and VPC endpoints when possible. Simultaneously, all internet accessible endpoints are protected by authentication (OAuth2.0) via the customer chosen identity provider, expecting the configured issuer, audience, clients, and scopes via access tokens. 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 (a notable exception being Backstage), 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 to minimize needed compute resources and prevent bottlenecks. 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

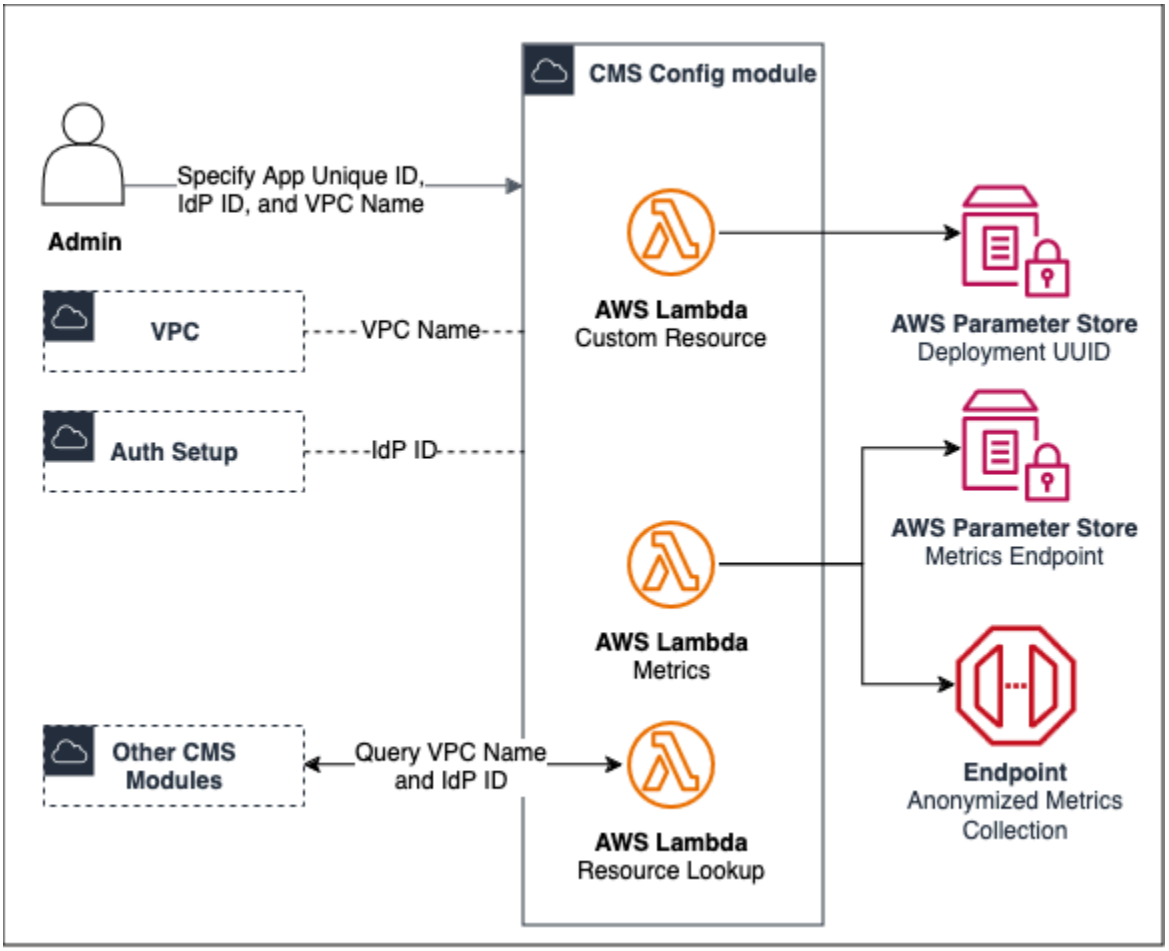
This section describes how we architected this solution using the principles and best practices of the sustainability pillar.

This solution uses primarily 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.

Config



Config module architecture

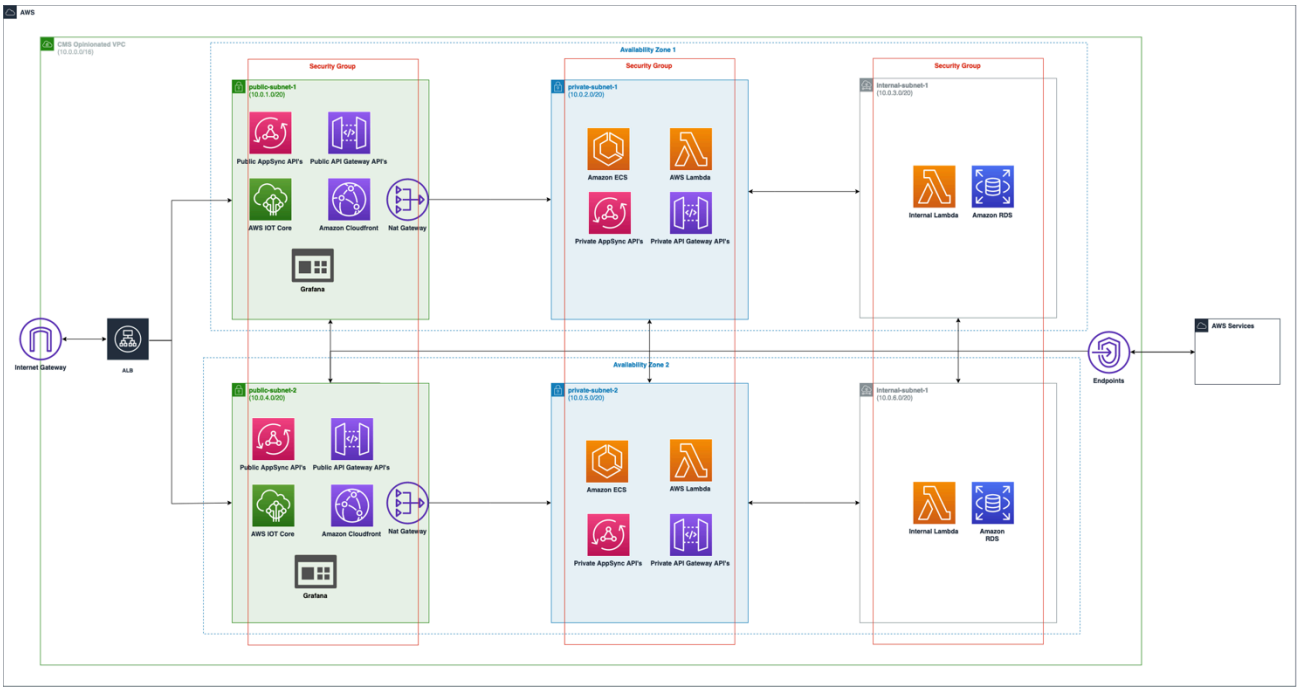
The Config module is a prerequisite deployment required for the deployment of any CMS on AWS module (all modules except ACDP, VPC, and Auth Setup). The Config module introduces the concept of an App Unique ID, which serves as a unique namespace for all CMS on AWS modules. This module enables sharing the same VPC and IdP among multiple CMS on AWS modules without requiring input of the VPC and IdP configuration in each module's deployment. The App Unique ID serves the following purposes:

- A unique naming prefix for all AWS CloudFormation resources that avoids conflicts when deploying multiple instances of the same CMS on AWS module in the same region and account
- A unique naming prefix for easy lookup of AWS SSM Parameters
- Registering deployment of CMS on AWS modules associated with the App Unique ID, preventing multiple deployments of the same module within the same CMS on AWS deployment

The CMS Config module also provides AWS Lambda functions that perform the following functions:

- Create a deployment UUID which is used to tag all CloudFormation resources created by CMS on AWS modules
- Send anonymized metrics related to Amazon S3 and AWS Timestream
- Lookup VPC name and Identity Provider ID using the App Unique ID

Virtual Private Cloud (VPC)



VPC module architecture

CMS on AWS requires a VPC to deploy other modules including ACDP. Users can bring their own VPC or deploy the CMS on AWS VPC module. The VPC module provides a preconfigured VPC with the architecture shown in the diagram. It has the following configuration:

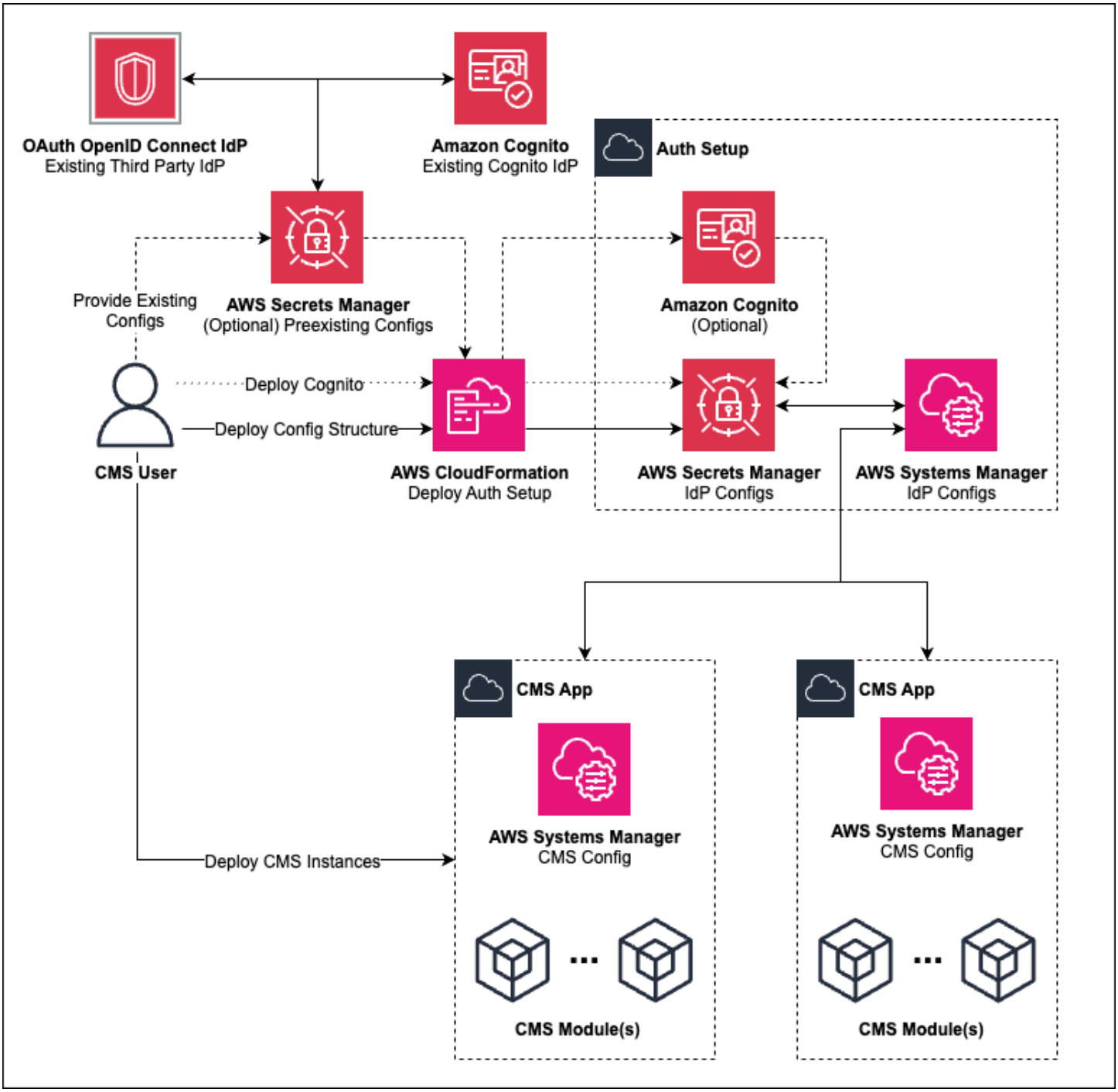
- Two Availability Zones (AZ)
- VPC endpoints for most of the AWS services that support AWS PrivateLink
- One NAT gateway in each AZ
- Public, private, and isolated subnets in each AZ
- One internet gateway

Having two AZs and two NAT Gateways ensures that all the services are still functioning in case there is a failure in one of the two AZs. The three subnets in each AZ are meant to be used by modules to allocate resources as needed.

- Public subnet for resources that require inbound and outbound connection to the internet via internet gateway
- Private subnet for resources that require an outbound only connection to the internet via NAT gateway
- Isolated subnet for resources that do not require any inbound or outbound connection to the internet

Resources should be assigned a security group at the module level depending on their needs.

Auth Setup



Auth Setup module architecture

The Auth Setup module creates Secrets Manager secrets, which provide the configuration information necessary for the CMS Auth module to communicate with any OIDC compliant IdP.

This provides the ability to deploy CMS on AWS using your IdP. The IdP must be configured with users, clients, a domain (issuer), and a resource server (audience).

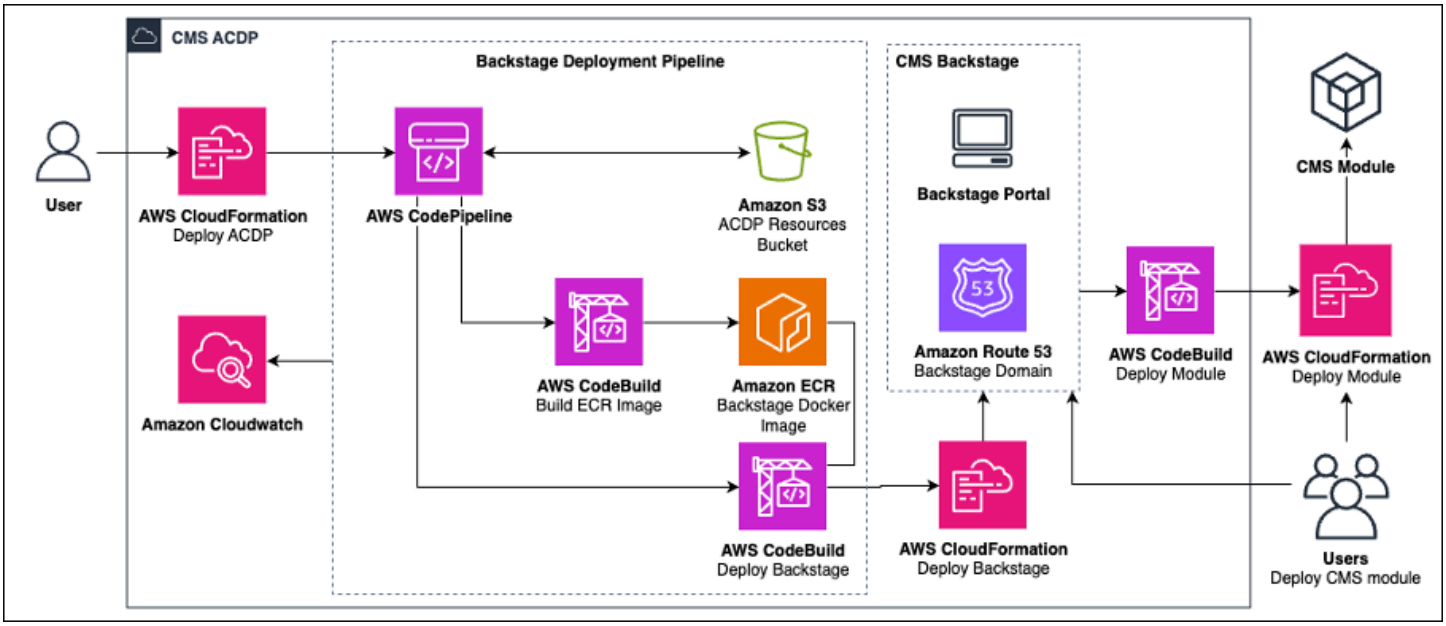
This module also provides an optional deployment of Amazon Cognito infrastructure that is pre-configured to serve as the IdP for the CMS on AWS deployment.

Each configuration secret is a known JSON structure with consistent key/value pairs expected by CMS on AWS modules. The key structure is consistent regardless of deployment path, but the values are populated dynamically. The three deployment paths are as follows:

- **Deploy Amazon Cognito infrastructure** – If choosing to deploy Amazon Cognito infrastructure, the configuration JSONs will be pre-configured with values from the newly deployed Amazon Cognito resources.
- **Provide existing configs** – During deployment, you are given the option to provide ARNs for zero to all of the configuration secrets. If providing an existing secret ARN, a new secret will not be created and no validation of the secret value's structure will be performed. The existing secret will be configured to be used by the CMS on AWS deployment.
- **Deploy config structure** – During deployment, you are given the option to provide ARNs for zero to all of the configuration secrets. If not providing an existing secret ARN, a new secret will be created with the expected JSON structure. The values of this JSON structure will be empty however and will need to be manually set for the CMS on AWS deployment's authentication to function properly.

Since the choice to provide an existing secret ARN is individual to each config, the latter two deployment paths can be combined in a single deployment for differing secrets.

Automotive Cloud Developer Portal



Automotive Cloud Developer Portal architecture

The initial deployment of CMS on AWS includes the ACDP and Backstage, which assist in managing the deployment of CMS on AWS modules. The ACDP provides a presentation layer through the Backstage module, which you can use to select, configure, and deploy individual modules. Backstage also allows for monitoring and teardown of these modules, as well as viewing of module documentation. Backstage integrates with AWS CodeBuild for deployment execution. The ACDP is a powerful platform that enables flexibility and quick insights into deployed infrastructure.

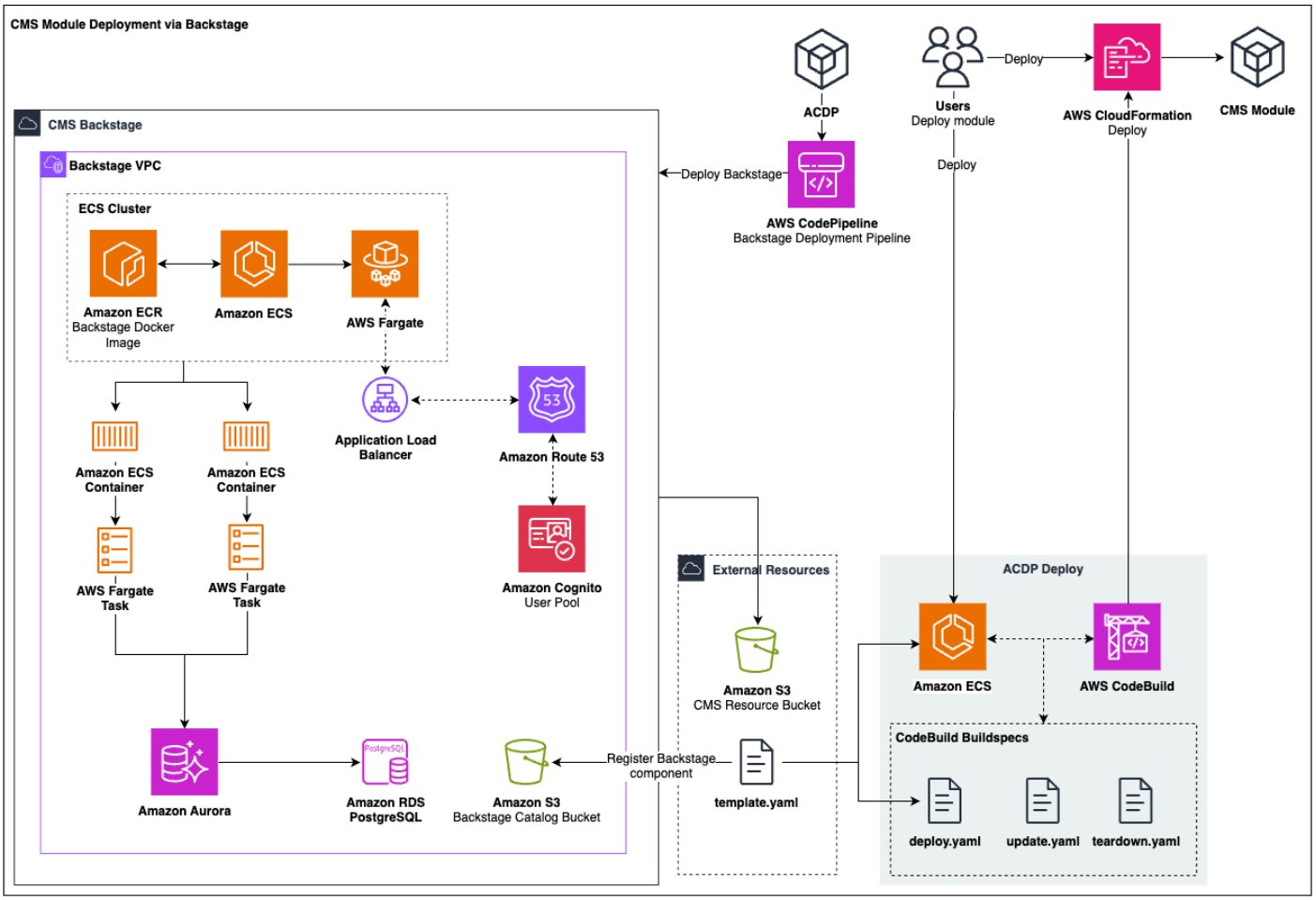
Deploying the ACDP creates and configures an AWS CodePipeline for deploying the Backstage module. The pipeline is configured with the necessary environment to carry out the Backstage deployment. Values for this configuration are taken from the local environment of the ACDP deploy, which is configured via Make. An Amazon ECR repository is created and permissions are shared between it and the Backstage pipeline.

The pipeline then handles the deployment of the Backstage module in three stages. These stages are abstracted from the previous diagram:

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

Once deployed, you can then use the Backstage module to deploy CMS on AWS modules.

Backstage module



Backstage module architecture

CMS on AWS uses Backstage as the preferred approach for deploying its modules. The Backstage module is deployed and configured to deploy within the provided VPC and requires an Amazon Route 53 hosted zone. This hosted zone is specified in the deployment parameters as the host for the Backstage portal (see Create an Amazon Route 53 Hosted Zone). CMS on AWS modules can then be deployed directly from the Backstage portal (see Deploy CMS Modules via Backstage). When deploying the Backstage module, an Amazon Cognito user pool is created with an initial user to enable authorization to the Backstage console supported by the Amazon Route 53 hosted zone.

The ACDP module contains two main parts: the AWS Infrastructure required by the Backstage deployment pipeline, and a Pipeline to build and deploy a copy of Backstage in the specified account. The infrastructure deployed by the pipeline includes an Amazon ECS Fargate job to run

Backstage's Docker image and Aurora PostgreSQL-Compatible Edition database, an Amazon S3 bucket that acts as the Backstage catalog, the Amazon Route 53 domain setup, and an Amazon Cognito authentication setup. This infrastructure makes up the Backstage module.

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:

- **ELB** – 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.

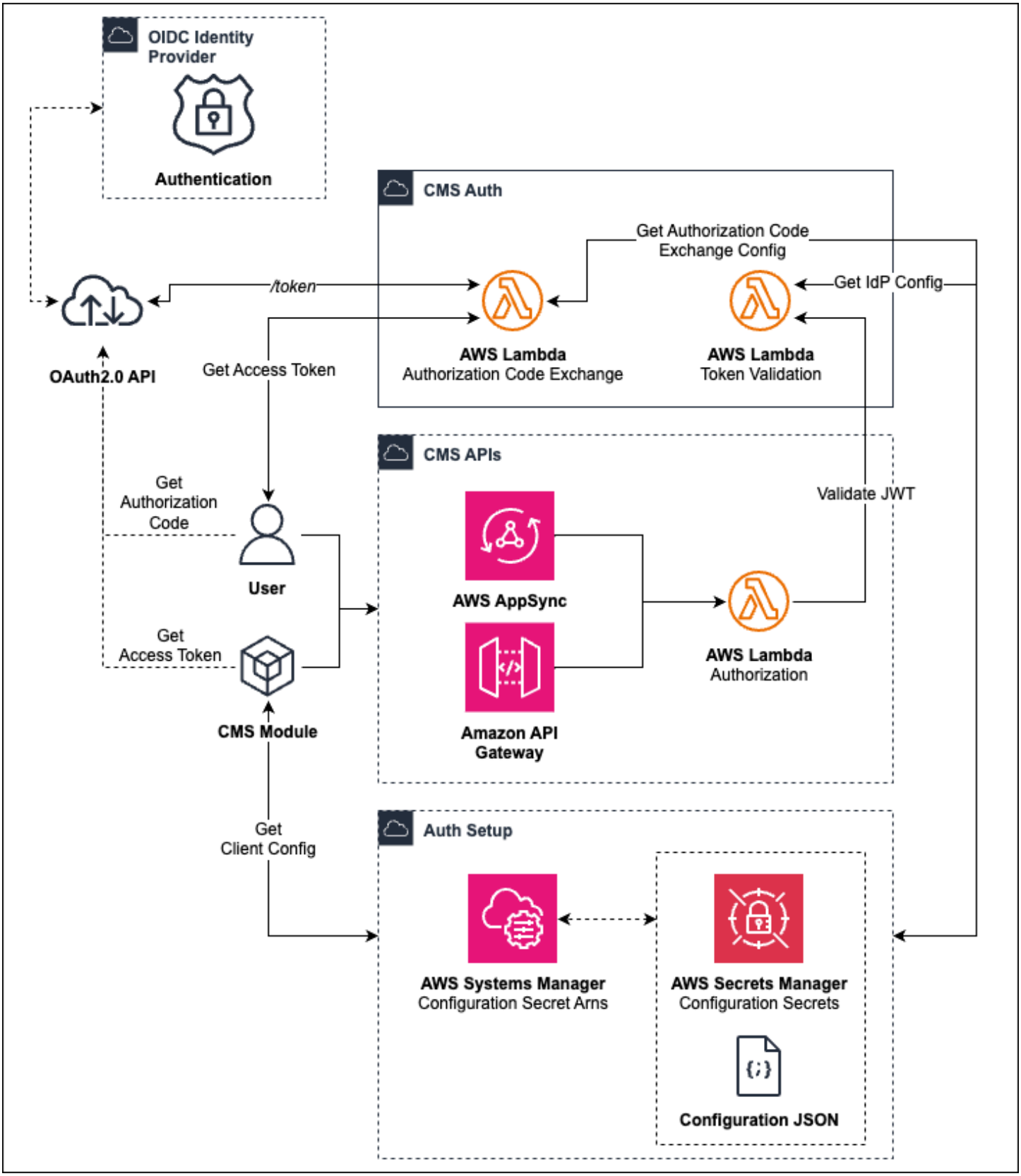
The Backstage module is integrated with AWS CodeBuild to enable its deployment functionality. The module deploys with pre-registered Backstage templates for each of the CMS on AWS modules. Deploying a CMS on AWS module through the Backstage UI requires providing required parameters, and in some cases, deploying the modules in the right order as shown in the deployment diagram. When a Backstage component is deployed, it uses the `template.yaml` file to instruct the process and performs the following steps:

1. Copy required deployment assets and docs to the Amazon S3 catalog bucket.
2. Write the module's catalog info to the Amazon S3 catalog bucket.
3. Register the module within the Backstage module's catalog.
4. Configure the ACDP deployment backend to be able to deploy the module.

After creating the `catalog-info.yaml`, deployment progress can be monitored on the CI/CD tab of the Backstage Catalog Item.

Lastly, the Backstage module can be used to view documentation related to each module directly from within the portal. These docs are included in the assets that are built and uploaded to Amazon S3 for use by Backstage.

Auth module


OIDC Identity Provider
Authentication
CMS Auth
Get Authorization Code Exchange Config
/token
Get IdP Config
OAuth2.0 API
Get Access Token
AWS Lambda
Authorization Code Exchange
AWS Lambda
Token Validation
CMS APIs
Get Authorization Code
User
Validate JWT
AWS AppSync
Get Access Token
AWS Lambda
Authorization
CMS Module
Amazon API Gateway
Get Client Config
Auth Setup
AWS Systems Manager
Configuration Secret Arns
AWS Secrets Manager
Configuration Secrets
Configuration JSON

Auth module architecture

The Auth module provides the means for CMS on AWS users and services to authenticate and authorize themselves for use with CMS on AWS APIs and portals. Users signed up with the identity provider can request authorization codes or access tokens from the authorization server via authorization code flow, or implicit flow. Services can request access tokens from the authorization server via client-credentials flow. In either case, the access token will then be validated by the identity provider resource server to grant access to CMS on AWS APIs. Access tokens can also be used to grant access to bespoke implementations of CMS on AWS front-end portals.

These authentication flows and functionality are supported via two Lambda functions deployed by the Auth module: the `token validation` and `token exchange` Lambda functions.

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

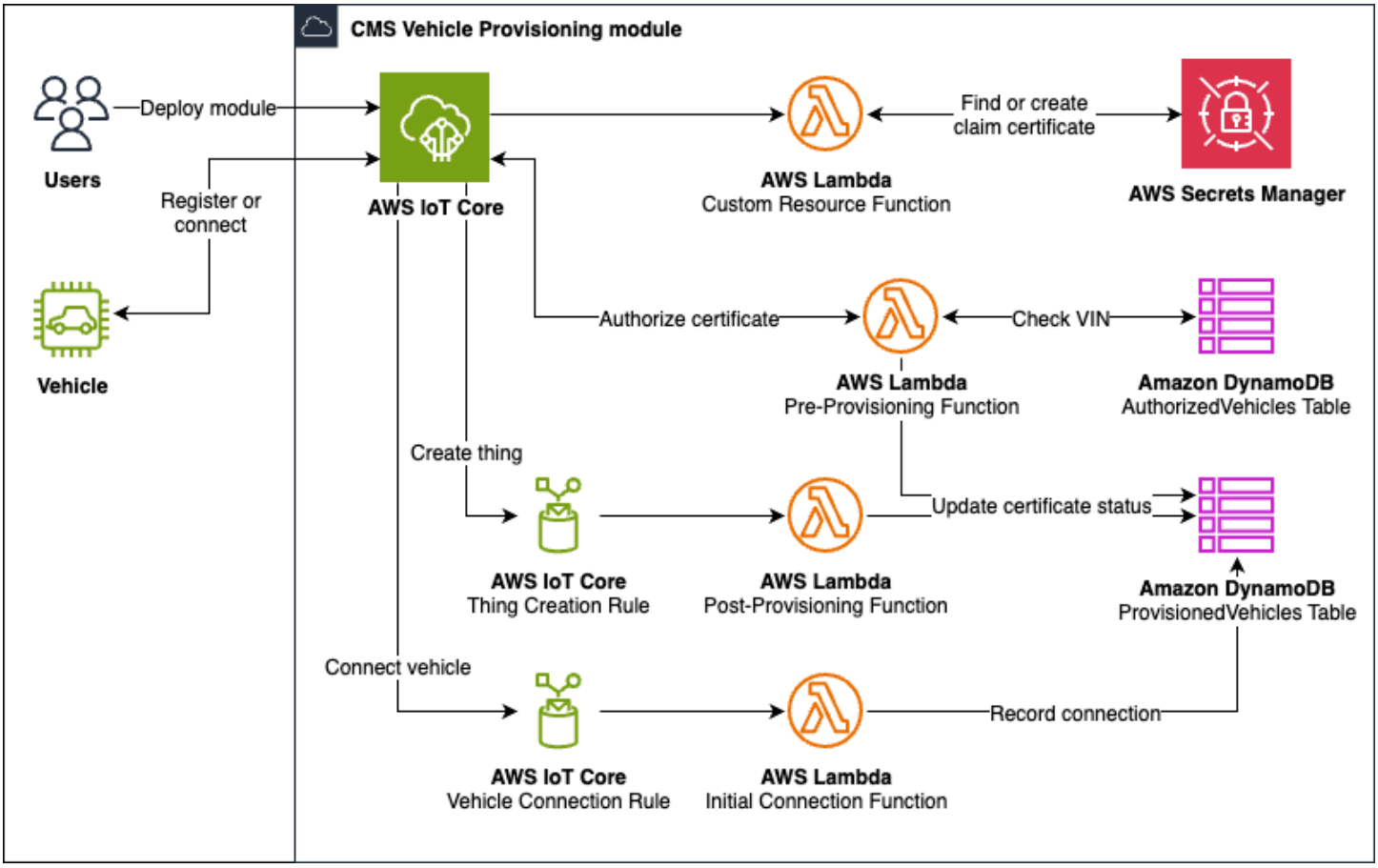
1. Validate the integrity of the JWT signature.
2. Check the token expiration.
3. Authorize the token's claims and scope against the identity provider configuration provided by the Auth Setup module.

> **Note**
>
> The details of this process are abstracted from the diagram above.

The `token exchange` Lambda function implements an authorization code exchange, defined by the authorization code flow, to retrieve an access token from the `/token` endpoint. The access tokens can then be used to authenticate and authorize users against CMS on AWS APIs. The token exchange Lambda function communicates with the authorization server, specified in the identity provider config secret, to verify the integrity of the authorization code against the associated identity provider.

The `token exchange` Lambda function allows for usage of an optional Proof Key for Code Exchange (PKCE) code verifier to protect against injection attacks, which could intercept the access token.

Vehicle Provisioning module



Vehicle Provisioning module architecture

The 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 setup, 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 the 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 activated, and a policy is created and attached to the certificate. This policy is 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:

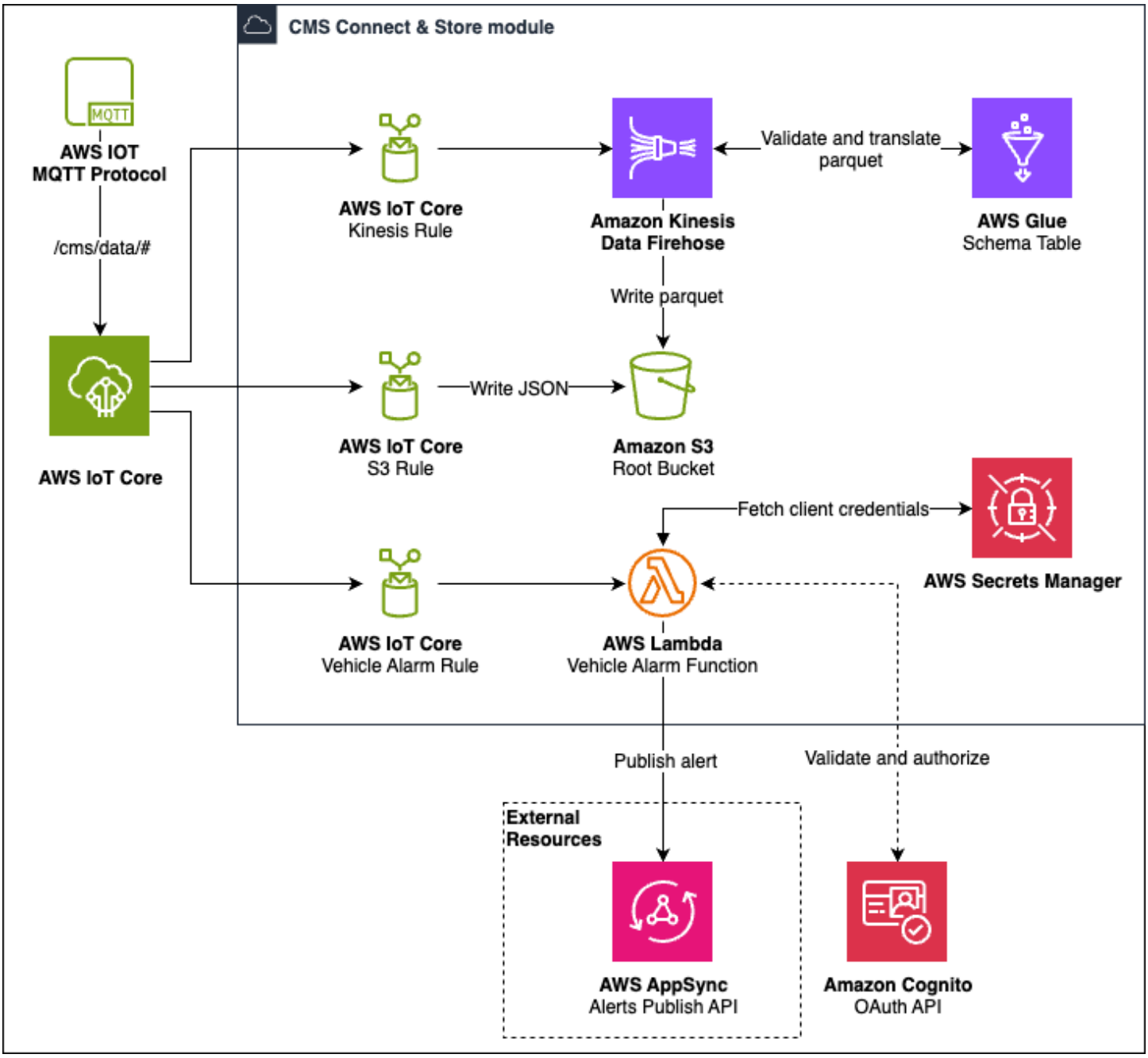
1. Updates the new certificate's status to `ACTIVE` in the `ProvisionedVehicles` DynamoDB table.
2. Deletes this vehicle's `INACTIVE` certificates from AWS IoT Core.
3. Deletes certificates' record in the `ProvisionedVehicles` DynamoDB table.

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.

CMS Connect and Store module



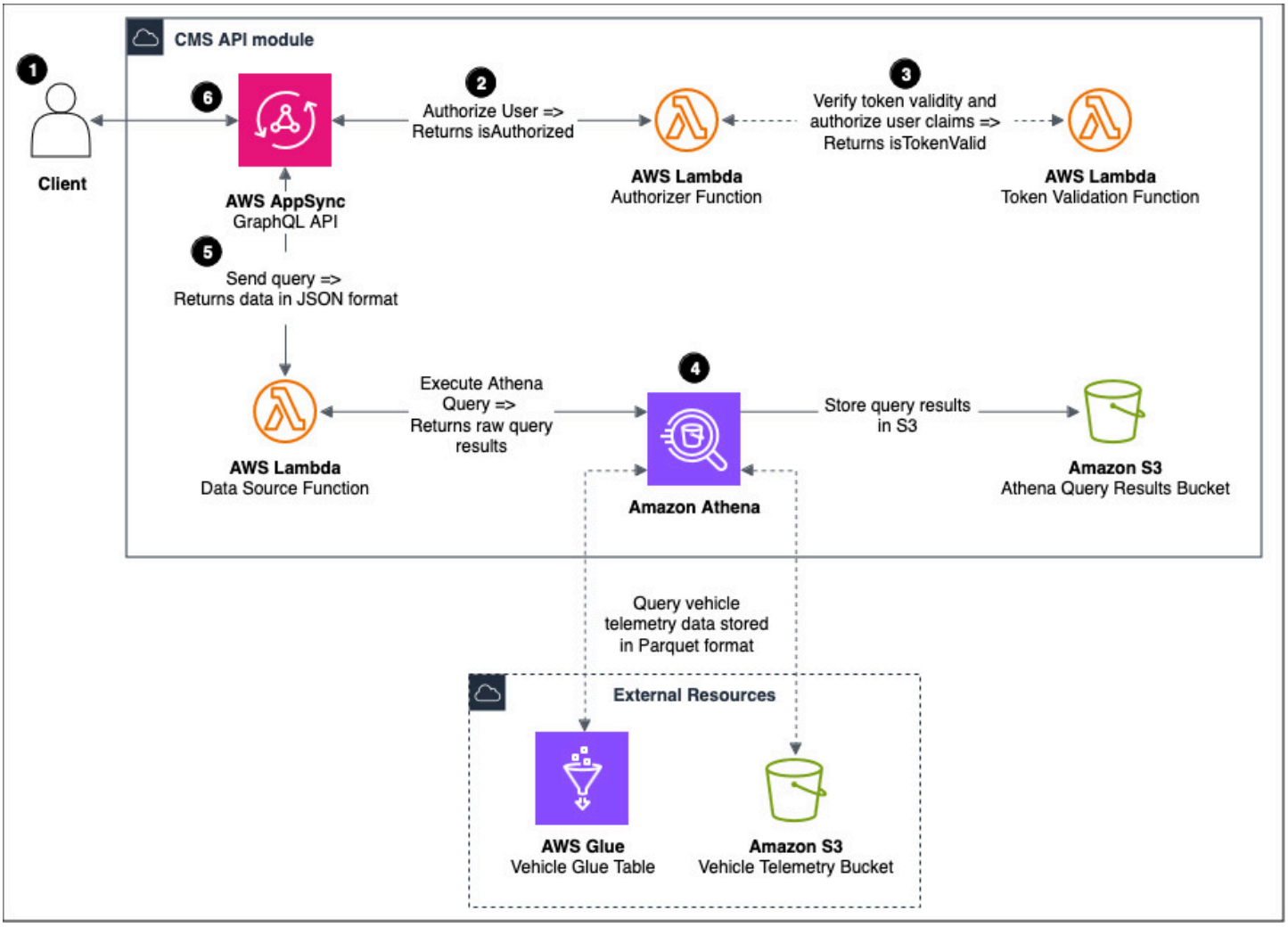
CMS Connect and Store module architecture

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.

API module



API module architecture

The 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 CMS 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.

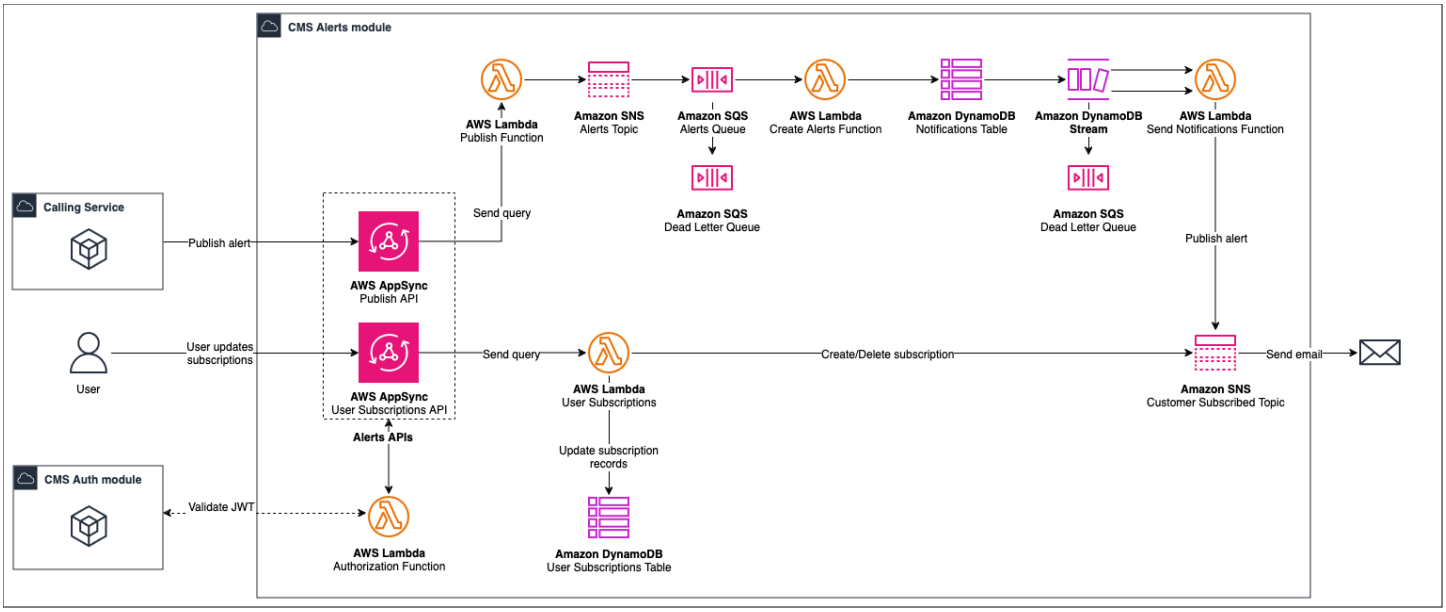
The AWS AppSync API uses Amazon Athena with a Lambda resolver data source, allowing the data to be queried the moment it is stored in the solution. This provides a near real-time representation of the state of vehicles.

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 Auth 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 Auth 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.

Alerts module



Alerts module architecture

The 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 Auth module as a prerequisite to authenticate API requests.

The user subscription AWS AppSync API uses Amazon 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 (left to right in diagram):

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 Auth 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.

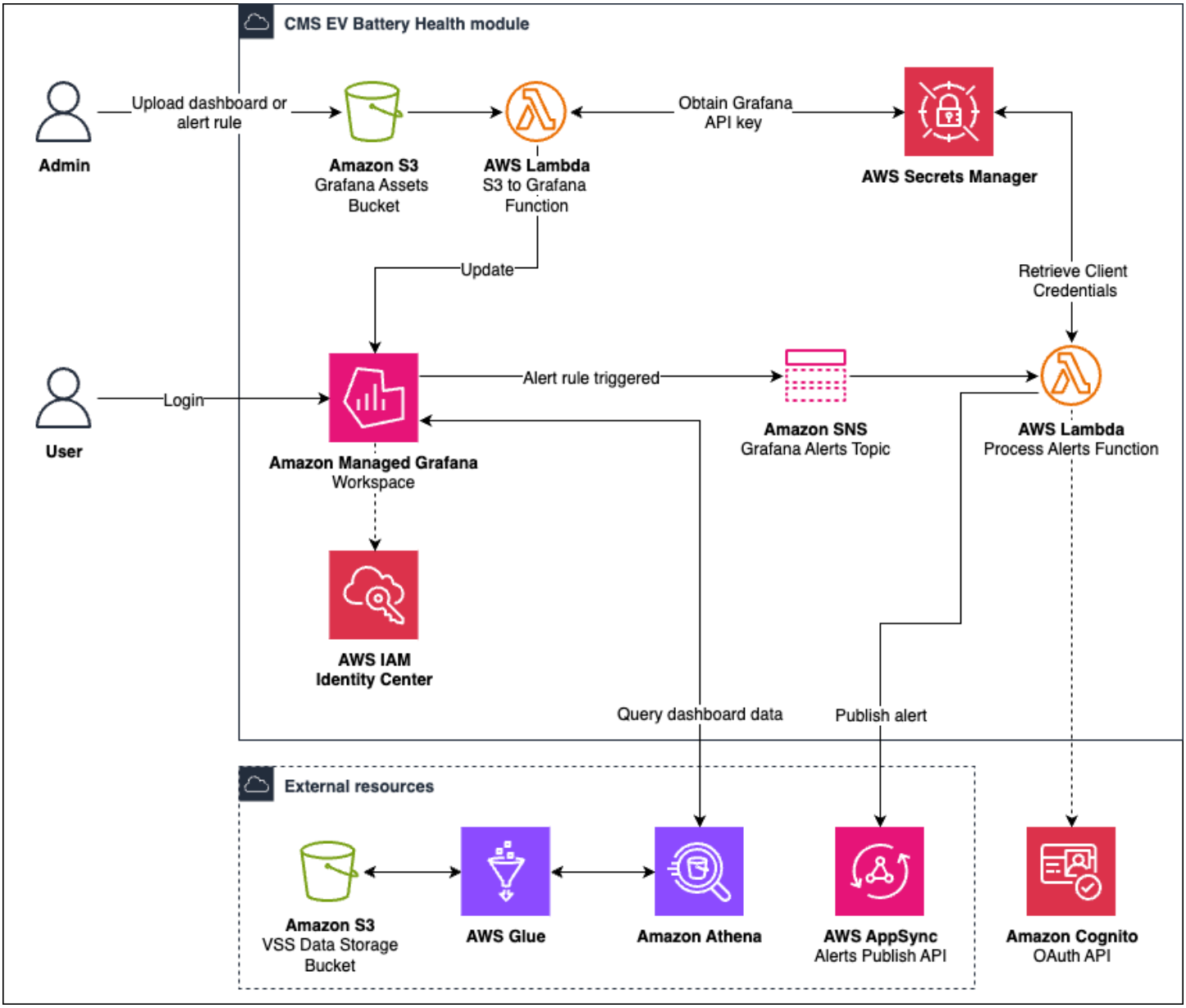
The publish API endpoint uses a Lambda function as its data source. When a CMS on AWS module sends a request, the Lambda function relays the message through the system.

The following steps explain how a request to the publish API is handled (left to right in diagram):

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 Auth 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



EV Battery Health module architecture

The 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.

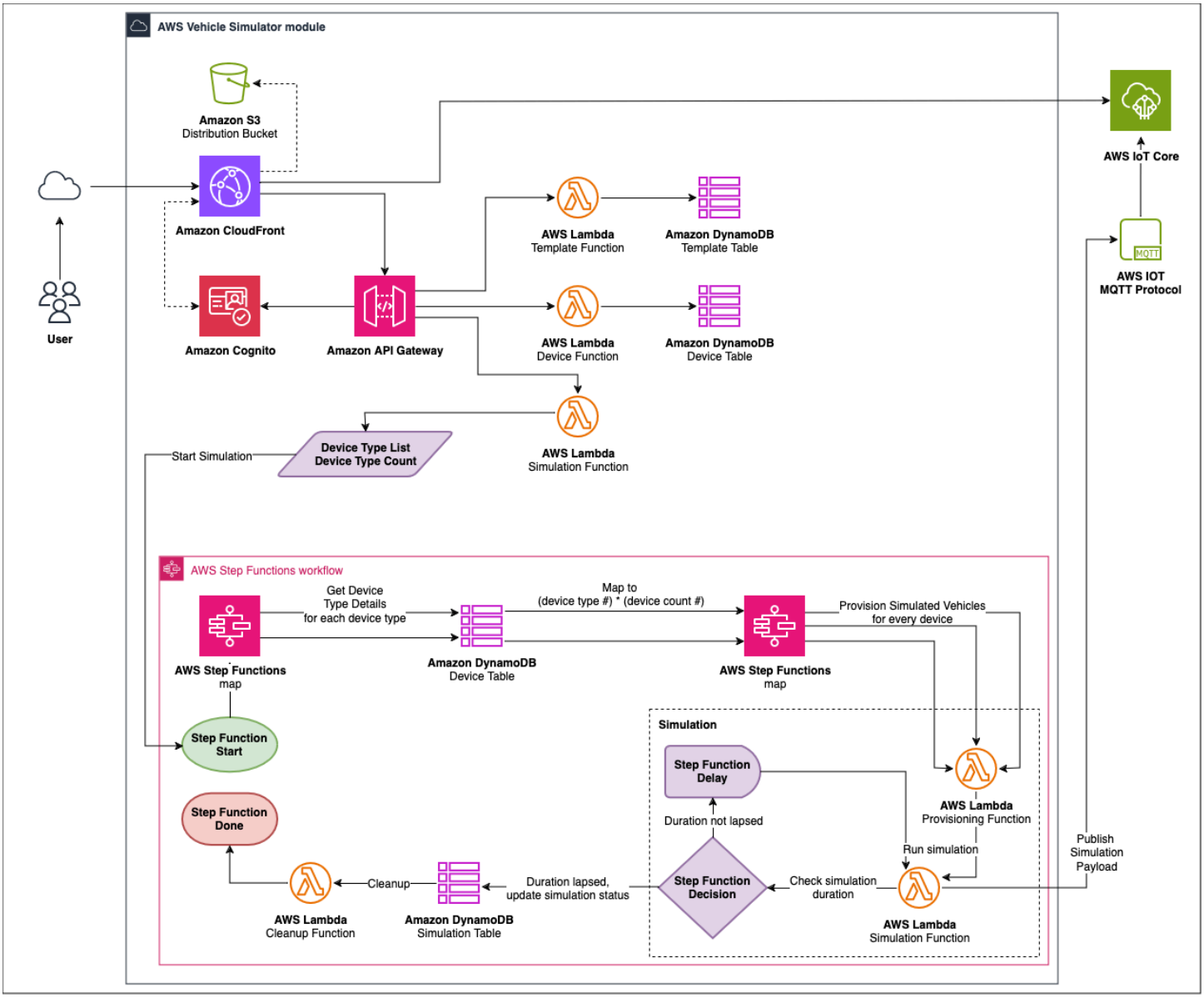
You can configure alert rules as to how often they evaluate queries, and how long they should wait before firing an alert. If the alert rule is breached, the solution sends an alert message to an Amazon SNS topic. The Amazon SNS topic has a Lambda function subscriber that processes the alert messages. 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 flow, using the service app client provided by the CMS Auth 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



Vehicle Simulator module architecture

The 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.

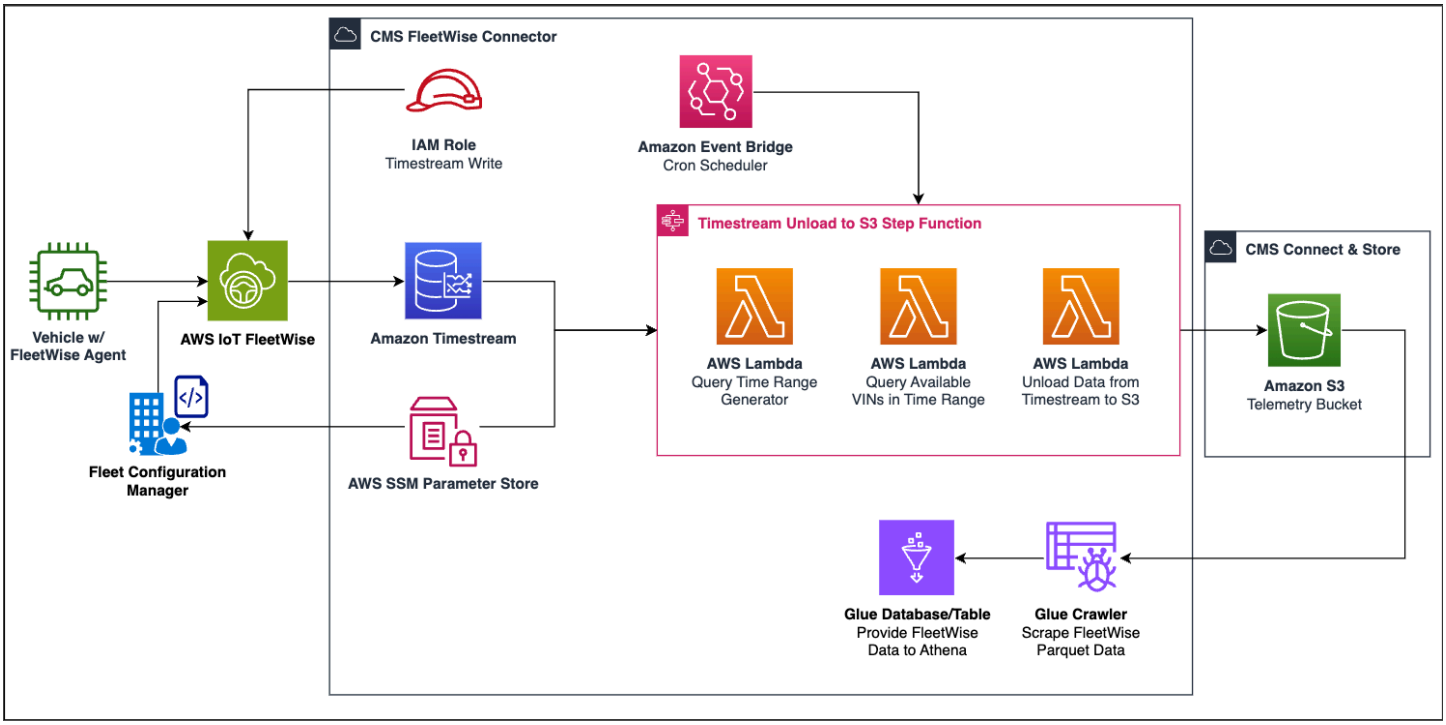
A REST API is implemented through Amazon API Gateway and backed by Lambda functions. This API provides functionality to create and store devices and simulations into DynamoDB tables.

Invoking the simulation API invokes the start of an AWS Step Functions 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.

> **Important**
>
> This module is designed to simulate data for testing. It is not recommended for production environments.

FleetWise Connector module



FleetWise Connector module architecture

The FleetWise Connector module provides the required resources and roles to consume data that is captured by AWS IoT FleetWise campaigns. This is done by leveraging the Amazon Timestream's *Unload to S3* feature to migrate data to the CMS Connect & Store module's telemetry bucket on a recurring interval, orchestrated by AWS Step Functions. The data is then indexed using an AWS Glue Crawler job and made accessible via Amazon Athena.

In order for the CMS FleetWise Connector module to function, AWS IoT FleetWise must be configured and running. When configuring an AWS IoT FleetWise Campaign, the data store must be configured to the Timestream database, table, and role created by the CMS FleetWise Connector module during deploy. This Timestream database and table are used by default for the *Unload to S3* operation. SSM parameters are created with the Timestream database, table, and role that can be used in conjunction with automated scripting of AWS IoT FleetWise Campaign creation.

AWS services in this solution

AWS service	Description
Amazon API Gateway	**Core.** Hosts REST API endpoints in the solution.
AWS AppSync	**Core.** Provides GraphQL API endpoints in the solution.
Amazon Athena	**Core.** Performs queries on vehicle data stored in Amazon S3.
AWS Certificate Manager (ACM)	**Core.** Generates certificates used for validatio n of HTTPS requests.
AWS Chalice	**Core.** Framework used to define and deploy the Vehicle Simulator module's API.
AWS CDK	**Core.** Enables infrastructure as code for the entirety of CMS on AWS.
AWS CloudFormation	**Core.** Manages deployments for the solution infrastructure.
Amazon CloudWatch	**Core.** The solution's code and infrastructure emit logs to Amazon CloudWatch.
AWS CodeBuild	**Core.** Defines build steps to aid in deploying and managing the solution.
AWS CodePipeline	**Core.** Runs collections of AWS CodeBuild steps for deploying and managing the solution.
Amazon DynamoDB	**Core.** Primary non-relational data storage used for vehicle and user records.

AWS service	Description
AWS Glue	**Core.** Validates and transforms JSON formatted payloads of a specified schema into Parquet format.
AWS IAM	**Core.** Authorizes CMS on AWS resources and users throughout solution via associated least-privilege roles and policies.
AWS IoT Core	**Core.** 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 for communication between modules.
AWS KMS	**Core.** Encrypts data in transit and at rest throughout the solution.
Amazon Data Firehose	**Core.** High throughput writing and transform ation of MQTT topic payloads to Amazon S3.
AWS Lambda	**Core.** A variety of runtime logic with serverless code throughout the solution.
Amazon Managed Grafana	**Core.** Build dashboards and configure alerts based on vehicle data stored in Amazon S3.
Parameter Store, a capability of AWS Systems Manager	**Core.** Stores configuration level information used throughout the solution.
AWS Secrets Manager	**Core.** Stores and rotates secrets used throughout the solution.
Amazon S3	**Core.** General purpose shared data storage used throughout CMS on AWS.

AWS service	Description
Amazon SNS	**Core.** Used to publish and subscribe to messages.
Amazon SQS	**Core.** Used to deliver messages between modules.
AWS Step Functions	**Core.** Orchestrates and manages Lambda functions throughout solution.
Amazon Timestream	**Core.** Timestream database used within the FleetWise Connector module.
Amazon VPC	**Core.** Networking boundary used throughout the solution.
Aurora PostgreSQL-Compatible	**Supporting.** Database used to aid with deployment.
Amazon CloudFront	**Supporting.** Provides a domain name to serve static web content and reduce latency of API endpoints.
Amazon ECS	**Supporting.** Simplifies the deployment, management, and scaling of the Backstage module.
Amazon ECR	**Supporting.** Image repository for Dockerized containers used by the Backstage module.
ELB	**Supporting.** Provides network connections to Amazon ECS tasks running Backstage to aid with deployment.
Amazon EventBridge	**Supporting.** Used within the FleetWise Connector module for scheduled execution of the Step Functions flow.

AWS service	Description
AWS IoT Fleetwise	**Supporting.** Used by the FleetWise Connector module.
AWS Fargate	**Supporting.** Combined with ECS, aids in management of EC2 instances within the Backstage module.
Amazon IAM Identity Center	**Supporting.** Service that provides authentic ation for the Amazon Managed Grafana workspace.
Amazon Route 53	**Supporting.** Domain hosting integration used by the Backstage module.
AWS X-Ray	**Supporting.** Traces runs of the AWS Step Functions.
Amazon Cognito	**Optional.** Authenticates users and internal services across the solution.
Amazon Location Service	**Optional.** Geographical functionality for the Vehicle Simulator module.

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 **$352.80 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

AWS service	Amount	Dimensions	Cost [USD]
Amazon VPC	17	Endpoints in 2 AZs	
	2	NAT gateways	$314.00
AWS KMS	27	Customer managed keys	$27.00
AWS Secrets Manager	7	Sensitive data stored in secrets	$2.80
Amazon Managed Grafana	1	1 Workspace × API license	$9.00
Total			**$352.80 / month**

Config module

AWS service	Amount	Dimensions	Cost [USD]
Total			**$0.00 / month**

VPC module

AWS service	Amount	Dimensions	Cost [USD]
Amazon VPC	17	Endpoints in 2 AZs	
	2	NAT gateways	$314.00
Total			**$314.00 / month**

Auth Setup module

AWS service	Amount	Dimensions	Cost [USD]
AWS Secrets Manager	3	PostgreSQL secret	$1.20
Total			**$1.20 / month**

Automotive Cloud Developer Portal

AWS service	Amount	Dimensions	Cost [USD]
AWS KMS	5	Customer managed keys	$5.00
Total			**$5.00 / month**

Backstage module

AWS service	Amount	Dimensions	Cost [USD]
AWS KMS	1	Customer managed keys	$1.00
AWS Secrets Manager	2	PostgreSQL secret	$0.80
Total			**$1.80 / month**

Auth module

AWS service	Amount	Dimensions	Cost [USD]
Total			**$0.00 / month**

Vehicle Provisioning module

AWS service	Amount	Dimensions	Cost [USD]
AWS KMS	2	Customer managed keys	$2.00
AWS Secrets Manager	1	Claim certificate created on deploy	$0.40
Total			**$2.40 / month**

CMS Connect and Store module

AWS service	Amount	Dimensions	Cost [USD]
AWS KMS	3	Customer managed keys	$3.00
Total			**$3.00 / month**

API module

AWS service	Amount	Dimensions	Cost [USD]
AWS KMS	1	Customer managed keys	$1.00
Total			**$1.00 / month**

Alerts module

AWS service	Amount	Dimensions	Cost [USD]
AWS KMS	7	Customer managed keys	$7.00
Total			**$7.00 / month**

EV Battery Health module

AWS service	Amount	Dimensions	Cost [USD]
AWS KMS	3	Customer managed keys	$3.00
AWS Secrets Manager	1	Amazon Managed Grafana API key secret	$0.40
AWS Grafana	1	1 Workspace x API license	$9.00
Total			**$12.40 / month**

Vehicle Simulator module

AWS service	Amount	Dimensions	Cost [USD]
AWS KMS	2	Customer managed keys	$2.00
Total			**$2.00 / month**

FleetWise Connector module

AWS service	Amount	Dimensions	Cost [USD]
AWS KMS	3	Customer managed keys	$3.00
Total			**$3.00 / month**

Example costs

This solution provides zero-cost-on-no-use. Consider the following use cases:

Use case 1 - Solution deployed for 100 devices connected with little to no analytics run.

Service	Configuration summary	Monthly Cost [USD]
MQTT	Number of devices (MQTT) (100), Average size of each message (5 KB), Average size of each record (1 KB), Average size of each record (1 KB), Average number of actions applied per rule (2), Average size of each message (5 KB), Average number of connection minutes for a device (24000), Number	$ 2.59

Service	Configuration summary	Monthly Cost [USD]
	of messages for a device (24000), Number of rules initiated (5).	
S3 Standard	S3 Standard storage (1 GB per month), PUT, COPY, POST, LIST requests to S3 Standard (720), GET, SELECT, and all other requests from S3 Standard (720), Data returned by S3 Select (1 GB per month), Data scanned by S3 Select (4 GB per month).	$ 0.04
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 (1200 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).	$ 4.91

Service	Configuration summary	Monthly Cost [USD]
Amazon Athena	Amount of data scanned per query (1 GB), Total number of queries (100 per day)	$ 14.85
DynamoDB on-demand capacity	Table class (Standard), Average item size (all attributes) (3 KB), Data storage size (0.1 GB)	$ 0.05
Amazon CloudFront	Data transfer out to internet (1 GB per month), Number of requests (HTTPS) (1000 per month).	$ 0.09
Amazon CloudWatch	Standard Logs: Data Ingested (0.1 GB), Logs Delivered to CloudWatch Logs: Data Ingested (0.1 GB), Logs Delivered to Amazon S3: Data Ingested (0.1 GB), Number of Lambda functions (20), Number of requests per function (10 per day).	$ 48.13
Amazon Cognito	Number of monthly active users (MAU) (100), Advanced security features (Enabled)	$ 5.00
Amazon Managed Grafana	Number of active editors/ administrators (1 per workspace per month), Number of active viewers (5 per workspace per month).	$ 34.00

Service	Configuration summary	Monthly Cost [USD]
Standard topics	DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), Requests (10000 per month), EMAIL/EMAIL-JSON Notifications (10000 per month), Mobile Push Notifications (20000 per month).	$ 0.19
AWS AppSync API Request	Number of API Requests (10 thousand per month).	$ 0.04
Total		**$ 109.89**

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 10MB 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 1MB of vehicle data stored in DynamoDB storage. Historical data will be exported to Amazon S3. Vehicle profile data will reside in DynamoDB, while all vehicle-generated data will be stored in Amazon S3.
8. 100 Simple Notification Service (Amazon SNS) notification requests per vehicle per day, alongside 20 mobile push notifications per vehicle per day.
9. Five Dashboard editors and 50 viewers for Grafana.

10. In the context of DevSecOps, we anticipate retaining logs for a period of three months. Please be aware that the cost of CloudWatch will increase because we are obligated 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.

AWS service	Dimensions	Monthly Cost [USD]
MQTT	Number of devices (MQTT) (1200000), Average size of each message (5 KB), Average size of each record (1 KB), Average size of each record (1 KB), Average number of actions taken per rule (2), Average size of each message (5 KB), Average number of connection minutes for a device (28000), Number of messages for a device (28000), Number of rules triggered (5)	$ 26,910.7
S3 Standard	S3 Standard storage (10 TB per month), PUT, COPY, POST, LIST requests to S3 Standard (920000000), GET, SELECT, and all other requests from S3 Standard (920000000), Data returned by S3 Select (5 TB per month), Data scanned by S3 Select (8 TB per month)	$ 5,249.09
Amazon Data Firehose	Source Type (Direct PUT or Kinesis Data Stream), Data records units (exact	$ 11,679.87

AWS service	Dimensions	Monthly Cost [USD]
	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 (1200000000 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)	
Amazon Athena	Amount of data scanned per query (100 GB), Total number of queries (110000 per month)	$ 53,710.94
DynamoDB on-demand capacity	Table class (Standard), Average item size (all attributes) (3 KB), Data storage size (10 GB)	$ 185.00
Amazon CloudFront	Data transfer out to internet (1 TB per month), Number of requests (HTTPS) (60000 per month), Data transfer out to origin (1 TB per month)	$ 107.58

AWS service	Dimensions	Monthly Cost [USD]
Amazon CloudWatch	Standard Logs: Data Ingested (1 TB), Logs Delivered to CloudWatch Logs: Data Ingested (10 GB), Logs Delivered to S3: Data Ingested (10 GB), Number of Lambda functions (20), Number of requests per function (10000 per day)	$ 575.64
Amazon Cognito	Number of monthly active users (MAU) (100), Advanced security features (Enabled)	$ 5.00
Amazon Managed Grafana	Number of active editors/ administrators (5 per workspace per month), Number of active viewers (50 per workspace per month)	$ 295.00
Standard topics	Requests (200 million per month), EMAIL/EMAIL-JSON Notifications (200 million per month), Mobile Push Notifications (20 million per month)	$ 4,109.48
AWS AppSync API Request	Number of API Requests (300 million per month)	$ 1,200.00
Amazon Elastic Container Registry	DT Inbound: All other regions (1 TB per month), DT Outbound: Not selected (0 TB per month), Amount of data stored (280 GB per month)	$ 28.00
Total		**$ 104,056.30**

Note

The price is based on public pricing as displayed in the AWS Price Calculator. Customers may have a special pricing agreement based on reserve capacity and negotiated rates with AWS.

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 and services must provide a valid access token associated with the IdP configured within the solution. By allowing customers to use their own identity provider, they have full control over the configuration of their authentication system.

All JWTs used for authentication and authorization are validated through protocol defined by OAuth2.0 standards. For more details, see Auth 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, an optional PKCE code verifier for user authentication, and the use of the authorization code flow for user authentication.

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 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*.

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

For the most current availability of AWS services by Region, see the AWS Regional Services List.

Connected Mobility Solution on AWS is supported in the following AWS Regions:

Region name	
US East (Ohio)	Asia Pacific (Tokyo)
US East (N. Virginia)	Asia Pacific (Sydney)
US West (Oregon)	Europe (Frankfurt)
Europe (Ireland)	

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 services implemented 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 deploying 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*.

Deploy the solution

This solution uses AWS CloudFormation templates and stacks to automate its deployment. The CloudFormation templates specify the AWS resources included in this solution and their properties. The CloudFormation stack provisions the resources that are described in the templates.

Deployment process overview

Follow the instructions in this section to configure and deploy CMS on AWS into your account, and learn how to deploy CMS on AWS modules through Backstage.

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

> **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.
>
> To opt out of this feature, download the template, modify the AWS CloudFormation mapping section, and then use the AWS CloudFormation console to upload your updated template and deploy the solution. For more information, see the Anonymized data collection section of this guide.

AWS CloudFormation templates

You can download the CloudFormation template for this solution before deploying it.

View template

acdp.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the Automative Cloud Developer Portal section, but you can customize the template to meet your specific needs.

View template

auth.setup.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the Auth Setup module section, but you can customize the template to meet your specific needs.

View template

cms-api.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the API module section, but you can customize the template to meet your specific needs.

View template

cms-ev-battery-health.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the EV Battery Health module section, but you can customize the template to meet your specific needs.

View template

cms-auth.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the Auth module section, but you can customize the template to meet your specific needs.

View template

cms-sample.template - Use this template for integrating custom modules.

View template

cms-vehicle-simulator.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the Vehicle Simulator module section, but you can customize the template to meet your specific needs.

View template

cms-alerts.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the Alerts module section, but you can customize the template to meet your specific needs.

View template

cms-fleetwise-connector.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the FleetWise Connector module section, but you can customize the template to meet your specific needs.

View template

vpc.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the VPC module section, but you can customize the template to meet your specific needs.

View template

cms-provisioning.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the Vehicle Provisioning module section, but you can customize the template to meet your specific needs.

View template

cms-connect-store.template - Use this template to deploy the module and all associated components. The default configuration deploys the services found in the CMS Connect and Store module section, but you can customize the template to meet your specific needs.

View template

cms-config.template - Use this template to deploy the module and all associated components.

The default configuration deploys the services found in the Config module section, but you can customize the template to meet your specific needs.

> **Note**
>
> AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.
> If you deployed a previous version of this solution, you must first uninstall your previous deployment and then deploy the new version.

Prerequisites

This solution uses AWS CDK to produce AWS CloudFormation templates for consistent provisioning and configuration of deployments. You must meet the following prerequisites before launching the solution.

Clone the repository

Use the following command to clone the repository:

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

Required tools

To deploy CMS on AWS, a variety of tools are required. These deploy instructions will install the following to your machine.

1. NVM
2. Node
3. NPM
4. Yarn
5. Pyenv
6. Python
7. Pip

8. Pipenv
9. AWS CLI
10. AWS CDK Toolkit

Required tool versions

Certain tools also require specific versions. See the table below for the appropriate versions. Following the provided install instructions will install the correct versions.

For tools not listed here, stable versions should work appropriately.

NodeJS	18.17.*
Python	3.10.*

Install required tools

> **Note**
>
> 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.

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

Node / NPM

```
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.

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

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 ...
```

AWS CDK Toolkit

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

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

Verify required tool installations

Run the following command to verify the proper installation of all of the tools listed above. If any errors are displayed, attempt to reinstall that tool.

```
make verify-required-tools
```

Install solution dependencies

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

```
make install
```

Create an Amazon Route 53 hosted zone

To deploy the solution, an Amazon Route 53 hosted zone is required in your account. You will provide the domain for this hosted zone in the following step when you setup your environment variables. 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 values to your deployment. To generate the file which will store these environment variables and provide their values, run the following command:

```
make deploy-variables
```

> **Note**
>
> The ROUTE53_ZONE_NAME can be found from the Amazon Route 53 hosted zone you set up in the previous step. Use the AWS Management Console to find this domain.

Step 1: Build the Solution's modules

The build target manages dependencies, builds required assets (e.g. packaged lambdas), and creates the AWS CloudFormation templates for all modules.

```
make build
```

Step 2: Upload Assets to S3

The upload target creates the necessary buckets for, and uploads, the global and regional assets. It also uploads the Backstage .zip asset.

```
make upload
```

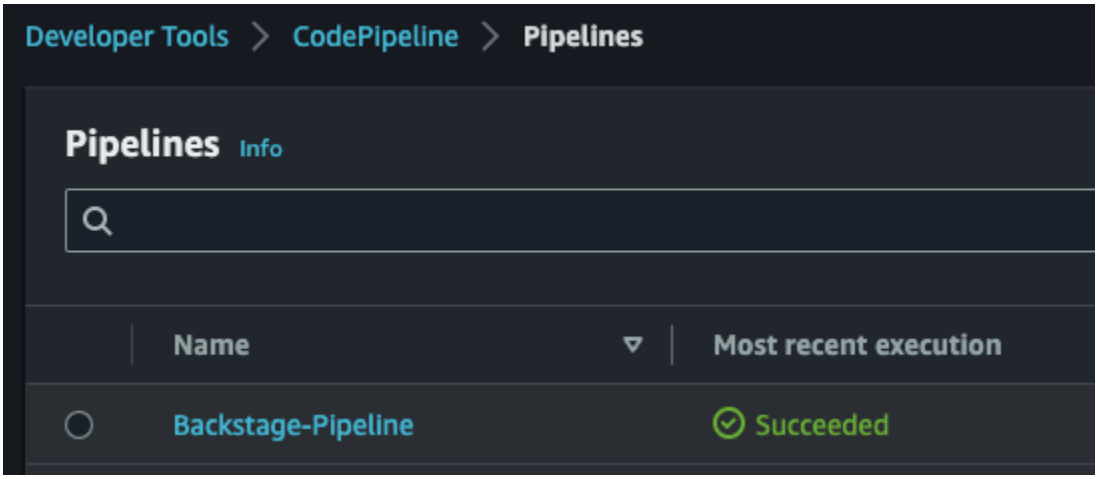
Step 3: Deploy on AWS

The deploy target deploys all CMS modules, including the ACDP, in an enforced order.

```
make deploy
```

Step 4: Monitor the ACDP deployment

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



Backstage-Pipeline Succeeded in CodePipeline

After the pipeline has completed, the deployment can be considered successfully complete and Backstage is ready for us.

Step 5: Deploy CMS Modules via Backstage

CMS Module Deployment Order

All CMS on AWS modules have dependencies on the initial three deployments for configuring CMS on AWS.

Some CMS on AWS modules have secondary dependencies on other modules and must be deployed in order.

The rest of the modules do not have dependencies on other modules and can be deployed in any order after CMS Config.

The deployment order that must be observed is as follows:

Deployment Order of Required CMS Config

1. VPC
2. Auth Setup
3. CMS Config

> **Note**
>
> At this point, ACDP and Backstage should be deployed to assist with the further module deployments.

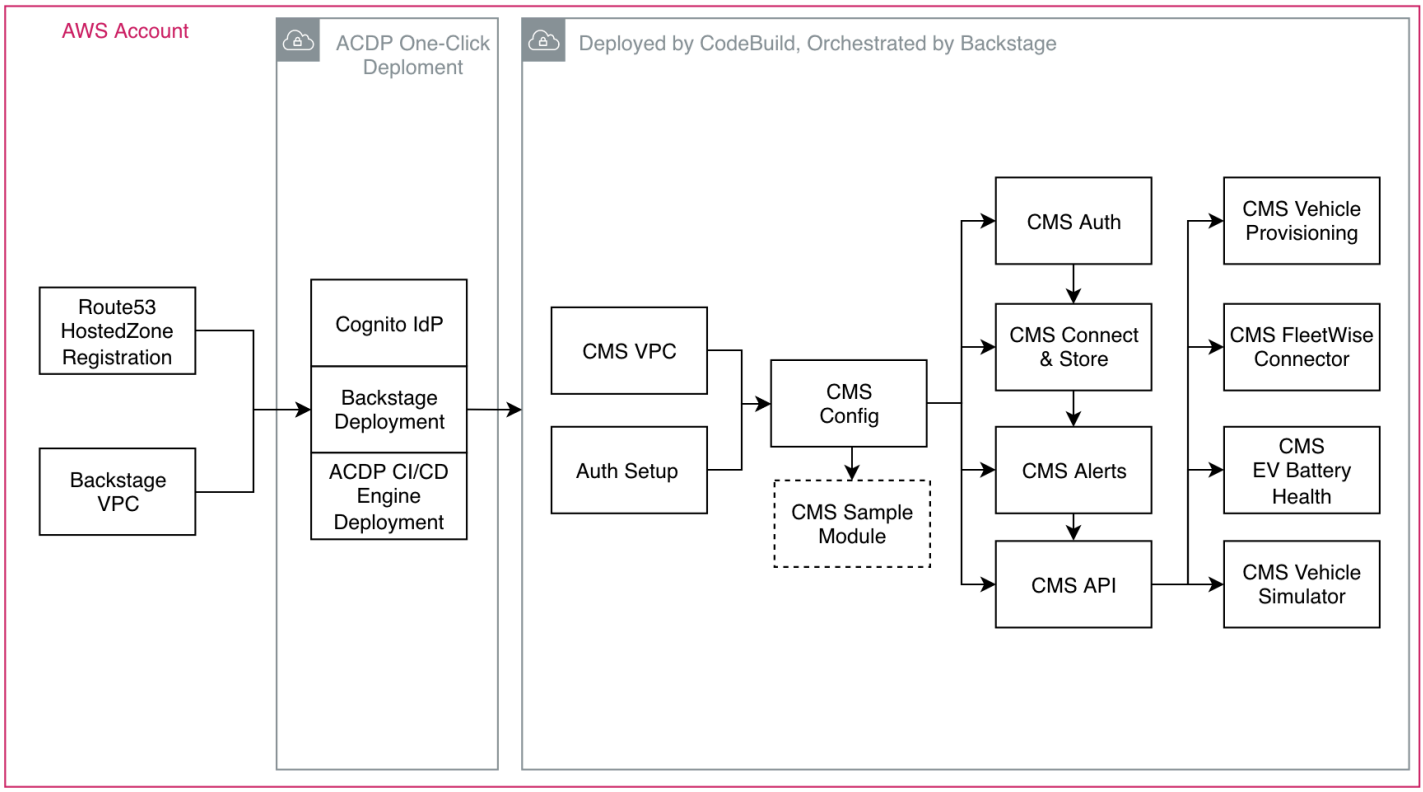
Deployment Order of Modules with Dependencies

1. Auth
2. CMS Connect and Store
3. Alerts
4. API
5. EV Battery Health
6. FleetWise Connector

Modules Without Dependencies

- Vehicle Provisioning
- Vehicle Simulator

This order is represented by the diagram below:

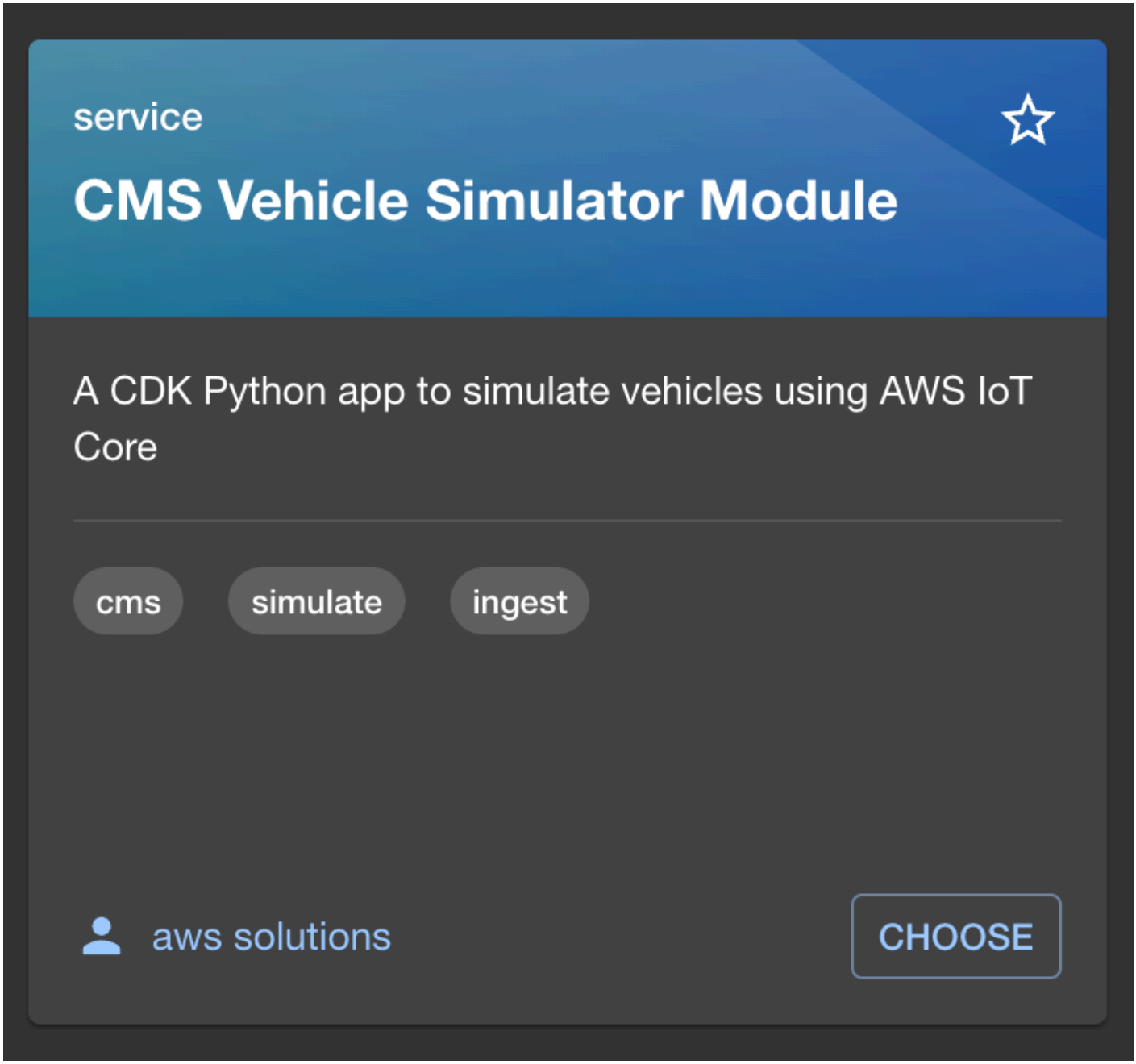


CMS module deployment order

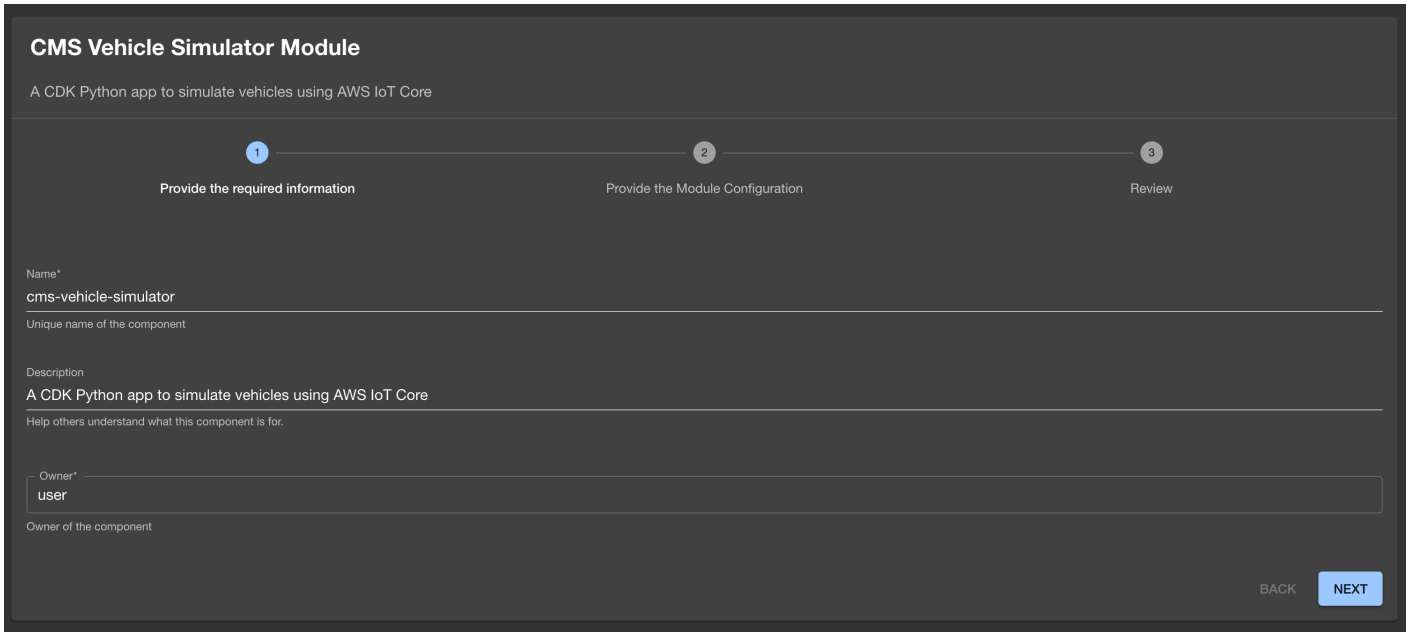
Example module deployment via Backstage

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.

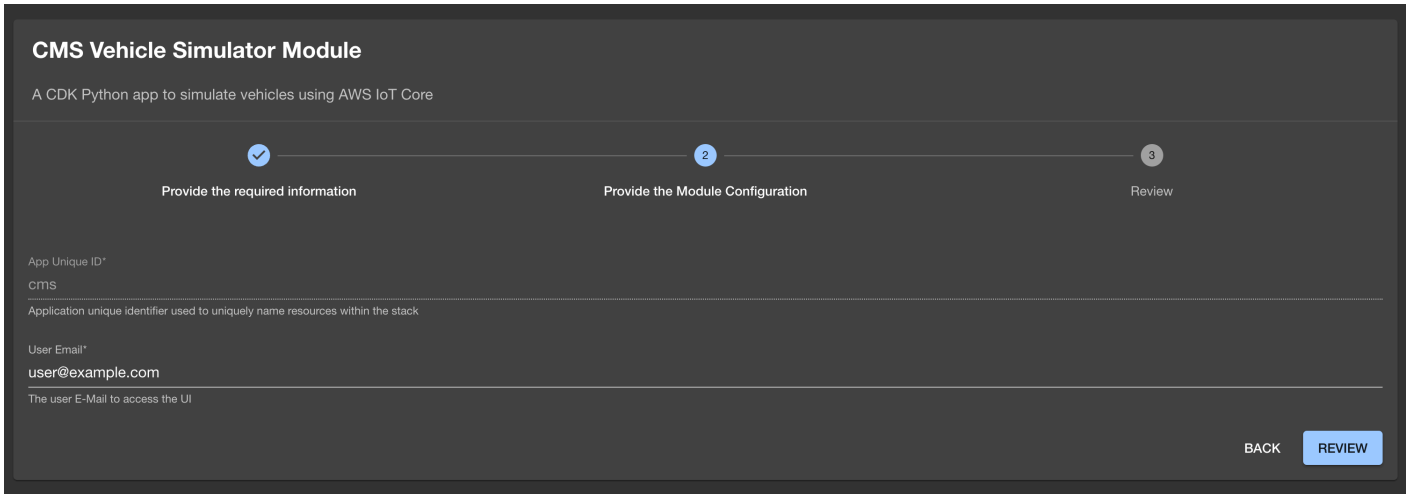
1. Navigate to the Backstage URL in a web browser (`ROUTE53_BASE_DOMAIN` that was specified during deployment).
2. Sign in to Backstage using the credentials that were emailed to the user email specified during deployment.
3. Follow the prompts to create a new password and set up multi-factor authentication (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** card.



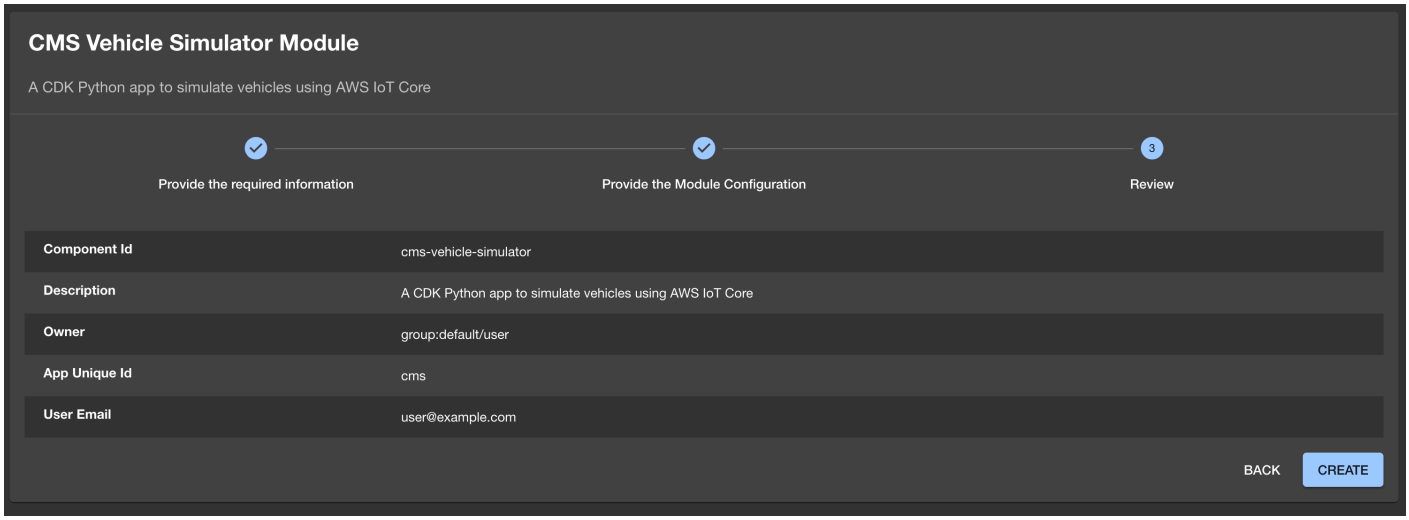
5. Fill in the form as required by the CMS Vehicle Simulator module's template and choose **Next**.



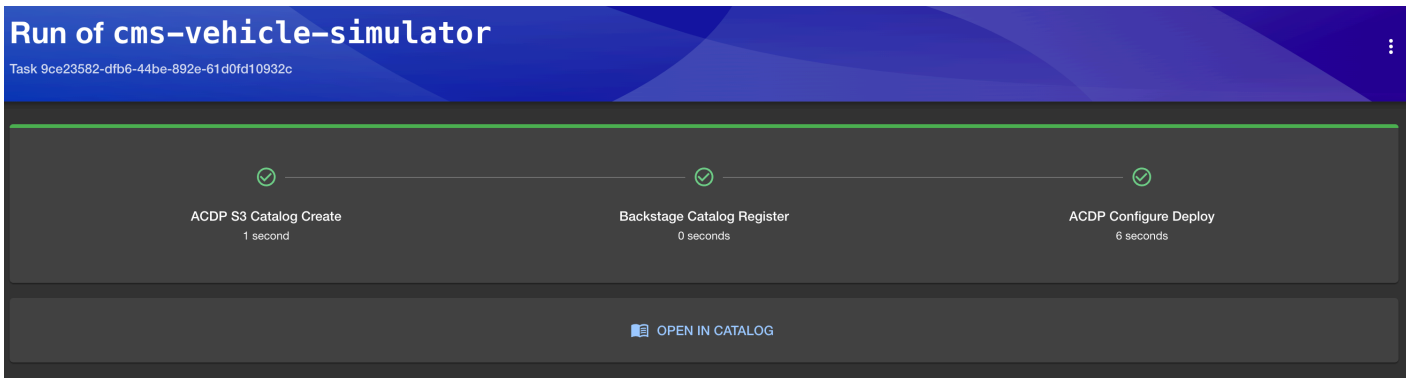
6. Choose **Review**.



7. Choose **Create**.



8. Monitor the deployment and ensure that the CMS Vehicle Simulator module deploys successfully.



Step 6: Secure the solution with network access control

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.

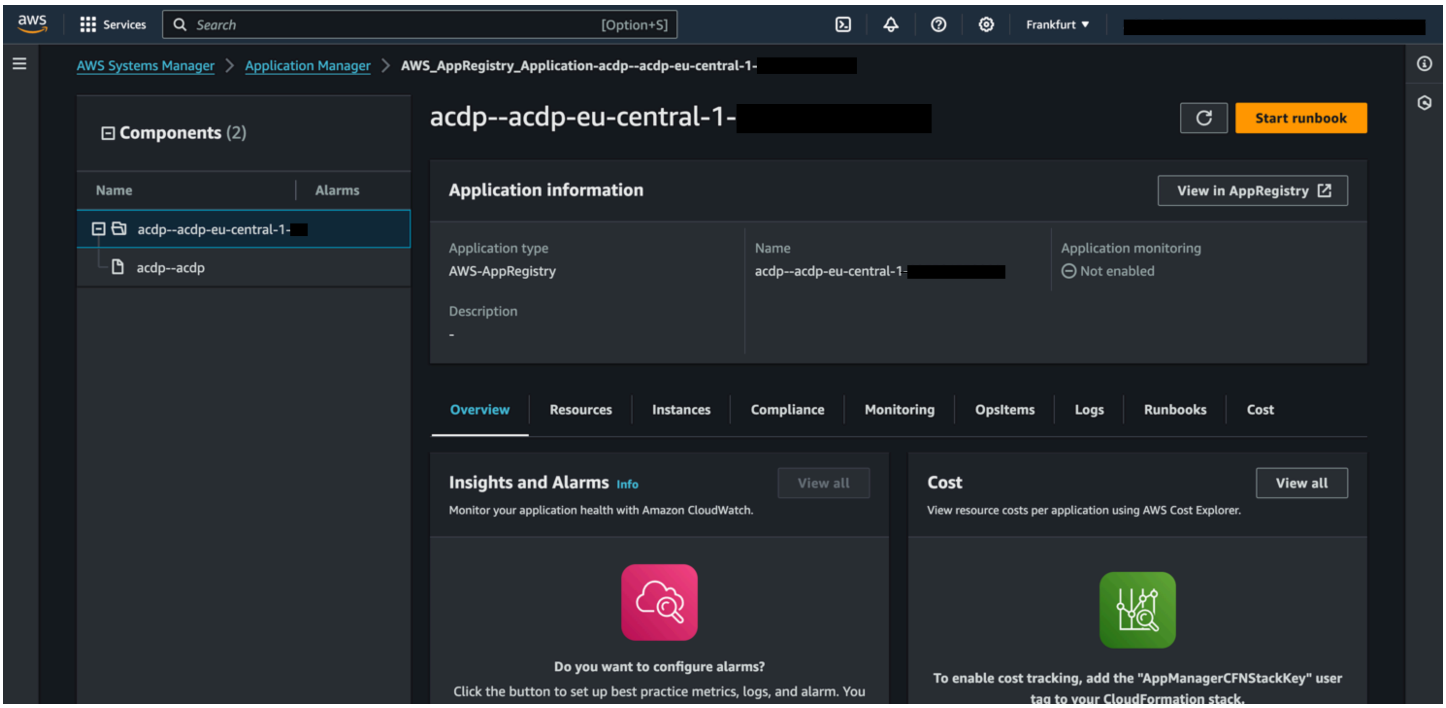
Monitoring the solution with Service Catalog AppRegistry

The solution 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 this solution stack in Application Manager.



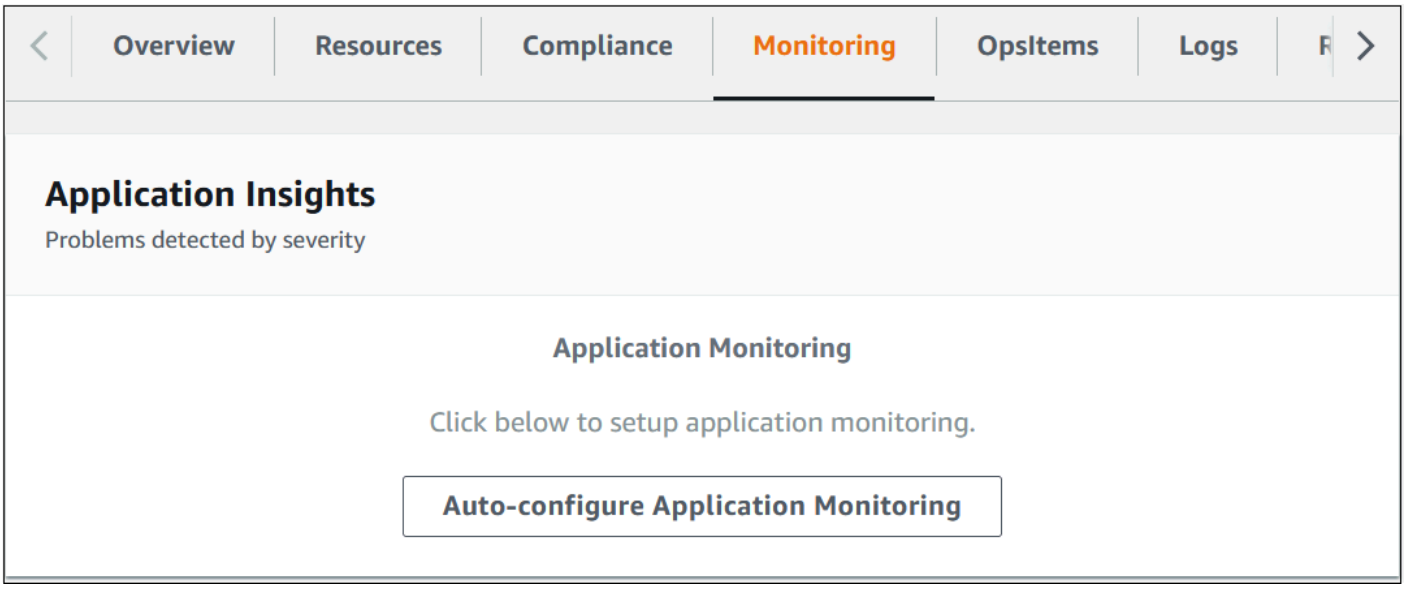
CMS on AWS stack in Application Manager

> **Note**
>
> Activate CloudWatch Application Insights, AWS Cost Explorer, and cost allocation tags associated with this solution. They are not activated by default.

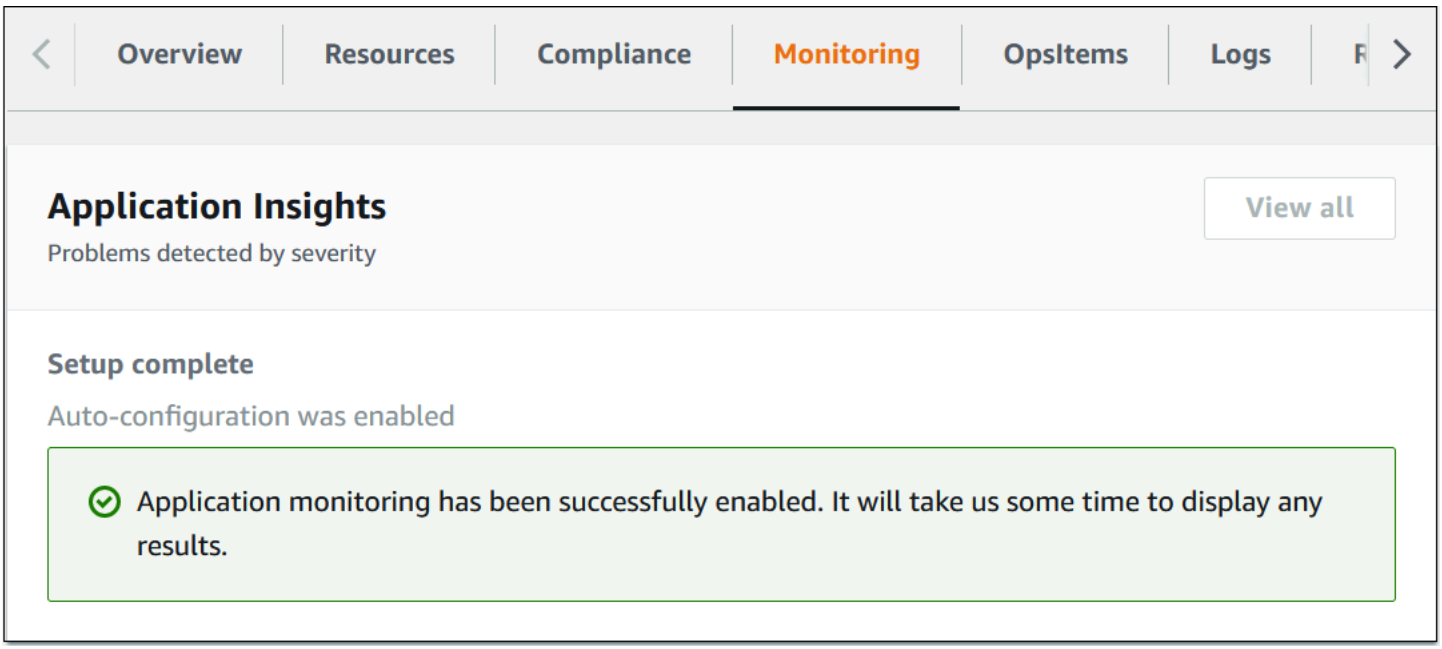
Activate CloudWatch Application Insights

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.

   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.
6. In the Monitoring tab, in Application Insights, select Auto-configure Application Monitoring.



Monitoring for your applications is now activated and the following status box appears:



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 Management console.
2. In the navigation pane, select **Cost Explorer**.
3. On the **Welcome to Cost Explorer** page, choose **Launch Cost Explorer**.

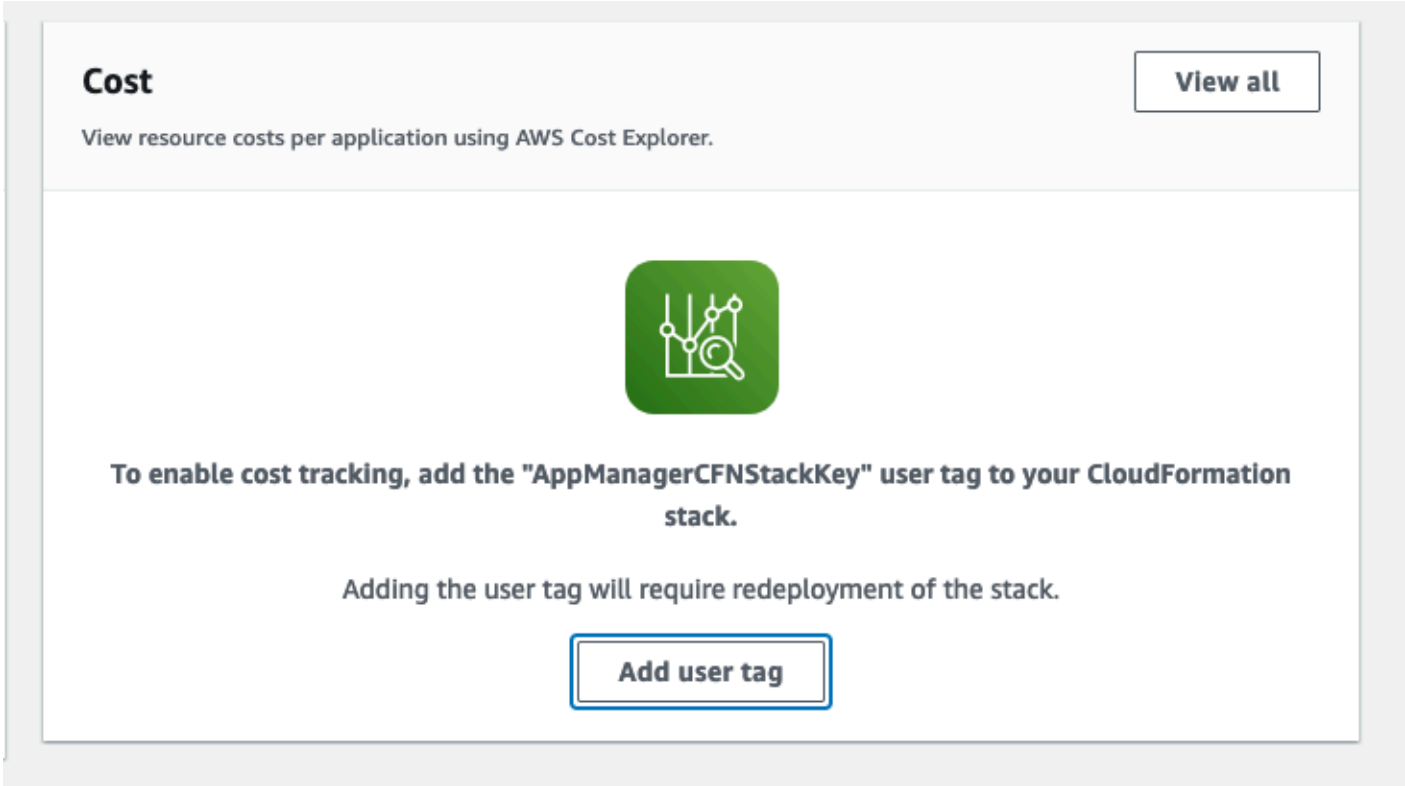
The activation process can take up to 24 hours to complete. Once activated, you can open the Cost Explorer user interface to further analyze cost data for the solution.

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 and Cost Management console.
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.

Update the solution

If you deployed a previous version of this solution, you must first uninstall your previous deployment and then deploy the new version.

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 (AWS CLI).
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.

Contact AWS Support

If you have AWS Developer Support, AWS Business Support, or AWS Enterprise Support, you can use the Support Center to get expert assistance with this solution. The following sections provide instructions.

Create case

1. Sign in to Support Center.
2. Choose **Create case**.

How can we help?

1. Choose **Technical**.
2. For **Service**, select **Solutions**.
3. For **Category**, select **Other Solutions**.
4. For **Severity**, select the option that best matches your use case.
5. When you enter the **Service**, **Category**, and **Severity**, the interface populates links to common troubleshooting questions. If you can't resolve your question with these links, choose **Next step: Additional information**.

Additional information

1. For **Subject**, enter text summarizing your question or issue.
2. For **Description**, describe the issue in detail.
3. Choose **Attach files**.
4. Attach the information that AWS Support needs to process the request.

Help us resolve your case faster

1. Enter the requested information.
2. Choose **Next step: Solve now or contact us**.

Solve now or contact us

1. Review the **Solve now** solutions.
2. If you can't resolve your issue with these solutions, choose **Contact us**, enter the requested information, and choose **Submit**.

Uninstall the solution

This solution creates multiple CloudFormation deployments. Some resources cannot be uninstalled directly with CloudFormation and must be deleted by using the AWS Management Console or the AWS CLI.

Capture the deployment UUID

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

```
make get-acdp-deployment-uuid
make get-cms-deployment-uuid
```

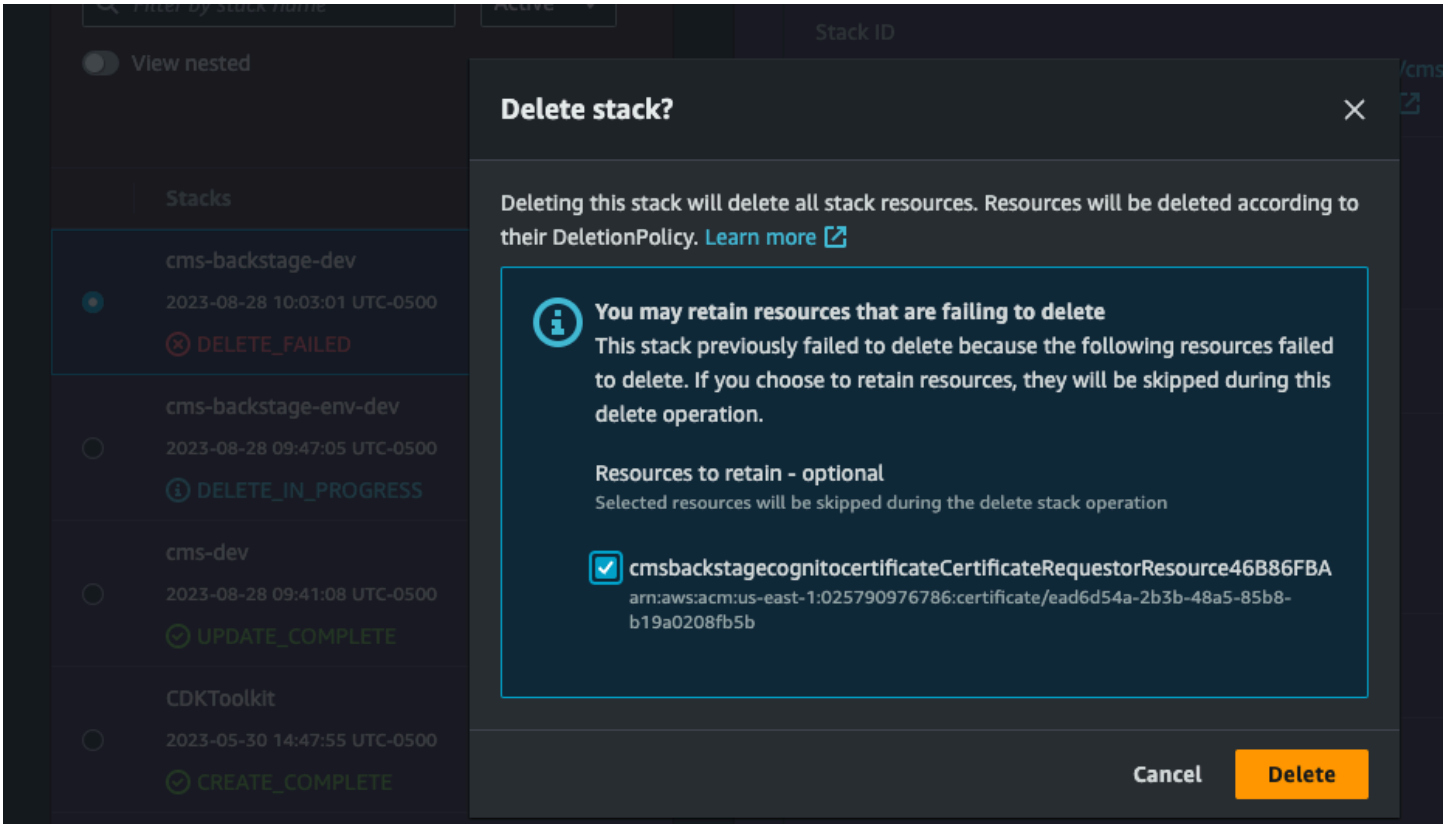
The output will be uuidv4 strings, capture and store both:

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

Delete CMS on AWS modules in order

```
make destroy
```

> **Note**
>
> Backstage might fail to delete due to the ACM certificate creation custom resource. After delete fails, select DELETE again and select retain on the custom resource. This will not leave any resources in the account.



Backstage stack delete fails. Redelete with retention.

Delete the Backstage ACM certificate (optional)

Navigate to Amazon Certificate Manager, and delete the Backstage certificate.

Manually clean up resources

You must manually clean up the following resources:

- S3 buckets
- DynamoDB tables
- Cognito user pool
- KMS keys

Locate leftover resources using the following command, which first requires you to export the `DEPLOYMENT_UUID` variable using each of the values previously acquired from AWS Systems Manager.

If you tore down the ACDP 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.

```
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, utilizing the resource ARNs where appropriate.

> **Important**
>
> 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 S3 buckets created by the solution.
4. Select an S3 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:

```
$ 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.
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:

```
$ 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.

1. Sign in to the AWS KMS console.
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.
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>
```

Deleting 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

This section provides the source code for the solution and additional customizations.

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) (AWS CDK) generates the Connected Mobility Solution on AWS templates. See the README.md file for additional information.

Integrating custom modules

We have provided a CMS Sample module which serves as a template for existing and future CMS on AWS modules. See the code and README.md in the GitHub repository for more information on development.

For more information on integrating a CMS on AWS module with the Backstage deployment, see the Backstage documentation.

Reference

This section includes a list of builders and managers 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/modules/cms_config/source/infrastructure/constructs/metrics.py file, replace the `send_anonymous_usage_condition` value with No.
2. Continue with the deployment as specified in the Deploy the solution section.

Contributors

- Narmeen Ali
- Kevin Hargita
- Guru Koushik Senthil Kumar
- Anthony McIntosh
- Harshit Patel
- Alex Sansone
- Saif Shaikh
- Matt Wise

Revisions

Date	Change
October 2023	Initial release
December 2023	v1.0.1: • Resolved an issue where the Amazon Aurora PostgreSQL cluster's version defaulted to 11 instead of 13 in some Regions. • Pinned Node and Python versions in the Proton `manifest.yml` file for every module. • Updated cost table for Use case 2. • Corrected URLs in the `README.md` file. For more information, refer to the CHANGELOG.md file in the GitHub repository.
January 2024	v1.0.2: • Updated Grafana workspace in EV Battery Health module to include plugin management and install Amazon Athena plugin. • Removed yarn `tsc:full` from backstage image build. • Added octokit and resolution for follow-re directs to mitigate vulnerabilities. • Added ignore pattern for Axios in vehicle simulator. For more information, refer to the CHANGELOG.md file in the GitHub repository.

Date	Change
February 2024	v1.0.3: • Added resolutions to ECDSA, cryptography, and node-ip packages to mitigate vulnerabilities. For more information, refer to the CHANGELOG.md file in the GitHub repository.
February 2024	v1.0.4: • Upgraded Backstage to 1.23.3 to mitigate vulnerability. • Fixed a bug that could occur if the Amazon S3 version of the Backstage source was prefixed with a special character. For more information, refer to the CHANGELOG.md file in the GitHub repository.

Date	Change
April 2024	v 1.1.0: • Created a VPC module to provide a reference VPC implementation for ACDP and CMS and AWS modules. • Created a Config module to define common configurations within the solution. • Created an Auth Setup module to support choosing between Amazon Cognito or a compatible OAuth 2.0 compliant IdP. • Updated the ACDP module and improved support for Backstage. • Added one-click deployment support through CloudFormation templates. For more information, refer to the CHANGELOG.md file in the GitHub repository.
April 2024	v 1.1.1: • Upgraded mysql2 to resolve CVE. • Upgraded requests library with idna peer dependency to resolve pip-audit. • Upgraded @backstage/cli to resolve Jest errors. • Pin moto version in Alerts module to avoid moto Athena bug introduced in moto 5.0.3

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.