



Developer Guide

AWS IoT FleetWise



AWS IoT FleetWise: Developer Guide

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

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

Table of Contents

What is AWS IoT FleetWise?	1
Benefits	2
Use cases	2
Are you new to AWS IoT FleetWise?	3
Accessing AWS IoT FleetWise	3
Pricing for AWS IoT FleetWise	3
How AWS IoT FleetWise works	3
Key concepts	4
Features of AWS IoT FleetWise	8
Related services	8
Setting up AWS IoT FleetWise	9
Set up your AWS account	9
Sign up for an AWS account	9
Create a user with administrative access	10
Get started in the console	11
Configuring settings	11
Configure settings (console)	12
Configure settings (AWS CLI)	12
Getting started	14
Requirements	14
Using the Edge Agent software demo	14
Getting started (console)	15
Prerequisites	16
Step 1: Set up the Edge Agent software for AWS IoT FleetWise	16
Step 2: Create a vehicle model	18
Step 3: Create a decoder manifest	20
Step 4: Configure a decoder manifest	20
Step 5: Create a vehicle	21
Step 6: Create a campaign	23
Step 7: Clean up	24
Next steps	24
Ingesting data to the cloud	25
Modeling vehicles	28
Signal catalogs	31

Configure signals	33
Create a signal catalog (AWS CLI)	40
Import a signal catalog	44
Update a signal catalog (AWS CLI)	54
Delete a signal catalog (AWS CLI)	56
Get signal catalog information (AWS CLI)	56
Vehicle models	57
Create a vehicle model	58
Update a vehicle model (AWS CLI)	64
Delete a vehicle model	65
Get vehicle model information (AWS CLI)	66
Decoder manifests	67
Configure network interfaces and decoder signals	69
Create a decoder manifest	72
Update a decoder manifest (AWS CLI)	79
Delete a decoder manifest	80
Get decoder manifest information (AWS CLI)	81
Vehicles	83
Provision vehicles	84
Authenticate vehicles	85
Authorize vehicles	86
Reserved topics	88
Create a vehicle	89
Create a vehicle (console)	90
Create a vehicle (AWS CLI)	92
Create multiple vehicles (AWS CLI)	94
Update a vehicle (AWS CLI)	95
Update multiple vehicles (AWS CLI)	96
Delete a vehicle	97
Delete a vehicle (console)	98
Delete a vehicle (AWS CLI)	98
Get vehicle information (AWS CLI)	98
Fleets	100
Create a fleet (AWS CLI)	101
Associate a vehicle with a fleet (AWS CLI)	102
Disassociate a vehicle from a fleet (AWS CLI)	102

Update a fleet (AWS CLI)	103
Delete a fleet (AWS CLI)	103
Get fleet information (AWS CLI)	103
Campaigns	105
Create a campaign	110
Create a campaign (console)	110
Create a campaign (AWS CLI)	118
Logical expressions for campaigns	121
Update a campaign (AWS CLI)	122
Delete a campaign	123
Delete a campaign (console)	123
Delete a campaign (AWS CLI)	123
Get campaign information (AWS CLI)	124
Processing and visualizing vehicle data	125
Processing vehicle data in Timestream	125
Visualizing vehicle data stored in Timestream	126
Processing vehicle data in S3	126
S3 object format	127
Analyzing vehicle data stored in S3	127
AWS CLI and AWS SDKs	130
Troubleshooting	131
Decoder manifest issues	131
Edge Agent for AWS IoT FleetWise software issues	134
Issue: The Edge Agent software doesn't start.	135
Issue: [ERROR] [IoT FleetWise Engine::connect]: [Failed to init persistency library]	136
Issue: The Edge Agent software doesn't collect on-board diagnostics (OBD) II PIDs and diagnostic trouble codes (DTCs).	136
Issue: The Edge Agent for AWS IoT FleetWise software doesn't collect data from the network or isn't able to apply data inspection rules.	137
Issue: [ERROR] [AwsIotConnectivityModule::connect]: [Connection failed with error] or [WARN] [AwsIotChannel::send]: [No alive MQTT Connection.]	138
Security	139
Data protection	140
Encryption at rest	141
Encryption in transit	141
Data encryption	141

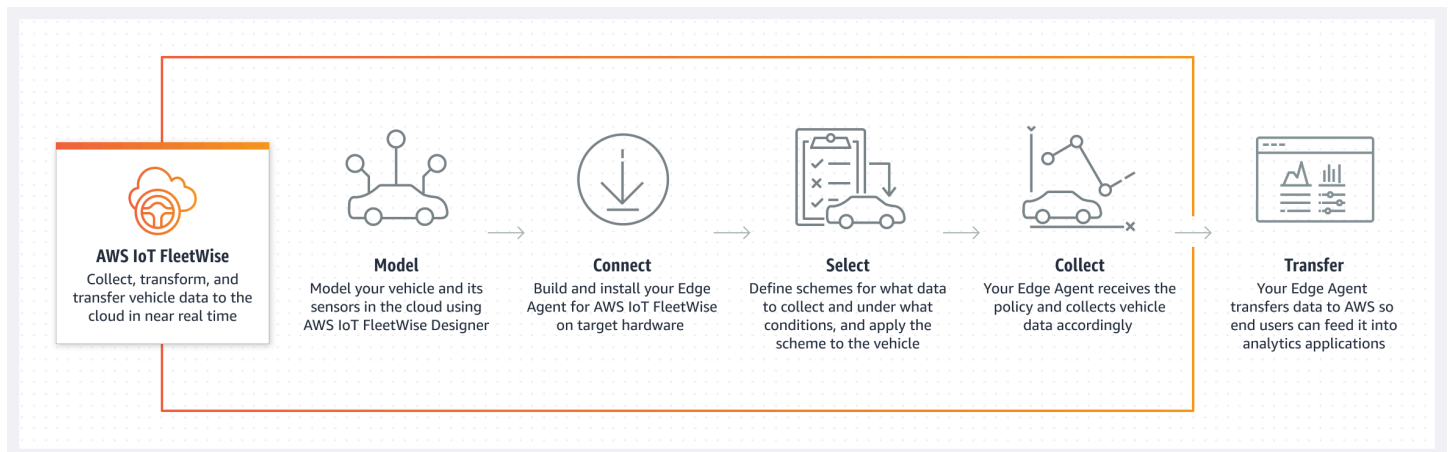
Controlling access	149
Grant AWS IoT FleetWise access to an Amazon S3 destination	150
Grant AWS IoT FleetWise access to a Amazon Timestream destination	153
Identity and Access Management	155
Audience	156
Authenticating with identities	157
Managing access using policies	160
How AWS IoT FleetWise works with IAM	162
Identity-based policy examples	171
Troubleshooting	175
Compliance Validation	177
Resilience	178
Infrastructure security	179
Connecting to AWS IoT FleetWise through an interface VPC endpoint	179
Configuration and vulnerability analysis	182
Security best practices	183
Grant minimum possible permissions	183
Don't log sensitive information	183
Use AWS CloudTrail to view API call history	183
Keep your device clock in sync	184
Monitoring	185
Monitoring with CloudWatch	185
Monitoring with CloudWatch Logs	189
Viewing AWS IoT FleetWise logs in the CloudWatch console	189
Configuring logging	194
CloudTrail logs	197
AWS IoT FleetWise information in CloudTrail	197
Understanding AWS IoT FleetWise log file entries	198
Document history	200

What is AWS IoT FleetWise?

AWS IoT FleetWise is a managed service that you can use to collect vehicle data and organize it in the cloud. You can use the collected data to improve vehicle quality, performance, and autonomy. With AWS IoT FleetWise, you can collect and organize data from vehicles that use different protocols and data formats. AWS IoT FleetWise helps to transform low-level messages into human-readable values and standardize the data format in the cloud for data analyses. You can also define data collection campaigns to control what vehicle data to collect and when to transfer that data to the cloud.

When the vehicle data is in the cloud, you can use it for applications that analyze vehicle fleet health. This data can help you to identify potential maintenance issues, make in-vehicle infotainment systems smarter, and improve advanced technologies like autonomous driving and driver-assistance systems with analytics and machine learning (ML).

The following diagram shows the basic architecture of AWS IoT FleetWise.



Topics

- [Benefits](#)
- [Use cases](#)
- [Are you new to AWS IoT FleetWise?](#)
- [Accessing AWS IoT FleetWise](#)
- [Pricing for AWS IoT FleetWise](#)
- [How AWS IoT FleetWise works](#)
- [Related services](#)

Benefits

The key benefits of AWS IoT FleetWise are:

Collect vehicle data more intelligently

Improve data relevance with intelligent data collection that sends only the data you need to the cloud for analysis.

Easily analyze standardized, fleet-wide data

Analyze standardized data from a fleet of vehicles without needing to develop a custom data collection or logging system.

Automatic data synchronization in the cloud

Gain a unified view of data collected from both standard sensors (telemetry data) and vision systems (data from cameras, radars, and lidars), and keep it automatically synchronized in the cloud. AWS IoT FleetWise keeps both structured and unstructured vision system data, metadata, and standard sensor data automatically synchronized in the cloud. This streamlines the process to assemble a full picture view of events and gain insights.

Note

Vision system data is in preview release and is subject to change.

Use cases

The scenarios in which you can use AWS IoT FleetWise include the following:

Train AI/ML models

Continuously improve machine learning models used for autonomous and advanced driver assistance systems by collecting data from production vehicles.

Enhance the digital customer experience

Use data from infotainment systems to make in-vehicle audiovisual content and in-app insights more relevant.

Maintain vehicle fleet health

Use insights from fleet data to monitor EV battery health and charge levels, manage maintenance schedules, analyze fuel consumption, and more.

Are you new to AWS IoT FleetWise?

If you're new to AWS IoT FleetWise, we recommend that you begin by reading the following sections:

- [How AWS IoT FleetWise works](#)
- [Setting up AWS IoT FleetWise](#)
- [Edge Agent software demo](#)
- [Ingesting data to the cloud](#)

Accessing AWS IoT FleetWise

You can use the AWS IoT FleetWise console or API to access AWS IoT FleetWise.

Pricing for AWS IoT FleetWise

Vehicles send data to the cloud through MQTT messages. You pay at the end of each month for the vehicles that you created in AWS IoT FleetWise. You also pay for messages that you collect from vehicles. For current information about pricing, see the [AWS IoT FleetWise Pricing](#) page. To learn more about the MQTT messaging protocol, see [MQTT](#) in the *AWS IoT Core Developer Guide*.

How AWS IoT FleetWise works

The following sections provide an overview of AWS IoT FleetWise service components and how they interact.

After you read this introduction, see the [Setting up AWS IoT FleetWise](#) section to learn how to set up AWS IoT FleetWise.

Topics

- [Key concepts](#)

- [Features of AWS IoT FleetWise](#)

Key concepts

AWS IoT FleetWise provides a vehicle modeling framework for you to model your vehicle and its sensors and actuators in the cloud. To enable the secure communication between your vehicle and the cloud, AWS IoT FleetWise also provides a reference implementation to help you develop Edge Agent software that you can install in your vehicle. You can define data collection schemes in the cloud and deploy them to your vehicle. The Edge Agent software running in your vehicle uses data collection schemes to control what data to collect and when to transfer it to the cloud.

The following are the core concepts of AWS IoT FleetWise.

Signal

Signals are fundamental structures that you define to contain vehicle data and its metadata. A signal can be an attribute, a branch, a sensor, or an actuator. For example, you can create a sensor to receive in-vehicle temperature values, and to store its metadata, including a sensor name, a data type, and a unit. For more information, see [Create and manage signal catalogs](#).

Attribute

Attributes represent static information that generally doesn't change, such as manufacturer and manufacturing date.

Branch

Branches represent signals in a nested structure. Branches demonstrate signal hierarchies. For example, the Vehicle branch has a child branch, Powertrain. The Powertrain branch has a child branch, combustionEngine. To locate the combustionEngine branch, use the `Vehicle.Powertrain.combustionEngine` expression.

Sensor

Sensor data reports the current state of the vehicle and change over time, as the state of the vehicle changes, such as fluid levels, temperatures, vibrations, or voltage.

Actuator

Actuator data reports the state of a vehicle device, such as motors, heaters, and door locks. Changing the state of a vehicle device can update actuator data. For example, you can define

an actuator to represent the heater. The actuator receives new data when you turn on or off the heater.

Custom structure

A custom structure (also known as a struct) represents a complex or higher-order data structure. It facilitates logical binding or grouping of data that originates from the same source. A struct is used when data is read or written in an atomic operation, such as to represent a complex data type or higher-order shape.

A signal of struct type is defined in the signal catalog using a reference to a struct data type instead of a primitive data type. Structs can be used for all types of signals including sensors, attributes, actuators, and vision system data types. If a signal of struct type is sent or received, AWS IoT FleetWise expects all included items to have valid values, so all items are mandatory. For example, if a struct contains the items `Vehicle.Camera.Image.height`, `Vehicle.Camera.Image.width`, and `Vehicle.Camera.Image.data` – it's expected that the sent signal contains values for all of these items.

Note

Vision system data is in preview release and is subject to change.

Custom property

A custom property represents a member of the complex data structure. The data type of the property can be either primitive or another struct.

When representing a higher-order shape using a struct and custom property, the intended higher-order shape is always defined and visioned as a tree structure. The custom property is used to define all the leaf nodes while the struct is used to define all the non-leaf nodes.

Signal catalog

A signal catalog contains a collection of signals. Signals in a signal catalog can be used to model vehicles that use different protocols and data formats. For example, there are two cars made by different automakers: one uses the Control Area Network (CAN bus) protocol; the other one uses the On-board Diagnostics (OBD) protocol. You can define a sensor in the signal catalog to receive in-vehicle temperature values. This sensor can be used to represent the thermocouples in both cars. For more information, see [Create and manage signal catalogs](#).

Vehicle model (model manifest)

Vehicle models are declarative structures that you can use to standardize the format of your vehicles and to define relationships between signals in the vehicles. Vehicle models enforce consistent information across multiple vehicles of the same type. You add signals to create vehicle models. For more information, see [Create and manage vehicle models](#).

Decoder manifest

Decoder manifests contain decoding information for each signal in vehicle models. Sensors and actuators in vehicles transmit low-level messages (binary data). With decoder manifests, AWS IoT FleetWise is able to transform binary data into human-readable values. Every decoder manifest is associated with a vehicle model. For more information, see [Create and manage decoder manifests](#).

Network interface

Contains information about the protocol that the in-vehicle network uses. AWS IoT FleetWise supports the following protocols.

Controller Area Network (CAN bus)

A protocol that defines how data is communicated between electronic control units (ECUs). ECUs can be the engine control unit, airbags, or the audio system.

On-board diagnostic (OBD) II

A further developed protocol that defines how self-diagnostic data is communicated between ECUs. It provides a number of standard diagnostic trouble codes (DTCs) that help identify what is wrong with your vehicle.

Vehicle middleware

The vehicle middleware defined as a type of network interface. Examples of vehicle middleware include Robot Operating System (ROS 2) and Scalable service-Oriented MiddlewarE over IP (SOME/IP).

Note

AWS IoT FleetWise supports ROS 2 middleware for vision system data.

Decoder signal

Provides detailed decoding information for a specific signal. Every signal specified in the vehicle model must be paired with a decoder signal. If the decoder manifest contains CAN network interfaces, it must contain CAN decoder signals. If the decoder manifest contains OBD network interfaces, it must contain OBD decoder signals.

The decoder manifest must contain message decoder signals if it also contains vehicle middleware interfaces.

Vehicle

A virtual representation of your physical vehicle, such a car or a truck. Vehicles are instances of vehicle models. Vehicles created from the same vehicle model inherit the same group of signals. Each vehicle corresponds to an AWS IoT thing.

Fleet

A fleet represents a group of vehicles. Before you can easily manage a fleet of vehicles, you must associate individual vehicles to a fleet.

Campaign

Contains data collection schemes. You define a campaign in the cloud and deploy it to a vehicle or fleet. Campaigns give the Edge Agent software instructions on how to select, collect, and transfer data to the cloud.

Data collection scheme

Data collection schemes give the Edge Agent software instructions on how to collect data. Currently, AWS IoT FleetWise supports the condition-based collection scheme and the time-based collection scheme.

Condition-based collection scheme

Use a logical expression to recognize what data to collect. The Edge Agent software collects data when the condition is met. For example, if the expression is `$variable.myVehicle.InVehicleTemperature >35.0`, the Edge Agent software collects temperature values that are greater than 35.0.

Time-based collection scheme

Specify a time period in milliseconds to define how often to collect data. For example, if the time period is 10,000 milliseconds, the Edge Agent software collects data once every 10 seconds.

Features of AWS IoT FleetWise

The following are the key features of AWS IoT FleetWise.

Vehicle modeling

Build virtual representations of your vehicles and apply a common format to organize vehicle signals. AWS IoT FleetWise supports [Vehicle Signal Specification \(VSS\)](#) that you can use to standardize vehicle signals.

Scheme-based data collection

Define schemes to transfer only high-value vehicle data to the cloud. You can define condition-based schemes to control what data to collect, such as data in-vehicle temperature values that are greater than 40 degrees. You can also define time-based schemes to control how often to collect data.

Edge Agent for AWS IoT FleetWise software

The Edge Agent software running in vehicles facilitates communication between vehicles and the cloud. While vehicles are connected to the cloud, the Edge Agent software continually receives data collection schemes and collects data accordingly.

Related services

AWS IoT FleetWise integrates with the following AWS services to improve the availability and scalability of your cloud solutions.

- **AWS IoT Core** – Register and control AWS IoT devices that upload vehicle data to AWS IoT FleetWise. For more information, see [What is AWS IoT](#) in the *AWS IoT Developer Guide*.
- **Amazon Timestream** – Use a time series database to store and analyze your vehicle data. For more information, see [What is Amazon Timestream](#) in the *Amazon Timestream Developer Guide*.
- **Amazon S3** – Use an object storage service to store and manage your vehicle data. For more information, see [What is Amazon S3](#) in the *Amazon Simple Storage Service User Guide*.

Setting up AWS IoT FleetWise

Before you use AWS IoT FleetWise for the first time, complete the steps in the following sections.

Topics

- [Set up your AWS account](#)
- [Get started in the console](#)
- [Configuring settings](#)

Set up your AWS account

Complete the following tasks to sign up for AWS and create an administrative user.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform [tasks that require root user access](#).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see [Enable a virtual MFA device for your AWS account root user \(console\)](#) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see [Configure user access with the default IAM Identity Center directory](#) in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Note

You can use a service-linked role with AWS IoT FleetWise. Service-linked roles are predefined by AWS IoT FleetWise and include the permissions that AWS IoT FleetWise needs to send metrics to Amazon CloudWatch. For more information, see [Using service-linked roles for AWS IoT FleetWise](#).

Get started in the console

If you aren't already signed in to your AWS account, sign in, then open the [AWS IoT FleetWise console](#). To get started with AWS IoT FleetWise, create a vehicle model. A vehicle model standardizes the format of your vehicles.

1. Navigate to the [AWS IoT FleetWise console](#).
2. In **Get started with AWS IoT FleetWise**, choose **Get started**.

For more information about creating a vehicle model, see [Create a vehicle model \(console\)](#).

Configuring settings

You can use the AWS IoT FleetWise console or API to configure settings for Amazon CloudWatch Logs metrics, Amazon CloudWatch Logs, and encrypt data with an AWS managed key.

With CloudWatch metrics, you can monitor AWS IoT FleetWise and other AWS resources. You can use CloudWatch metrics to collect and track metrics, such as to determine if there is an exceeded

service limit. For more information about CloudWatch metrics, see [Monitoring AWS IoT FleetWise with Amazon CloudWatch](#).

With CloudWatch Logs, AWS IoT FleetWise sends log data to a CloudWatch log group, where you can use it to identify and mitigate any issues. For more information about CloudWatch Logs, see [Configure AWS IoT FleetWise logging](#).

With data encryption, AWS IoT FleetWise uses AWS managed keys to encrypt data. You can also choose to create and manage keys with AWS KMS. For more information about encryption, see [Data encryption](#).

Configure settings (console)

If you aren't already signed in to your AWS account, sign in, then open the [AWS IoT FleetWise console](#).

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the left pane, choose **Settings**.
3. In **Metrics**, choose **Enable**. AWS IoT FleetWise automatically attaches a CloudWatch managed policy to the service-linked role and enables CloudWatch metrics.
4. In **Logging**, choose **Edit**.
 - a. In the **CloudWatch logging** section, enter the **Log group**.
 - b. To save your changes, choose **Submit**.
5. In the **Encryption** section, choose **Edit**.
 - a. Choose the type of key that you want to use. For more information, see [Key management](#).
 - i. **Use AWS key** – AWS IoT FleetWise owns and manages the key.
 - ii. **Choose a different AWS Key Management Service key** – You manage AWS KMS keys that are in your account.
 - b. To save your changes, choose **Submit**.

Configure settings (AWS CLI)

In the AWS CLI, register the account to configure settings.

1. To configure settings, run the following command.

```
aws iotfleetwise register-account
```

2. To verify your settings, run the following command to retrieve the registration status.

Note

The service-linked role is only used to publish AWS IoT FleetWise metrics to CloudWatch. For more information, see [Using service-linked roles for AWS IoT FleetWise](#).

```
aws iotfleetwise get-register-account-status
```

Example response

```
{
  "accountStatus": "REGISTRATION_SUCCESS",
  "creationTime": "2022-07-28T11:31:22.603000-07:00",
  "customerAccountId": "012345678912",
  "iamRegistrationResponse": {
    "errorMessage": "",
    "registrationStatus": "REGISTRATION_SUCCESS",
    "roleArn": "arn:aws:iam::012345678912:role/AWSIoT FleetwiseServiceRole"
  },
  "lastModificationTime": "2022-07-28T11:31:22.854000-07:00",
}
```

The registration status can be one of the following:

- REGISTRATION_SUCCESS – The AWS resource is successfully registered.
- REGISTRATION_PENDING – AWS IoT FleetWise is processing the registration request. This process takes approximately five minutes to complete.
- REGISTRATION_FAILURE – AWS IoT FleetWise can't register the AWS resource. Try again later.

Getting started with AWS IoT FleetWise

With AWS IoT FleetWise, you can collect, transform, and transfer your vehicle data. Use the tutorials in this section to get started with AWS IoT FleetWise.

See the following topics to learn more about AWS IoT FleetWise:

- [Ingesting data to the cloud](#)
- [Modeling vehicles](#)
- [Create, provision, and manage vehicles](#)
- [Create and manage fleets](#)
- [Collect and transfer data with campaigns](#)

Requirements

You must have an AWS account to get started with AWS IoT FleetWise. If you don't have one, see [Setting up AWS IoT FleetWise](#).

Use a Region where AWS IoT FleetWise is available. For more information, see [AWS IoT FleetWise endpoints and quotas](#). You can use the Region selector in the AWS Management Console to switch to one of these Regions.

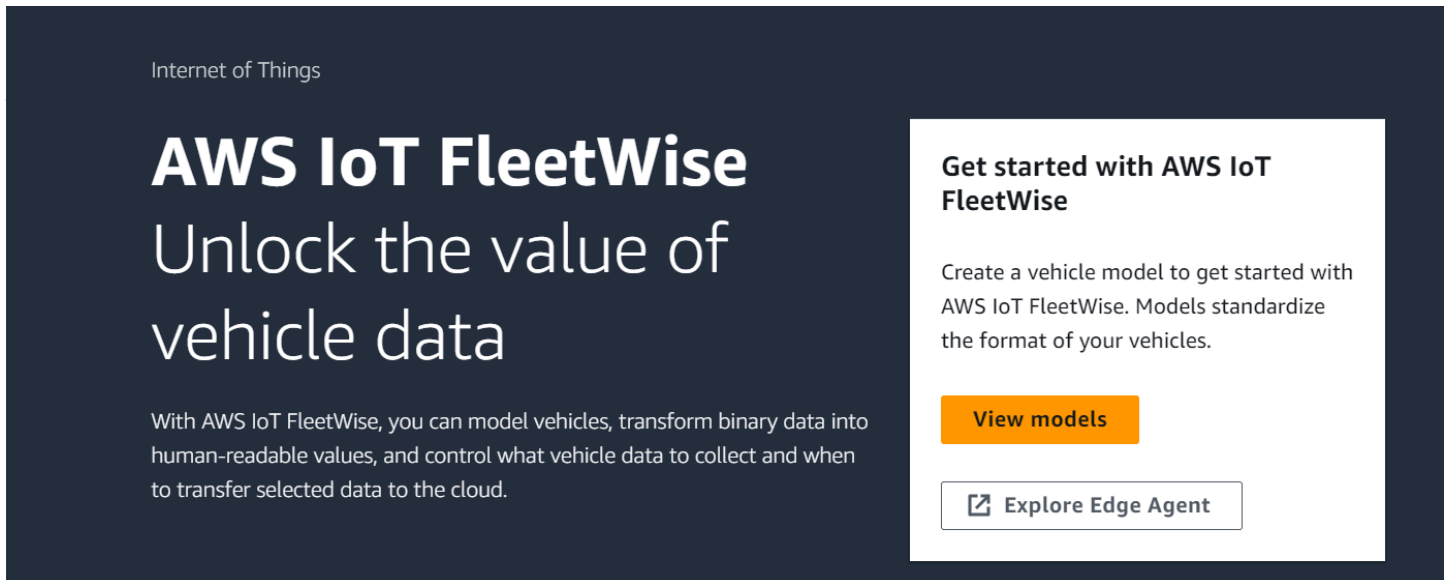
Edge Agent software demo

You can use the Explore Edge Agent quick start demo to explore AWS IoT FleetWise and learn how to develop Edge Agent software for AWS IoT FleetWise. This demo uses an AWS CloudFormation template. It walks you through reviewing the Edge Agent reference implementation, developing your Edge Agent, and then deploying your Edge Agent software on an Amazon EC2 Graviton and generating sample vehicle data. The demo also provides a script that you can use to create a signal catalog, a vehicle model, a decoder manifest, a vehicle, a fleet, and a campaign — all in the cloud. For more information about the quick start demo, do the following to download the *Edge Agent software Developer Guide*.

To download the quick start demo

1. Navigate to the [AWS IoT FleetWise console](#).

2. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.



Internet of Things

AWS IoT FleetWise

Unlock the value of vehicle data

With AWS IoT FleetWise, you can model vehicles, transform binary data into human-readable values, and control what vehicle data to collect and when to transfer selected data to the cloud.

Get started with AWS IoT FleetWise

Create a vehicle model to get started with AWS IoT FleetWise. Models standardize the format of your vehicles.

[View models](#)

[Explore Edge Agent](#)

Tutorial: Getting started with AWS IoT FleetWise (console)

Use AWS IoT FleetWise to collect, transform, and transfer the unique data format from automated vehicles to the cloud in near real time. You have access to fleet-wide insights. This can help you to efficiently detect and mitigate issues in vehicle health, transfer high-value data signals, and remotely diagnose problems, all while reducing costs.

This tutorial shows you how to get started with AWS IoT FleetWise. You'll learn how to create a vehicle model (model manifest), a decoder manifest, a vehicle, and a campaign.

For more information about the key components and concepts of AWS IoT FleetWise, see [How AWS IoT FleetWise works](#).

Estimated time: About 45 minutes.

Important

You will be charged for the AWS IoT FleetWise resources that this demo creates and consumes. For more information, see [AWS IoT FleetWise](#) in the *AWS IoT FleetWise Pricing* page.

Topics

- [Prerequisites](#)
- [Step 1: Set up the Edge Agent software for AWS IoT FleetWise](#)
- [Step 2: Create a vehicle model](#)
- [Step 3: Create a decoder manifest](#)
- [Step 4: Configure a decoder manifest](#)
- [Step 5: Create a vehicle](#)
- [Step 6: Create a campaign](#)
- [Step 7: Clean up](#)
- [Next steps](#)

Prerequisites

To complete this getting started tutorial, you first need the following:

- An AWS account. If you don't have an AWS account, see [Creating an AWS account](#) in the *AWS Account Management Reference Guide*.
- Access to an AWS Region that supports AWS IoT FleetWise. Currently, AWS IoT FleetWise is supported in US East (N. Virginia) and Europe (Frankfurt).
- Amazon Timestream resources:
 - An Amazon Timestream database. For more information, see [Create a database](#) in the *Amazon Timestream Developer Guide*.
 - An Amazon Timestream table created in Amazon Timestream that will hold your data. For more information, see [Create a table](#) in the *Amazon Timestream Developer Guide*.

Step 1: Set up the Edge Agent software for AWS IoT FleetWise

Note

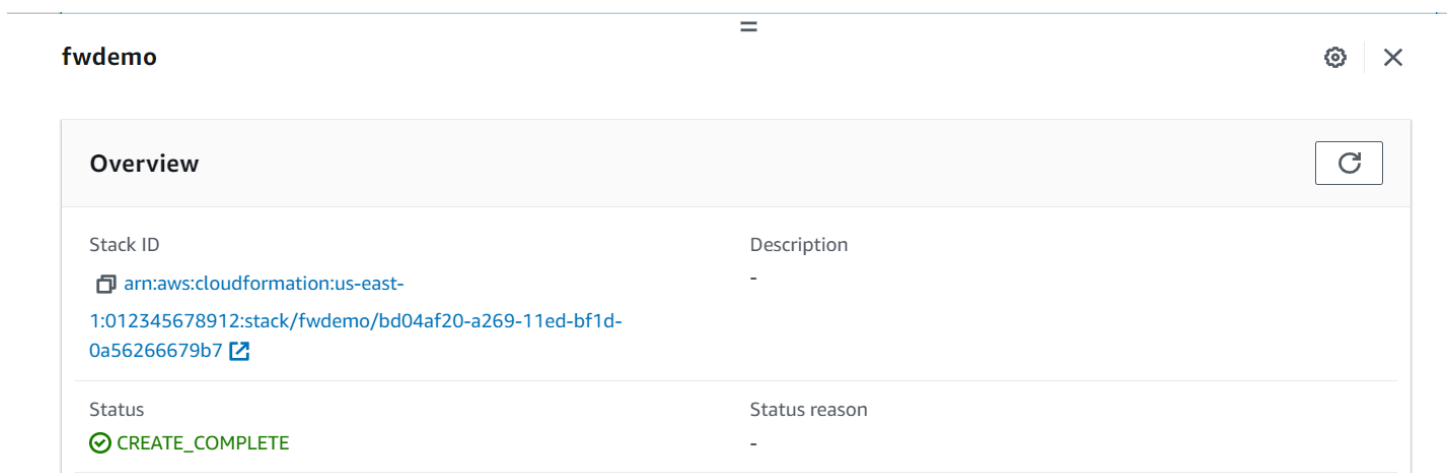
The CloudFormation stack in this step uses telemetry data. You can also create a CloudFormation stack using vision system data. For more information, see the [Vision System Data Developer Guide](#).

Vision system data is in preview release and is subject to change.

Your Edge Agent software for AWS IoT FleetWise facilitates communication between vehicles and the cloud. It receives instructions from data collection schemes on how to collect data from cloud-connected vehicles.

To set up your Edge Agent software, in **General information**, do the following:

1. Open the [Launch CloudFormation Template](#).
2. On the **Quick create stack** page, for **Stack name**, enter the name of your stack of AWS IoT FleetWise resources. A stack is a friendly name that appears as a prefix on the names of the resources this AWS CloudFormation template creates.
3. Under **Parameters**, enter your custom values for the parameters related to your stack.
 - a. **Fleetsize** - You can increase the number of vehicles in your fleet by updating the Fleetsize parameter.
 - b. **IoTCoreRegion** - You can specify the Region where the AWS IoT thing is created by updating the IoTCoreRegion parameter. You must use the same Region that you used to create your AWS IoT FleetWise vehicles. For more information about AWS Regions, see [Regions and Zones - Amazon Elastic Compute Cloud](#).
4. In the **Capabilities** section, select the box to acknowledge that AWS CloudFormation creates IAM resources.
5. Choose **Create stack**, then wait approximately 15 minutes for the status of the stack to display CREATE_COMPLETE.
6. To confirm the stack was created, choose the **Stack info** tab, refresh the view, and look for CREATE_COMPLETE.



The screenshot shows the AWS CloudFormation console interface. At the top, the stack name 'fwdemo' is displayed. Below it, the 'Overview' tab is active, showing a refresh button. The main content area contains two rows of information:

Stack ID	Description
arn:aws:cloudformation:us-east-1:012345678912:stack/fwdemo/bd04af20-a269-11ed-bf1d-0a56266679b7	-
Status	Status reason
✔ CREATE_COMPLETE	-

⚠ Important

You will be charged for the AWS IoT FleetWise resources that this demo creates and consumes. For more information, see [AWS IoT FleetWise](#) in the *AWS IoT FleetWise Pricing* page.

Step 2: Create a vehicle model

⚠ Important

You can't create a vehicle model with vision system data signals in the AWS IoT FleetWise console. Instead, use the AWS CLI.

You use vehicle models to standardize the format of your vehicles, and to help define the relationship between signals in the vehicles that you create. A *signal catalog* is also created when you create a vehicle model. A signal catalog is a collection of standardized signals that can be reused to create vehicle models. Signals are fundamental structures that you define to contain vehicle data and its metadata. At this time, the AWS IoT FleetWise service supports only one signal catalog per AWS Region per account. This helps to verify that data processed from a fleet of vehicles is consistent.

To create a vehicle model

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models**.
3. On the **Vehicle models** page, choose **Create vehicle model**.
4. In the **General information** section, enter the name of your vehicle model, such as Vehicle1, and an optional description. Then choose **Next**.
5. Choose one or more signals from the signal catalog. You can filter signals by name in the search catalog, or choose them from the list. For example, you can choose signals for tire pressure and brake pressure so that you can collect data related to these signals. Choose **Next**.
6. Choose your .dbc files and upload them from your local device. Choose **Next**.

Note

For this tutorial, you can download a [sample .dbc file](#) to upload for this step.

7. Add attributes to your vehicle model and then choose **Next**.
 - a. **Name** - Enter the name of the vehicle attribute, such as the manufacturer name or manufacturing date.
 - b. **Data Type** - On the **Data type** menu, choose a data type.
 - c. **Unit** - (Optional) Enter a unit value, such as kilometer or Celsius.
 - d. **Path** - (Optional) Enter a name for the path to a signal, such as `Vehicle.Engine.Light`. The dot (.) indicates that it is a child signal.
 - e. **Default value** - (Optional) Enter a default value.
 - f. **Description** - (Optional) Enter a description of the attribute.
8. Review your configurations. When you're ready, choose **Create**. A notification appears saying your vehicle model was successfully created.

✔ Vehicle model created
✕

You successfully created the vehicle model: demo.

AWS IoT FleetWise > Vehicle models > Demo

demo

Duplicate
Create vehicle
Create decoder manifest

When a decoder manifest is associated with a vehicle model, you can create a vehicle. To use the API to create vehicles with this vehicle model, follow the instructions in the AWS IoT FleetWise Developer Guide. After you create vehicles, you can create campaigns for them.

Summary [Info](#)

Vehicle model ARN 📄 <code>arn:aws:iotfleetwise:us-east-1:012345678912:model-manifest/demo</code>	Status ✔ ACTIVE	Date created February 01, 2023 at 14:40 (UTC-05)
Signal catalog ARN 📄 <code>arn:aws:iotfleetwise:us-east-1:012345678912:signal-catalog/DefaultSignalCatalog</code>	Description -	Last modified February 01, 2023 at 14:40 (UTC-05)

Step 3: Create a decoder manifest

Decoder manifests are associated with the vehicle models that you create. They contain information that helps AWS IoT FleetWise decode and transform vehicle data from a binary format into human-readable values that can be analyzed. Network interfaces and decoder signals are components that help configure decoder manifests. A network interface contains information about the CAN or OBD protocol that your vehicle network uses. The decoder signal provides decoding information for a specific signal.

To create a decoder manifest

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Vehicle models**.
3. In the **Vehicle models** section, choose the vehicle model that you want to use to create a decoder manifest.
4. Choose **Create decoder manifest**.

Step 4: Configure a decoder manifest

To configure a decoder manifest

Important

You can't configure vision system data signals in decoder manifests using the AWS IoT FleetWise console. Instead, use the AWS CLI. For more information, see [Create a decoder manifest \(AWS CLI\)](#).

1. To help you identify your decoder manifest, enter a name and an optional description for it. Then, choose **Next**.
2. To add one or more network interfaces, choose either the `CAN_INTERFACE` or the `OBD_INTERFACE` type.
 - **On-board diagnostic (OBD) interface** - Choose this interface type if you want a protocol that defines how self-diagnostic data is communicated between electronic control units (ECUs). This protocol provides a number of standard diagnostic trouble codes (DTCs) that can help you troubleshoot problems with your vehicle.

- **Controller Area Network (CAN bus) interface** -Choose this interface type if you want a protocol that defines how data is communicated between ECUs. ECUs can be engine control units, airbags, or the audio system.
3. Enter a network interface name.
 4. To add signals to the network interface, choose one or more signals from the list.
 5. Choose a decoder signal for the signal you added in the previous step. To provide decoding information, upload a .dbc file. Each signal in the vehicle model must be paired with a decoder signal that you can choose from the list.
 6. To add another network interface, choose **Add network interface**. When you're done adding network interfaces, choose **Next**.
 7. Review your configurations and then choose **Create**. A notification appears saying your decoder manifest was successfully created.

Step 5: Create a vehicle

In AWS IoT FleetWise, vehicles are virtual representations of your real-life, physical vehicle. All vehicles created from the same vehicle model inherit the same group of signals, and each vehicle that you create corresponds to a newly created IoT thing. You must associate all vehicles with a decoder manifest.

Prerequisites

1. Verify that you've already created the vehicle model and decoder manifest. Also, verify that the status of the vehicle model is **ACTIVE**.
 - a. To verify that the status of the vehicle model is **ACTIVE**, open the AWS IoT FleetWise console.
 - b. On the navigation pane, choose **Vehicle models**.
 - c. In the **Summary** section, under **Status**, check the status of your vehicle.

✔ **Vehicle model created**
✕

You successfully created the vehicle model: demo.

AWS IoT FleetWise > Vehicle models > Demo

demo

Duplicate
Create vehicle
Create decoder manifest

When a decoder manifest is associated with a vehicle model, you can create a vehicle. To use the API to create vehicles with this vehicle model, follow the instructions in the AWS IoT FleetWise Developer Guide. After you create vehicles, you can create campaigns for them.

Summary [Info](#)

<p>Vehicle model ARN</p> <p> <code>arn:aws:iotfleetwise:us-east-1:012345678912:model-manifest/demo</code></p>	<p>Status</p> <p>✔ ACTIVE</p>	<p>Date created</p> <p>February 01, 2023 at 14:40 (UTC-05)</p>
<p>Signal catalog ARN</p> <p> <code>arn:aws:iotfleetwise:us-east-1:012345678912:signal-catalog/DefaultSignalCatalog</code></p>	<p>Description</p> <p>-</p>	<p>Last modified</p> <p>February 01, 2023 at 14:40 (UTC-05)</p>

To create a vehicle

1. Open the AWS FleetWise console.
2. On the navigation pane, choose **Vehicles**.
3. Choose **Create vehicle**.
4. To define the vehicle properties, enter the vehicle name, and then choose a model manifest (vehicle model) and a decoder manifest.
5. (Optional) To define the vehicle attributes, enter a key-value pair and then choose **Add attributes**.
6. (Optional) To label your AWS resource, add tags and then choose **Add new tag**.
7. Choose **Next**.
8. To configure the vehicle certificate, you can either upload your own certificate or choose **Auto-generate a new certificate**. We recommend auto-generating your certificate for a quicker setup. If you already have a certificate, you can choose to use it instead.
9. Download the public and private key files and then choose **Next**.
10. To attach a policy to the vehicle certificate, you can either enter an existing policy name or create a new policy. To create a new policy, choose **Create policy** and then choose **Next**.
11. Review your configurations. When you're done, choose **Create vehicle**.

Step 6: Create a campaign

In AWS IoT FleetWise, campaigns are used to facilitate the selection, collection, and transfer of data from vehicles to the cloud. Campaigns contain data collection schemes that give the Edge Agent software instructions on how to collect data with a condition-based collection scheme or a time-based collection scheme.

To create a campaign

1. Open the AWS IoT FleetWise console.
2. On the navigation pane, choose **Campaigns**.
3. Choose **Create campaign**.
4. Enter your campaign name and an optional description.
5. To configure your campaign's data collection scheme, you can manually define the data collection scheme or upload a .json file from your local device. Uploading a .json file automatically defines the data collection scheme.
 - a. To manually define the data collection scheme, choose **Define Data Collection Scheme** and choose the type of data collection scheme you want to use for your campaign. You can choose either a **Condition-based** collection scheme or **Time-based** collection scheme.
 - b. If you choose a **Time-based** collection scheme, you must specify the duration of time that your campaign will collect the vehicle data.
 - c. If you choose a condition-based collection scheme, you must specify an expression to recognize what data to collect. Be sure to specify the signal's name as a variable, a comparison operator, and a comparison value.
 - d. (Optional) Choose the language version of your expression, or keep it as the default value of 1.
 - e. (Optional) Specify the trigger interval between two data collection events.
 - f. To collect data, choose the **Trigger** mode condition for the Edge Agent software. By default, the Edge Agent for AWS IoT FleetWise software **Always** collects data whenever the condition is met. Or, it can collect data only when the condition is met for the first time, **On first trigger**.
 - g. (Optional) You can choose more advanced scheme options.
6. To specify the signals that the data collection scheme will collect data from, search for the name of the signal from the menu.

7. (Optional) You can choose a maximum sample count or minimum sampling interval. You can also add more signals.
8. Choose **Next**.
9. Define the storage destination that you want the campaign to transfer data to. You can store data in Amazon S3 or Amazon Timestream.
 - a. Amazon S3 – Choose the S3 bucket that AWS IoT FleetWise has permissions to.
 - b. Amazon Timestream – choose the Timestream database and table name. Enter an IAM role that allows AWS IoT FleetWise to send data to Timestream.
10. Choose **Next**.
11. Choose vehicle attributes or vehicle names from the search box.
12. Enter the value related to the attribute or name that you chose for your vehicle.
13. Choose the vehicles that your campaign will collect data from. Then, choose **Next**.
14. Review the configurations of your campaign and then choose **Create campaign**. You or your team must deploy the campaign to vehicles.

Step 7: Clean up

To avoid further charges for the resources you used during this tutorial, delete the AWS CloudFormation stack and all stack resources.

To delete the AWS CloudFormation stack

1. Open the [AWS CloudFormation console](#).
2. From the list of **Stacks**, choose the stack that you created in step 1.
3. Choose **Delete**.
4. To confirm deletion, choose **Delete**. The stack takes around 15 minutes to delete.

Next steps

1. You can process and visualize the vehicle data that your campaign collects. For more information, see [Processing and visualizing vehicle data](#).
2. You can troubleshoot and resolve issues with AWS IoT FleetWise. For more information, see [Troubleshooting AWS IoT FleetWise](#).

Ingesting data to the cloud

The Edge Agent for AWS IoT FleetWise software, when installed and running in vehicles, is designed to facilitate secure communication between your vehicles and the cloud.

Note

- AWS IoT FleetWise is not intended for use in, or in association with, the operation of any hazardous environments or critical systems that may lead to serious bodily injury or death or cause environmental or property damage. Vehicle data collected through your use of AWS IoT FleetWise is for informational purposes only, and you may not use AWS IoT FleetWise to control or operate vehicle functions.
- Vehicle data collected through your use of AWS IoT FleetWise should be evaluated for accuracy as appropriate for your use case, including for purposes of meeting any compliance obligations you may have under applicable vehicle safety regulations (such as safety monitoring and reporting obligations). Such evaluation should include collecting and reviewing information through other industry standard means and sources (such as reports from drivers of vehicles).

To ingest data to the cloud, do the following:

1. Develop and install your Edge Agent for AWS IoT FleetWise software in your vehicle. For more information about how to work with the Edge Agent software, do the following to download the [Edge Agent for AWS IoT FleetWise software Developer Guide](#).
 1. Navigate to the [AWS IoT FleetWise console](#).
 2. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.
2. Create or import a signal catalog containing signals that you'll use to create a vehicle model. For more information, see [Create a signal catalog \(AWS CLI\)](#) and [Import a signal catalog \(AWS CLI\)](#).

Note

- If you use the AWS IoT FleetWise console to create the first vehicle model, you don't need to manually create a signal catalog. When you create your first vehicle

model, AWS IoT FleetWise automatically creates a signal catalog for you. For more information, see [Create a vehicle model \(console\)](#).

- AWS IoT FleetWise currently supports a signal catalog for each AWS account per AWS Region.

3. Use signals in the signal catalog to create a vehicle model. For more information, see [Create a vehicle model](#).

Note

- If you use the AWS IoT FleetWise console to create a vehicle model, you can upload .dbc files to import signals. .dbc is a file format that Controller Area Network (CAN bus) databases support. After the vehicle model is created, new signals are automatically added to the signal catalog. For more information, see [Create a vehicle model \(console\)](#).
- If you use the `CreateModelManifest` API operation to create a vehicle model, you must use the `UpdateModelManifest` API operation to activate the vehicle model. For more information, see [Update a vehicle model \(AWS CLI\)](#).
- If you use the AWS IoT FleetWise console to create a vehicle model, AWS IoT FleetWise automatically activates the vehicle model for you.

4. Create a decoder manifest. The decoder manifest contains decoding information for every signal specified in the vehicle model that you created in the previous step. The decoder manifest is associated with the vehicle model that you created. For more information, see [Create and manage decoder manifests](#).

Note

- If you use the `CreateDecoderManifest` API operation to create a decoder manifest, you must use the `UpdateDecoderManifest` API operation to activate the decoder manifest. For more information, see [Update a decoder manifest \(AWS CLI\)](#).
- If you use the AWS IoT FleetWise console to create a decoder manifest, AWS IoT FleetWise automatically activates the decoder manifest for you.

5. Create vehicles from the vehicle model. Vehicles created from the same vehicle model inherit the same group of signals. You must use AWS IoT Core to provision your vehicle before you can ingest data to the cloud. For more information, see [Create, provision, and manage vehicles](#).

6. (Optional) Create a fleet to represent a group of vehicles, and then associate individual vehicles with the fleet. This helps you manage multiple vehicles at the same time. For more information, see [Create and manage fleets](#).
7. Create campaigns. Campaigns are deployed to a vehicle or a fleet of vehicles. Campaigns give the Edge Agent software instructions on how to select, collect, and transfer data to the cloud. For more information, see [Collect and transfer data with campaigns](#).

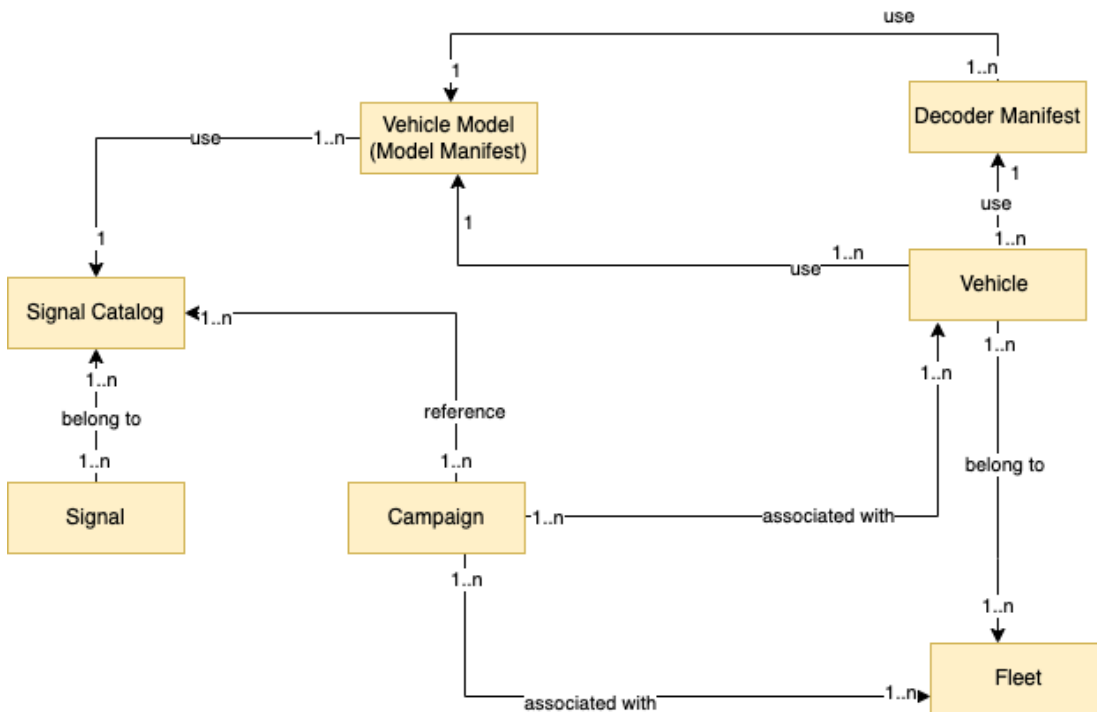
 **Note**

You must use the UpdateCampaign API operation to approve the campaign before AWS IoT FleetWise can deploy it to the vehicle or fleet. For more information, see [Update a campaign \(AWS CLI\)](#).

The Edge Agent software transfers vehicle data to AWS IoT Core using the reserved topic `$aws/iotfleetwise/vehicles/vehicleName/signals`, which sends data to to AWS IoT FleetWise. AWS IoT FleetWise then delivers the data to a Timestream table or Amazon S3 bucket. You can use Timestream to query your data, and use Amazon QuickSight or Grafana to visualize your data. For more information, see [Processing and visualizing vehicle data](#).

Modeling vehicles

AWS IoT FleetWise provides a vehicle modeling framework that you can use to build virtual representations of your vehicles in the cloud. Signals, signal catalogs, vehicle models, and decoder manifests are the core components that you work with to model your vehicles.



Signal

Signals are fundamental structures that you define to contain vehicle data and its metadata. A signal can be an attribute, a branch, a sensor, or an actuator. For example, you can create a sensor to receive in-vehicle temperature values, and to store its metadata, including a sensor name, a data type, and a unit. For more information, see [Create and manage signal catalogs](#).

Signal catalog

A signal catalog contains a collection of signals. Signals in a signal catalog can be used to model vehicles that use different protocols and data formats. For example, there are two cars made by different automakers: one uses the Control Area Network (CAN bus) protocol; the other one uses the On-board Diagnostics (OBD) protocol. You can define a sensor in the signal catalog to receive in-vehicle temperature values. This sensor can be used to represent the thermocouples in both cars. For more information, see [Create and manage signal catalogs](#).

Vehicle model (model manifest)

Vehicle models are declarative structures that you can use to standardize the format of your vehicles and to define relationships between signals in the vehicles. Vehicle models enforce consistent information across multiple vehicles of the same type. You add signals to create vehicle models. For more information, see [Create and manage vehicle models](#).

Decoder manifest

Decoder manifests contain decoding information for each signal in vehicle models. Sensors and actuators in vehicles transmit low-level messages (binary data). With decoder manifests, AWS IoT FleetWise is able to transform binary data into human-readable values. Every decoder manifest is associated with a vehicle model. For more information, see [Create and manage decoder manifests](#).

You can use the AWS IoT FleetWise console or API to model vehicles in the following way.

1. Create or import a signal catalog containing signals that you'll use to create a vehicle model. For more information, see [Create a signal catalog \(AWS CLI\)](#) and [Import a signal catalog \(AWS CLI\)](#).

Note

- If you use the AWS IoT FleetWise console to create the first vehicle model, you don't need to manually create a signal catalog. When you create your first vehicle model, AWS IoT FleetWise automatically creates a signal catalog for you. For more information, see [Create a vehicle model \(console\)](#).
- AWS IoT FleetWise currently supports a signal catalog for each AWS account per AWS Region.

2. Use signals in the signal catalog to create a vehicle model. For more information, see [Create a vehicle model](#).

Note

- If you use the AWS IoT FleetWise console to create a vehicle model, you can upload .dbc files to import signals. .dbc is a file format that Controller Area Network (CAN bus) databases support. After the vehicle model is created, new signals are automatically added to the signal catalog. For more information, see [Create a vehicle model \(console\)](#).

- If you use the `CreateModelManifest` API operation to create a vehicle model, you must use the `UpdateModelManifest` API operation to activate the vehicle model. For more information, see [Update a vehicle model \(AWS CLI\)](#).
- If you use the AWS IoT FleetWise console to create a vehicle model, AWS IoT FleetWise automatically activates the vehicle model for you.

3. Create a decoder manifest. The decoder manifest contains decoding information for every signal specified in the vehicle model that you created in the previous step. The decoder manifest is associated with the vehicle model that you created. For more information, see [Create and manage decoder manifests](#).

Note

- If you use the `CreateDecoderManifest` API operation to create a decoder manifest, you must use the `UpdateDecoderManifest` API operation to activate the decoder manifest. For more information, see [Update a decoder manifest \(AWS CLI\)](#).
- If you use the AWS IoT FleetWise console to create a decoder manifest, AWS IoT FleetWise automatically activates the decoder manifest for you.

CAN bus databases support the `.dbc` file format. You might upload `.dbc` files to import signals and decoder signals. To get an example `.dbc` file, do the following.

To get a `.dbc` file

1. Download the [EngineSignals.zip](#).
2. Navigate to the directory where you downloaded the `EngineSignals.zip` file.
3. Unzip the file and save it locally as `EngineSignals.dbc`.

Topics

- [Create and manage signal catalogs](#)
- [Create and manage vehicle models](#)
- [Create and manage decoder manifests](#)

Create and manage signal catalogs

Note

You can download a [demo script](#) to convert ROS 2 messages to VSS JSON files that are compatible with the signal catalog. For more information, see the [Vision System Data Developer Guide](#).

A signal catalog is a collection of standardized signals that can be reused to create vehicle models. AWS IoT FleetWise supports [Vehicle Signal Specification \(VSS\)](#) that you can follow to define signals. A signal can be any of the following type.

Attribute

Attributes represent static information that generally doesn't change, such as manufacturer and manufacturing date.

Branch

Branches represent signals in a nested structure. Branches demonstrate signal hierarchies. For example, the Vehicle branch has a child branch, Powertrain. The Powertrain branch has a child branch, combustionEngine. To locate the combustionEngine branch, use the `Vehicle.Powertrain.combustionEngine` expression.

Sensor

Sensor data reports the current state of the vehicle and change over time, as the state of the vehicle changes, such as fluid levels, temperatures, vibrations, or voltage.

Actuator

Actuator data reports the state of a vehicle device, such as motors, heaters, and door locks. Changing the state of a vehicle device can update actuator data. For example, you can define an actuator to represent the heater. The actuator receives new data when you turn on or off the heater.

Custom structure

A custom structure (also known as a struct) represents a complex or higher-order data structure. It facilitates logical binding or grouping of data that originates from the same source. A struct is

used when data is read or written in an atomic operation, such as to represent a complex data type or higher-order shape.

A signal of struct type is defined in the signal catalog using a reference to a struct data type instead of a primitive data type. Structs can be used for all types of signals including sensors, attributes, actuators, and vision system data types. If a signal of struct type is sent or received, AWS IoT FleetWise expects all included items to have valid values, so all items are mandatory. For example, if a struct contains the items `Vehicle.Camera.Image.height`, `Vehicle.Camera.Image.width`, and `Vehicle.Camera.Image.data` – it's expected that the sent signal contains values for all of these items.

Note

Vision system data is in preview release and is subject to change.

Custom property

A custom property represents a member of the complex data structure. The data type of the property can be either primitive or another struct.

When representing a higher-order shape using a struct and custom property, the intended higher-order shape is always defined and visioned as a tree structure. The custom property is used to define all the leaf nodes while the struct is used to define all the non-leaf nodes.

Note

- If you use the AWS IoT FleetWise console to create the first vehicle model, you don't need to manually create a signal catalog. When you create your first vehicle model, AWS IoT FleetWise automatically creates a signal catalog for you. For more information, see [Create a vehicle model \(console\)](#).
- If you use the AWS IoT FleetWise console to create a vehicle model, you can upload `.dbc` files to import signals. `.dbc` is a file format that Controller Area Network (CAN bus) databases support. After the vehicle model is created, new signals are automatically added to the signal catalog. For more information, see [Create a vehicle model \(console\)](#).
- AWS IoT FleetWise currently supports a signal catalog for each AWS account per Region.

AWS IoT FleetWise provides the following API operations that you can use to create and manage signal catalogs.

- [CreateSignalCatalog](#) – Creates a new signal catalog.
- [ImportSignalCatalog](#) – Imports signals to create a signal catalog by uploading a JSON file. Signals must be defined by following VSS and saved in the JSON format.
- [UpdateSignalCatalog](#) – Updates an existing signal catalog by updating, removing, or adding signals.
- [DeleteSignalCatalog](#) – Deletes an existing signal catalog.
- [ListSignalCatalogs](#) – Retrieves a paginated list of summaries of all signal catalogs.
- [ListSignalCatalogNodes](#) – Retrieves a paginated list of summaries of all signals (nodes) in a given signal catalog.
- [GetSignalCatalog](#) – Retrieves information about a signal catalog.

Tutorials

- [Configure signals](#)
- [Create a signal catalog \(AWS CLI\)](#)
- [Import a signal catalog](#)
- [Update a signal catalog \(AWS CLI\)](#)
- [Delete a signal catalog \(AWS CLI\)](#)
- [Get signal catalog information \(AWS CLI\)](#)

Configure signals

This section shows you how to configure branches, attributes, sensors, and actuators.

Topics

- [Configure branches](#)
- [Configure attributes](#)
- [Configure sensors or actuators](#)
- [Configure complex data types](#)

Configure branches

To configure a branch, specify the following information.

- `fullyQualifiedName` – The fully qualified name of the branch is the path to the branch plus the branch's name. Use a dot(.) to refer to a child branch. For example, `Vehicle.Chassis.SteeringWheel` is the fully qualified name for the `SteeringWheel` branch. `Vehicle.Chassis.` is the path to this branch.

The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, colon (:), and underscore (_).

- (Optional) `Description` – The description for the branch.

The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) `deprecationMessage` – The deprecation message for the node or branch being moved or deleted.

The `deprecationMessage` can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) `comment` – A comment in addition to the description. A comment can be used to provide additional information about the branch, such as the rationale for the branch or references to related branches.

The comment can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

Configure attributes

To configure an attribute, specify the following information.

- `dataType` – The attribute's data type must be one of the following: `INT8`, `UINT8`, `INT16`, `UINT16`, `INT32`, `UINT32`, `INT64`, `UINT64`, `BOOLEAN`, `FLOAT`, `DOUBLE`, `STRING`, `UNIX_TIMESTAMP`, `INT8_ARRAY`, `UINT8_ARRAY`, `INT16_ARRAY`, `UINT16_ARRAY`, `INT32_ARRAY`, `UINT32_ARRAY`, `INT64_ARRAY`, `UINT64_ARRAY`, `BOOLEAN_ARRAY`, `FLOAT_ARRAY`, `DOUBLE_ARRAY`, `STRING_ARRAY`, `UNIX_TIMESTAMP_ARRAY`, `UNKNOWN`, `fullyQualifiedName`, or a custom struct defined in the data type branch.

- **fullyQualifiedName** – The fully qualified name of the attribute is the path to the attribute plus the attribute's name. Use a dot(.) to refer to a child signal. For example, `Vehicle.Chassis.SteeringWheel.Diameter` is the fully qualified name for the `Diameter` attribute. `Vehicle.Chassis.SteeringWheel.` is the path to this attribute.

The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).

- (Optional) **Description** – The description for the attribute.

The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) **unit** – The scientific unit for the attribute, such as km or Celsius.
- (Optional) **min** – The minimum value of the attribute.
- (Optional) **max** – The maximum value of the attribute.
- (Optional) **defaultValue** – The default value of the attribute.
- (Optional) **assignedValue** – The value assigned to the attribute.
- (Optional) **allowedValues** – A list of values that the attribute accepts.
- (Optional) **deprecationMessage** – The deprecation message for the node or branch that's being moved or deleted.

The `deprecationMessage` can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) **comment** – A comment in addition to the description. A comment can be used to provide additional information about the attribute, such as the rationale for the attribute or references to related attributes.

The comment can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

Configure sensors or actuators

To configure a sensor or actuator, specify the following information.

- **dataType** – The signal's data type must be one of the following: `INT8`, `UINT8`, `INT16`, `UINT16`, `INT32`, `UINT32`, `INT64`, `UINT64`, `BOOLEAN`, `FLOAT`, `DOUBLE`, `STRING`, `UNIX_TIMESTAMP`, `INT8_ARRAY`, `UINT8_ARRAY`, `INT16_ARRAY`, `UINT16_ARRAY`, `INT32_ARRAY`, `UINT32_ARRAY`,

INT64_ARRAY, UINT64_ARRAY, BOOLEAN_ARRAY, FLOAT_ARRAY, DOUBLE_ARRAY, STRING_ARRAY, UNIX_TIMESTAMP_ARRAY, UNKNOWN, fullyQualifiedName, or a custom struct defined in the data type branch.

- `fullyQualifiedName` – The fully qualified name of the signal is the path to the signal plus the signal's name. Use a dot(.) to refer to a child signal. For example, `Vehicle.Chassis.SteeringWheel.HandsOff.HandsOffSteeringState` is the fully qualified name for the `HandsOffSteeringState` actuator. `Vehicle.Chassis.SteeringWheel.HandsOff.` is the path to this actuator.

The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).

- (Optional) `Description` – The description for the signal.

The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) `unit` – The scientific unit for the signal, such as km or celsius.
- (Optional) `min` – The minimum value of the signal.
- (Optional) `max` – The maximum value of the signal.
- (Optional) `assignedValue` – The value assigned to the signal.
- (Optional) `allowedValues` – list of values that the signal accepts.
- (Optional) `deprecationMessage` – The deprecation message for the node or branch that's being moved or deleted.

The `deprecationMessage` can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) `comment` – A comment in addition to the description. A comment can be used to provide additional information about the sensor or actuator, such as their rationale or references to related sensors or actuators.

The comment can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

Configure complex data types

Complex data types are used when modeling vision systems. In addition to branches, these data types are made up of structures (also known as a struct) and properties. A struct is a signal that

is described by multiple values, like an image. A property represents a member of the struct, like a primitive data type (such as UINT8) or another struct (such as timestamp). For example, `Vehicle.Cameras.Front` represents a branch, `Vehicle.Cameras.Front.Image` represents a struct, and `Vehicle.Cameras.Timestamp` represents a property.

The following complex data type example demonstrates how signals and data types are exported to a single JSON file.

Example complex data type

```
{
  "Vehicle": {
    "type": "branch"
    // Signal tree
  },
  "ComplexDataTypes": {
    "VehicleDataTypes": {
      // complex data type tree
      "children": {
        "branch": {
          "children": {
            "Struct": {
              "children": {
                "Property": {
                  "type": "property",
                  "datatype": "Data type",
                  "description": "Description",
                  //          ...
                }
              },
              "description": "Description",
              "type": "struct"
            }
          },
          "description": "Description",
          "type": "branch"
        }
      }
    }
  }
}
```

Note

You can download a [demo script](#) to convert ROS 2 messages to VSS JSON files that are compatible with the signal catalog. For more information, see the [Vision System Data Developer Guide](#).

Vision system data is in preview release and is subject to change.

Configure struct

To configure a custom structure (or struct), specify the following information.

- `fullyQualifiedName` – The fully qualified name of the custom structure. For example, the fully qualified name of a custom structure might be `ComplexDataTypes.VehicleDataTypes.SVMCamera`.

The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).

- (Optional) `Description` – The description for the signal.

The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) `deprecationMessage` – The deprecation message for the node or branch that's being moved or deleted.

The `deprecationMessage` can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) `comment` – A comment in addition to the description. A comment can be used to provide additional information about the sensor or actuator, such as their rationale or references to related sensors or actuators.

The comment can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

Configure property

To configure a custom property, specify the following information.

- `dataType` – The signal's data type must be one of the following: INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, BOOLEAN, FLOAT, DOUBLE, STRING, UNIX_TIMESTAMP, INT8_ARRAY, UINT8_ARRAY, INT16_ARRAY, UINT16_ARRAY, INT32_ARRAY, UINT32_ARRAY, INT64_ARRAY, UINT64_ARRAY, BOOLEAN_ARRAY, FLOAT_ARRAY, DOUBLE_ARRAY, STRING_ARRAY, UNIX_TIMESTAMP_ARRAY, STRUCT, STRUCT_ARRAY, or UNKNOWN.
- `fullyQualifiedName` – The fully qualified name of the custom property.
For example, the fully qualified name of a custom property might be `ComplexDataTypes.VehicleDataTypes.SVMCamera.FPS`.

The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore)

- (Optional) `Description` – The description for the signal.

The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) `deprecationMessage` – The deprecation message for the node or branch that's being moved or deleted.

The `deprecationMessage` can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) `comment` – A comment in addition to the description. A comment can be used to provide additional information about the sensor or actuator, such as their rationale or references to related sensors or actuators.

The `comment` can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

- (Optional) `dataEncoding` – Indicates whether the property is binary data. The custom property's data encoding must be one of the following: BINARY or TYPED.
- (Optional) `structFullyQualifiedName` – The fully qualified name of the structure (struct) node for the custom property if the data type of the custom property is Struct or StructArray.

The fully qualified name can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).

Create a signal catalog (AWS CLI)

You can use the [CreateSignalCatalog](#) API operation to create a signal catalog. The following example uses AWS CLI.

To create a signal catalog, run the following command.

Replace *signal-catalog-configuration* with the name of the JSON file that contains the configuration.

```
aws iotfleetwise create-signal-catalog --cli-input-json file://signal-catalog-configuration.json
```

- Replace *signal-catalog-name* with the name of the signal catalog that you're creating.
- (Optional) Replace *description* with a description to help you identify the signal catalog.

For more information about how to configure branches, attributes, sensors, and actuators, see [Configure signals](#).

```
{
  "name": "signal-catalog-name",
  "description": "description",
  "nodes": [
    {
      "branch": {
        "fullyQualifiedName": "Types"
      }
    },
    {
      "struct": {
        "fullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage"
      }
    },
    {
      "struct": {
        "fullyQualifiedName": "Types.std_msgs_Header"
      }
    },
    {
      "struct": {
        "fullyQualifiedName": "Types.builtin_interfaces_Time"
      }
    }
  ]
}
```

```

    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.builtin_interfaces_Time.sec",
      "dataType": "INT32",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.builtin_interfaces_Time.nanosec",
      "dataType": "UINT32",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.std_msgs_Header.stamp",
      "dataType": "STRUCT",
      "structFullyQualifiedName": "Types.builtin_interfaces_Time"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.std_msgs_Header.frame_id",
      "dataType": "STRING",
      "dataEncoding": "TYPED"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage.header",
      "dataType": "STRUCT",
      "structFullyQualifiedName": "Types.std_msgs_Header"
    }
  },
  {
    "property": {
      "fullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage.format",
      "dataType": "STRING",
      "dataEncoding": "TYPED"
    }
  },
},

```

```
{
  "property": {
    "fullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage.data",
    "dataType": "UINT8_ARRAY",
    "dataEncoding": "BINARY"
  }
},
{
  "branch": {
    "fullyQualifiedName": "Vehicle",
    "description": "Vehicle"
  }
},
{
  "branch": {
    "fullyQualifiedName": "Vehicle.Cameras"
  }
},
{
  "branch": {
    "fullyQualifiedName": "Vehicle.Cameras.Front"
  }
},
{
  "sensor": {
    "fullyQualifiedName": "Vehicle.Cameras.Front.Image",
    "dataType": "STRUCT",
    "structFullyQualifiedName": "Types.sensor_msgs_msg_CompressedImage"
  }
},
{
  "struct": {
    "fullyQualifiedName": "Types.std_msgs_msg_Float64"
  }
},
{
  "property": {
    "fullyQualifiedName": "Types.std_msgs_msg_Float64.data",
    "dataType": "DOUBLE",
    "dataEncoding": "TYPED"
  }
},
{
  "sensor": {
```



```
    "fullyQualifiedName": "Vehicle.Velocity",
    "dataType": "STRUCT",
    "structFullyQualifiedName": "Types.std_msgs_msg_Float64"
  }
},
{
  "struct": {
    "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest"
  }
},
{
  "property": {
    "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.x_offset",
    "dataType": "UINT32",
    "dataEncoding": "TYPED"
  }
},
{
  "property": {
    "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.y_offset",
    "dataType": "UINT32",
    "dataEncoding": "TYPED"
  }
},
{
  "property": {
    "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.height",
    "dataType": "UINT32",
    "dataEncoding": "TYPED"
  }
},
{
  "property": {
    "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.width",
    "dataType": "UINT32",
    "dataEncoding": "TYPED"
  }
},
{
  "property": {
    "fullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest.do_rectify",
    "dataType": "BOOLEAN",
    "dataEncoding": "TYPED"
  }
}
```

```
},
{
  "branch": {
    "fullyQualifiedName": "Vehicle.Perception"
  }
},
{
  "sensor": {
    "fullyQualifiedName": "Vehicle.Perception.Obstacle",
    "dataType": "STRUCT",
    "structFullyQualifiedName": "Types.sensor_msgs_msg_RegionOfInterest"
  }
}
]
}
```

Note

You can download a [demo script](#) to convert ROS 2 messages to VSS JSON files that are compatible with the signal catalog. For more information, see the [Vision System Data Developer Guide](#).

Vision system data is in preview release and is subject to change.

Import a signal catalog

You can use the AWS IoT FleetWise console or API to import a signal catalog.

Topics

- [Import a signal catalog \(console\)](#)
- [Import a signal catalog \(AWS CLI\)](#)

Import a signal catalog (console)

You can use the AWS IoT FleetWise console to import a signal catalog.

Important

You can have a maximum of one signal catalog. If you already have a signal catalog, you won't see the option to import a signal catalog in the console.

To import a signal catalog

1. Open the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Signal catalog**.
3. On the signal catalog summary page, choose **Import signal catalog**.
4. Import the file containing the signals.
 - To upload a file from an S3 bucket:
 - a. Choose **Import from S3**.
 - b. Choose **Browse S3**.
 - c. For **Buckets**, enter the bucket name or object, choose it from the list, and then choose the file from the list. Choose the **Choose file** button.

Or, for **S3 URI**, enter an Amazon Simple Storage Service URI. For more information, see [Methods for accessing a bucket](#) in the *Amazon S3 User Guide*.
 - To upload a file from your computer:
 - a. Choose **Import from file**.
 - b. Upload a .json file in a [Vehicle Signal Specification \(VSS\)](#) format.
5. Verify the signal catalog, and then choose **Import file**.

Import a signal catalog (AWS CLI)

You can use the [ImportSignalCatalog](#) API operation to upload a JSON file that helps create a signal catalog. You must follow the [Vehicle Signal Specification \(VSS\)](#) to save signals in the JSON file. The following example uses AWS CLI.

To import a signal catalog, run the following command.

- Replace *signal-catalog-name* with the name of the signal catalog that you're creating.

- (Optional) Replace description with a *description* to help you identify the signal catalog.
- Replace *signal-catalog-configuration-vss* with the name of the JSON string file that contains signals defined in VSS.

For more information about how to configure branches, attributes, sensors, and actuators, see [Configure signals](#).

```
aws iotfleetwise import-signal-catalog \  
    --name signal-catalog-name \  
    --description description \  
    --vss file://signal-catalog-configuration-vss.json
```

The JSON must be stringified and passed through the `vssJson` field. The following is an example of signals defined in VSS.

```
{  
  "Vehicle": {  
    "type": "branch",  
    "children": {  
      "Chassis": {  
        "type": "branch",  
        "description": "All data concerning steering, suspension, wheels, and brakes.",  
        "children": {  
          "SteeringWheel": {  
            "type": "branch",  
            "description": "Steering wheel signals",  
            "children": {  
              "Diameter": {  
                "type": "attribute",  
                "description": "The diameter of the steering wheel",  
                "datatype": "float",  
                "unit": "cm",  
                "min": 1,  
                "max": 50  
              },  
              "HandsOff": {  
                "type": "branch",  
                "children": {  
                  "HandsOffSteeringState": {  
                    "type": "actuator",  
                    "description": "HndsOffStrWhlDtSt. Hands Off Steering State",
```

```

        "datatype": "boolean"
    },
    "HandsOffSteeringMode": {
        "type": "actuator",
        "description": "HndsOffStrWhlDtMd. Hands Off Steering Mode",
        "datatype": "int8",
        "min": 0,
        "max": 2
    }
}
}
},
"Accelerator": {
    "type": "branch",
    "description": "",
    "children": {
        "AcceleratorPedalPosition": {
            "type": "sensor",
            "description": "Throttle__Position. Accelerator pedal position as percent. 0 =
Not depressed. 100 = Fully depressed.",
            "datatype": "uint8",
            "unit": "%",
            "min": 0,
            "max": 100.000035
        }
    }
}
},
"Powertrain": {
    "type": "branch",
    "description": "Powertrain data for battery management, etc.",
    "children": {
        "Transmission": {
            "type": "branch",
            "description": "Transmission-specific data, stopping at the drive shafts.",
            "children": {
                "VehicleOdometer": {
                    "type": "sensor",
                    "description": "Vehicle_Odometer",
                    "datatype": "float",
                    "unit": "km",
                    "min": 0,

```

```
    "max": 67108863.984375
  }
}
},
"CombustionEngine": {
  "type": "branch",
  "description": "Engine-specific data, stopping at the bell housing.",
  "children": {
    "Engine": {
      "type": "branch",
      "description": "Engine description",
      "children": {
        "timing": {
          "type": "branch",
          "description": "timing description",
          "children": {
            "run_time": {
              "type": "sensor",
              "description": "Engine run time",
              "datatype": "int16",
              "unit": "ms",
              "min": 0,
              "max": 10000
            },
            "idle_time": {
              "type": "sensor",
              "description": "Engine idle time",
              "datatype": "int16",
              "min": 0,
              "unit": "ms",
              "max": 10000
            }
          }
        }
      }
    }
  }
}
},
"Axle": {
  "type": "branch",
  "description": "Axle signals",
  "children": {
```

```
"TireRRPrs": {
  "type": "sensor",
  "description": "TireRRPrs. Right rear Tire pressure in kilo-Pascal",
  "datatype": "float",
  "unit": "kPaG",
  "min": 0,
  "max": 1020
}
}
},
"Cameras": {
  "type": "branch",
  "description": "Branch to aggregate all cameras in the vehicle",
  "children": {
    "FrontViewCamera": {
      "type": "sensor",
      "datatype": "VehicleDataTypes.SVMCamera",
      "description": "Front view camera"
    },
    "RearViewCamera": {
      "type": "sensor",
      "datatype": "VehicleDataTypes.SVMCamera",
      "description": "Rear view camera"
    },
    "LeftSideViewCamera": {
      "type": "sensor",
      "datatype": "VehicleDataTypes.SVMCamera",
      "description": "Left side view camera"
    },
    "RightSideViewCamera": {
      "type": "sensor",
      "datatype": "VehicleDataTypes.SVMCamera",
      "description": "Right side view camera"
    }
  }
},
"ComplexDataTypes": {
  "VehicleDataTypes": {
    "type": "branch",
    "description": "Branch to aggregate all camera related higher order data types",
    "children": {
      "SVMCamera": {
```

```
"type": "struct",
"description": "This data type represents Surround View Monitor (SVM) camera
system in a vehicle",
"comment": "Test comment",
"deprecation": "Test deprecation message",
"children": {
  "Make": {
    "type": "property",
    "description": "Make of the SVM camera",
    "datatype": "string",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  },
  "Description": {
    "type": "property",
    "description": "Description of the SVM camera",
    "datatype": "string",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  },
  "FPS": {
    "type": "property",
    "description": "FPS of the SVM camera",
    "datatype": "double",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  },
  "Orientation": {
    "type": "property",
    "description": "Orientation of the SVM camera",
    "datatype": "VehicleDataTypes.Orientation",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  },
  "Range": {
    "type": "property",
    "description": "Range of the SVM camera",
    "datatype": "VehicleDataTypes.Range",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  },
  "RawData": {
    "type": "property",
    "description": "Represents binary data of the SVM camera",
```



```

    "datatype": "uint8[]",
    "dataencoding": "binary",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  },
  "CapturedFrames": {
    "type": "property",
    "description": "Represents selected frames captured by the SVM camera",
    "datatype": "VehicleDataTypes.Frame[]",
    "dataencoding": "typed",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  }
}
},
"Range": {
  "type": "struct",
  "description": "Range of a camera in centimeters",
  "comment": "Test comment",
  "deprecation": "Test deprecation message",
  "children": {
    "Min": {
      "type": "property",
      "description": "Minimum range of a camera in centimeters",
      "datatype": "uint32",
      "comment": "Test comment",
      "deprecation": "Test deprecation message"
    },
    "Max": {
      "type": "property",
      "description": "Maximum range of a camera in centimeters",
      "datatype": "uint32",
      "comment": "Test comment",
      "deprecation": "Test deprecation message"
    }
  }
}
},
"Orientation": {
  "type": "struct",
  "description": "Orientation of a camera",
  "comment": "Test comment",
  "deprecation": "Test deprecation message",
  "children": {
    "Front": {

```

```
    "type": "property",
    "description": "Indicates whether the camera is oriented to the front of the
vehicle",
    "datatype": "boolean",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  },
  "Rear": {
    "type": "property",
    "description": "Indicates whether the camera is oriented to the rear of the
vehicle",
    "datatype": "boolean",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  },
  "Side": {
    "type": "property",
    "description": "Indicates whether the camera is oriented to the side of the
vehicle",
    "datatype": "boolean",
    "comment": "Test comment",
    "deprecation": "Test deprecation message"
  }
}
},
"Frame": {
  "type": "struct",
  "description": "Represents a camera frame",
  "comment": "Test comment",
  "deprecation": "Test deprecation message",
  "children": {
    "Data": {
      "type": "property",
      "datatype": "string",
      "dataencoding": "binary",
      "comment": "Test comment",
      "deprecation": "Test deprecation message"
    }
  }
}
}
}
}
```

}

The following example shows the same signals defined in VSS in a JSON string.

```
{
  "vssJson": "{\\"Vehicle\\":{\\"type\\":\\"branch\\",\\"children\\":{\\"Chassis\\":{\\"type\\":\\"branch\\",\\"description\\":\\"All data concerning steering, suspension, wheels, and brakes.\\",\\"children\\":{\\"SteeringWheel\\":{\\"type\\":\\"branch\\",\\"description\\":\\"Steering wheel signals\\",\\"children\\":{\\"Diameter\\":{\\"type\\":\\"attribute\\",\\"description\\":\\"The diameter of the steering wheel\\",\\"datatype\\":\\"float\\",\\"unit\\":\\"cm\\",\\"min\\":1,\\"max\\":50},\\"HandsOff\\":{\\"type\\":\\"branch\\",\\"children\\":{\\"HandsOffSteeringState\\":{\\"type\\":\\"actuator\\",\\"description\\":\\"HndsOffStrWhlDtSt. Hands Off Steering State\\",\\"datatype\\":\\"boolean\\"},\\"HandsOffSteeringMode\\":{\\"type\\":\\"actuator\\",\\"description\\":\\"HndsOffStrWhlDtMd. Hands Off Steering Mode \\",\\"datatype\\":\\"int8\\",\\"min\\":0,\\"max\\":2}}}},\\"Accelerator\\":{\\"type\\":\\"branch\\",\\"description\\":\\"\\",\\"children\\":{\\"AcceleratorPedalPosition\\":{\\"type\\":\\"sensor\\",\\"description\\":\\"Throttle__Position. Accelerator pedal position as percent. 0 = Not depressed. 100 = Fully depressed.\\",\\"datatype\\":\\"uint8\\",\\"unit\\":\\"%\\",\\"min\\":0,\\"max\\":100.000035}}}},\\"Powertrain\\":{\\"type\\":\\"branch\\",\\"description\\":\\"Powertrain data for battery management, etc.\\",\\"children\\":{\\"Transmission\\":{\\"type\\":\\"branch\\",\\"description\\":\\"Transmission-specific data, stopping at the drive shafts.\\",\\"children\\":{\\"VehicleOdometer\\":{\\"type\\":\\"sensor\\",\\"description\\":\\"Vehicle_Odometer\\",\\"datatype\\":\\"float\\",\\"unit\\":\\"km\\",\\"min\\":0,\\"max\\":67108863.984375}}},\\"CombustionEngine\\":{\\"type\\":\\"branch\\",\\"description\\":\\"Engine-specific data, stopping at the bell housing.\\",\\"children\\":{\\"Engine\\":{\\"type\\":\\"branch\\",\\"description\\":\\"Engine description\\",\\"children\\":{\\"timing\\":{\\"type\\":\\"branch\\",\\"description\\":\\"timing description\\",\\"children\\":{\\"run_time\\":{\\"type\\":\\"sensor\\",\\"description\\":\\"Engine run time\\",\\"datatype\\":\\"int16\\",\\"unit\\":\\"ms\\",\\"min\\":0,\\"max\\":10000},\\"idle_time\\":{\\"type\\":\\"sensor\\",\\"description\\":\\"Engine idle time\\",\\"datatype\\":\\"int16\\",\\"min\\":0,\\"unit\\":\\"ms\\",\\"max\\":10000}}}}}}}},\\"Axle\\":{\\"type\\":\\"branch\\",\\"description\\":\\"Axle signals\\",\\"children\\":{\\"TireRRPrs\\":{\\"type\\":\\"sensor\\",\\"description\\":\\"TireRRPrs. Right rear Tire pressure in kilo-Pascal\\",\\"datatype\\":\\"float\\",\\"unit\\":\\"kPaG\\",\\"min\\":0,\\"max\\":1020}}}}}}}"
}
```

Note

You can download a [demo script](#) to convert ROS 2 messages to VSS JSON files that are compatible with the signal catalog. For more information, see the [Vision System Data Developer Guide](#).

Vision system data is in preview release and is subject to change.

Update a signal catalog (AWS CLI)

You can use the [UpdateSignalCatalog](#) API operation to update an existing signal catalog. The following example uses AWS CLI.

To update an existing signal catalog, run the following command.

Replace *signal-catalog-configuration* with the name of the JSON file that contains the configuration.

```
aws iotfleetwise update-signal-catalog --cli-input-json file://signal-catalog-configuration.json
```

Replace *signal-catalog-name* with the name of the signal catalog that you're updating.

For more information about how to configure branches, attributes, sensors, and actuators, see [Configure signals](#).

Important

Custom structures are immutable. If you need to re-order or insert properties to an existing custom structure (struct), delete the structure and create a brand new structure with the desired order of properties.

To delete a custom structure, add the structure's fully qualified name in `nodesToRemove`. A structure can't be deleted if it's referred to by any signals. Any signals that refer to the structure (their data type is defined as the target structure) must be updated or deleted before the request to update the signal catalog.

```
{
  "name": "signal-catalog-name",
  "nodesToAdd": [{
    "branch": {
      "description": "Front left of vehicle specific data.",
      "fullyQualifiedName": "Vehicle.Front.Left"
    }
  ]
},
```

```
{
  "branch": {
    "description": "Door-specific data for the front left of vehicle.",
    "fullyQualifiedName": "Vehicle.Front.Left.Door"
  }
},
{
  "actuator": {
    "fullyQualifiedName": "Vehicle.Front.Left.Door.Lock",
    "description": "Whether the front left door is locked.",
    "dataType": "BOOLEAN"
  }
},
{
  "branch": {
    "fullyQualifiedName": "Vehicle.Camera"
  }
},
{
  "struct": {
    "fullyQualifiedName": "Vehicle.Camera.SVMCamera"
  }
},
{
  "property": {
    "fullyQualifiedName": "Vehicle.Camera.SVMCamera.ISO",
    "dataType": "STRING"
  }
}
],
"nodesToRemove": ["Vehicle.Chassis.SteeringWheel.HandsOffSteeringState"],
"nodesToUpdate": [{
  "attribute": {
    "dataType": "FLOAT",
    "fullyQualifiedName": "Vehicle.Chassis.SteeringWheel.Diameter",
    "max": 55
  }
}]
}
```

Delete a signal catalog (AWS CLI)

You can use the [DeleteSignalCatalog](#) API operation to delete a signal catalog. The following example uses AWS CLI.

Important

Before deleting a signal catalog, make sure it has no associated vehicle models, decoder manifests, vehicles, fleets, or campaigns. For instructions, see the following:

- [Delete a vehicle model](#)
- [Delete a decoder manifest](#)
- [Delete a vehicle](#)
- [Delete a fleet \(AWS CLI\)](#)
- [Delete a campaign](#)

To delete an existing signal catalog, run the following command. Replace *signal-catalog-name* with the name of the signal catalog that you're deleting.

```
aws iotfleetwise delete-signal-catalog --name signal-catalog-name
```

Note

This command doesn't produce output.

Get signal catalog information (AWS CLI)

You can use the [ListSignalCatalogs](#) API operation to verify if a signal catalog has been deleted. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all signal catalogs, run the following command.

```
aws iotfleetwise list-signal-catalogs
```

You can use the [ListSignalCatalogNodes](#) API operation to verify if a signal catalog has been updated. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all signals (nodes) in a given signal catalog, run the following command.

Replace *signal-catalog-name* with the name of the signal catalog that you're checking.

```
aws iotfleetwise list-signal-catalog-nodes --name signal-catalog-name
```

You can use the [GetSignalCatalog](#) API operation to retrieve signal catalog information. The following example uses AWS CLI.

To retrieve information about a signal catalog, run the following command.

Replace *signal-catalog-name* with the name of the signal catalog that you want to retrieve.

```
aws iotfleetwise get-signal-catalog --name signal-catalog-name
```

Note

This operation is [eventually consistent](#). In other words, changes to the signal catalog might not be reflected immediately.

Create and manage vehicle models

You use signals to create vehicle models that help standardize the format of your vehicles. Vehicle models enforce consistent information across multiple vehicles of the same type, so that you can process data from fleets of vehicles. Vehicles created from the same vehicle model inherit the same group of signals. For more information, see [Create, provision, and manage vehicles](#).

Each vehicle model has a status field that contains the state of the vehicle model. The state can be one of the following values:

- ACTIVE – The vehicle model is active.
- DRAFT – The configuration of the vehicle model is saved.

⚠ Important

- If you want to use the `CreateModelManifest` API operation to create the first vehicle model, you must create a signal catalog first. For more information, see [Create a signal catalog \(AWS CLI\)](#).
- If you use the AWS IoT FleetWise console to create a vehicle model, AWS IoT FleetWise automatically activates the vehicle model for you.
- If you use the `CreateModelManifest` API operation to create a vehicle model, the vehicle model stays in the DRAFT state.
- You can't create vehicles from vehicle models that are in the DRAFT state. Use the `UpdateModelManifest` API operation to change vehicle models to the ACTIVE state.
- You can't edit vehicle models that are in the ACTIVE state.

Topics

- [Create a vehicle model](#)
- [Update a vehicle model \(AWS CLI\)](#)
- [Delete a vehicle model](#)
- [Get vehicle model information \(AWS CLI\)](#)

Create a vehicle model

You can use the AWS IoT FleetWise console or API to create vehicle models.

⚠ Important

You must have a signal catalog before you can create a vehicle model by using the `CreateModelManifest` API operation.

Topics

- [Create a vehicle model \(console\)](#)
- [Create a vehicle model \(AWS CLI\)](#)

Create a vehicle model (console)

In the AWS IoT FleetWise console, you can create a vehicle model in the following ways:

- [Use a template provided by AWS](#)
- [Manually create a vehicle model](#)
- [Duplicate a vehicle model](#)

Use a template provided by AWS

AWS IoT FleetWise provides an On-board Diagnostic (OBD) II, J1979 template that automatically creates a signal catalog, a vehicle model, and a decoder manifest for you. The template also adds OBD network interfaces to the decoder manifest. For more information, see [Create and manage decoder manifests](#).

To create a vehicle model by using a template

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Vehicle models**.
3. On the **Vehicle models** page, choose **Add provided template**.
4. Choose **On-board diagnostics (OBD) II**.
5. Enter a name for the OBD network interface that AWS IoT FleetWise is creating.
6. Choose **Add**.

Manually create a vehicle model

You can add signals from the signal catalog or import signals by uploading one or more .dbc files. A .dbc file is a file format that Controller Area Network (CAN bus) databases support.

Important

You can't create a vehicle model with vision system data signals using the AWS IoT FleetWise console. Instead, use the AWS CLI to create a vehicle model. Vision system data is in preview release and is subject to change.

To manually create a vehicle model

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Vehicle models**.
3. On the **Vehicle models** page, choose **Create vehicle model**, and then do the following.

Topics

- [Step 1: Configure vehicle model](#)
- [Step 2: Add signals](#)
- [Step 3: Import signals](#)
- [\(Optional\) Step 4: Add attributes](#)
- [Step 5: Review and create](#)

Step 1: Configure vehicle model

In **General information**, do the following.

1. Enter a name for the vehicle model.
2. (Optional) Enter a description.
3. Choose **Next**.

Step 2: Add signals

Note

- If this is the first time you've used AWS IoT FleetWise, this step isn't available until you have a signal catalog. When the first vehicle model is created, AWS IoT FleetWise automatically creates a signal catalog with signals added to the first vehicle model.
- If you're experienced with AWS IoT FleetWise, you can add signals to your vehicle model by selecting signals from the signal catalog or uploading .dbc files to import signals.
- You must have at least one signal to create a vehicle model.

To add signals

1. Choose one or more signals from the signal catalog that you're adding to the vehicle model. You can review selected signals in the right pane.

Note

Only selected signals will be added to the vehicle model.

2. Choose **Next**.

Step 3: Import signals

Note

- If this is the first time you've used AWS IoT FleetWise, you must upload at least one .dbc file to import signals.
- If you're experienced with AWS IoT FleetWise, you can add signals to your vehicle model by selecting signals from the signal catalog or uploading .dbc files to import signals.
- You must have at least one signal to create a vehicle model.

To import signals

1. Choose **Choose files**.
2. In the dialog box, choose the .dbc file that contains signals. You can upload multiple .dbc files.
3. AWS IoT FleetWise parses your .dbc files to retrieve signals.

In the **Signals** section, specify the following metadata for each signal.

- **Name** – The signal's name.

The signal name must be unique. The signal name plus the path can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).

- **Data type** – The signal's data type must be one of the following: INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, BOOLEAN, FLOAT, DOUBLE, STRING, UNIX_TIMESTAMP, INT8_ARRAY, UINT8_ARRAY, INT16_ARRAY, UINT16_ARRAY,

INT32_ARRAY, UINT32_ARRAY, INT64_ARRAY, UINT64_ARRAY, BOOLEAN_ARRAY, FLOAT_ARRAY, DOUBLE_ARRAY, STRING_ARRAY, UNIX_TIMESTAMP_ARRAY, or UNKNOWN.

- **Signal type** – The type of the signal, which can be **Sensor** or **Actuator**.
- (Optional) **Unit** – The scientific unit for the signal, such as km or Celsius.
- (Optional) **Path** – The path to the signal. Similar to JSONPath, use a dot(.) to refer to a child signal. For example, **Vehicle.Engine.Light**.

The signal name plus the path can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore).

- (Optional) **Min** – The minimum value of the signal.
- (Optional) **Max** – The maximum value of the signal.
- (Optional) **Description** – The description for the signal.

The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

4. Choose **Next**.

(Optional) Step 4: Add attributes

You can add up to 100 attributes, including the existing attributes in the signal catalog.

To add attributes

1. In **Add attributes**, specify the following metadata for each attribute.

- **Name** – The attribute's name.

The signal name must be unique. The signal name and path can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore)

- **Data type** – The attribute's data type must be one of the following: INT8, UINT8, INT16, UINT16, INT32, UINT32, INT64, UINT64, BOOLEAN, FLOAT, DOUBLE, STRING, UNIX_TIMESTAMP, INT8_ARRAY, UINT8_ARRAY, INT16_ARRAY, UINT16_ARRAY, INT32_ARRAY, UINT32_ARRAY, INT64_ARRAY, UINT64_ARRAY, BOOLEAN_ARRAY, FLOAT_ARRAY, DOUBLE_ARRAY, STRING_ARRAY, UNIX_TIMESTAMP_ARRAY, or UNKNOWN
- (Optional) **Unit** – The scientific unit for the attribute, such as km or Celsius.
- (Optional) **Path** – The path to the signal. Similar to JSONPath, use a dot(.) to refer to a child signal. For example, **Vehicle.Engine.Light**.

The signal name plus the path can have up to 150 characters. Valid characters: a–z, A–Z, 0–9, : (colon), and _ (underscore)

- (Optional) **Min** – The minimum value of the attribute.
- (Optional) **Max** – The maximum value of the attribute.
- (Optional) **Description** – The description for the attribute.

The description can have up to 2048 characters. Valid characters: a–z, A–Z, 0–9, : (colon), _ (underscore), and - (hyphen).

2. Choose **Next**.

Step 5: Review and create

Verify the configurations for the vehicle model, and then choose **Create**.

Duplicate a vehicle model

AWS IoT FleetWise can copy the configurations of an existing vehicle model to create a new model. Signals specified in the selected vehicle model are copied to the new vehicle model.

To duplicate a vehicle model

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Vehicle models**.
3. Choose a model from the vehicle model list, and then choose **Duplicate model**.

To configure the vehicle model, follow the [Manually create a vehicle model](#) tutorial.

It can take a few minutes for AWS IoT FleetWise to process your request to create the vehicle model. After the vehicle model is successfully created, on the **Vehicle models** page, the **Status** column shows **ACTIVE**. When the vehicle model becomes active, you can't edit it.

Create a vehicle model (AWS CLI)

You can use the [CreateModelManifest](#) API operation to create vehicle models (model manifests). The following example uses the AWS CLI.

⚠ Important

If you want to use the AWS IoT FleetWise API to create the first vehicle model, you must create a signal catalog first. For more information about how to create a signal catalog, see [Create a signal catalog \(AWS CLI\)](#).

To create a vehicle model, run the following command.

Replace *vehicle-model-configuration* with the name of the JSON file that contains the configuration.

```
aws iotfleetwise create-model-manifest --cli-input-json file://vehicle-model-configuration.json
```

- Replace *vehicle-model-name* with the name of the vehicle model that you're creating.
- Replace *signal-catalog-ARN* with the Amazon Resource Name (ARN) of the signal catalog.
- (Optional) Replace *description* with a description to help you identify the vehicle model.

For more information about how to configure branches, attributes, sensors, and actuators, see [Configure signals](#).

```
{  
  "name": "vehicle-model-name",  
  "signalCatalogArn": "signal-catalog-ARN",  
  "description": "description",  
  "nodes": ["Vehicle.Chassis"]  
}
```

Update a vehicle model (AWS CLI)

You can use the [UpdateModelManifest](#) API operation to update an existing vehicle model (model manifests). The following example uses the AWS CLI.

To update an existing vehicle model, run the following command.

Replace *update-vehicle-model-configuration* with the name of the JSON file that contains the configuration.

```
aws iotfleetwise update-model-manifest --cli-input-json file://update-vehicle-model-configuration.json
```

- Replace *vehicle-model-name* with the name of the vehicle model that you're updating.
- (Optional) To activate the vehicle model, replace *vehicle-model-status* with ACTIVE.

Important

After the vehicle model is activated, you can't change the vehicle model.

- (Optional) Replace *description* with an updated description to help you identify the vehicle model.

```
{  
  "name": "vehicle-model-name",  
  "status": "vehicle-model-status",  
  "description": "description",  
  "nodesToAdd": ["Vehicle.Front.Left"],  
  "nodesToRemove": ["Vehicle.Chassis.SteeringWheel"],  
}
```

Delete a vehicle model

You can use the AWS IoT FleetWise console or API to delete vehicle models.

Important

Vehicles and decoder manifests associated with the vehicle model must be deleted first. For more information, see [Delete a vehicle](#) and [Delete a decoder manifest](#).

Delete a vehicle model (console)

To delete a vehicle model, use the AWS IoT FleetWise console.

To delete a vehicle model

1. Navigate to the [AWS IoT FleetWise console](#).

2. On the navigation pane, choose **Vehicle models**.
3. On the **Vehicle models** page, choose the target vehicle model.
4. Choose **Delete**.
5. In **Delete vehicle-model-name?**, enter the name of the vehicle model to delete, and then choose **Confirm**.

Delete a vehicle model (AWS CLI)

You can use the [DeleteModelManifest](#) API operation to delete an existing vehicle model (model manifests). The following example uses the AWS CLI.

To delete a vehicle model, run the following command.

Replace *model-manifest-name* with the name of the vehicle model that you're deleting.

```
aws iotfleetwise delete-model-manifest --name model-manifest-name
```

Note

This command doesn't produce output.

Get vehicle model information (AWS CLI)

You can use the [ListModelManifests](#) API operation to verify if a vehicle model has been deleted. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all vehicle models, run the following command.

```
aws iotfleetwise list-model-manifests
```

You can use the [ListModelManifestNodes](#) API operation to verify if a vehicle model has been updated. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all signals (nodes) in a given vehicle model, run the following command.

Replace *vehicle-model-name* with the name of the vehicle model that you're checking.

```
aws iotfleetwise list-model-manifest-nodes /  
    --name vehicle-model-name
```

To retrieve information about a vehicle model, run the following command.

Replace *vehicle-model* with the name of the vehicle model that you want to retrieve.

```
aws iotfleetwise get-model-manifest --name vehicle-model
```

Note

This operation is [eventually consistent](#). In other words, changes to the vehicle model might not be reflected immediately.

Create and manage decoder manifests

Decoder manifests contain decoding information that AWS IoT FleetWise uses to transform vehicle data (binary data) into human-readable values and to prepare your data for data analyses. Network interface and decoder signals are the core components that you work with to configure decoder manifests.

Network interface

Contains information about the protocol that the in-vehicle network uses. AWS IoT FleetWise supports the following protocols.

Controller Area Network (CAN bus)

A protocol that defines how data is communicated between electronic control units (ECUs). ECUs can be the engine control unit, airbags, or the audio system.

On-board diagnostic (OBD) II

A further developed protocol that defines how self-diagnostic data is communicated between ECUs. It provides a number of standard diagnostic trouble codes (DTCs) that help identify what is wrong with your vehicle.

Vehicle middleware

The vehicle middleware defined as a type of network interface. Examples of vehicle middleware include Robot Operating System (ROS 2) and Scalable service-Oriented MiddlewarE over IP (SOME/IP).

Note

AWS IoT FleetWise supports ROS 2 middleware for vision system data.

Decoder signal

Provides detailed decoding information for a specific signal. Every signal specified in the vehicle model must be paired with a decoder signal. If the decoder manifest contains CAN network interfaces, it must contain CAN decoder signals. If the decoder manifest contains OBD network interfaces, it must contain OBD decoder signals.

The decoder manifest must contain message decoder signals if it also contains vehicle middleware interfaces.

Each decoder manifest must be associated with a vehicle model. AWS IoT FleetWise uses the associated decoder manifest to decode data from vehicles created based on the vehicle model.

Each decoder manifest has a status field that contains the state of the decoder manifest. The state can be one of the following values:

- **ACTIVE** – The decoder manifest is active.
- **DRAFT** – The configuration of the decoder manifest isn't saved.
- **VALIDATING** – The decoder manifest is under validation for its eligibility. This only applies to decoder manifests that contain at least one vision system data signal.
- **INVALID** – The decoder manifest failed validation and can't be activated yet. This only applies to decoder manifests that contain at least one vision system data signal. You can use the `ListDecoderManifests` and `GetDecoderManifest` APIs to check the reason for a failed validation.

Important

- If you use the AWS IoT FleetWise console to create a decoder manifest, AWS IoT FleetWise automatically activates the decoder manifest for you.
- If you use the `CreateDecoderManifest` API operation to create a decoder manifest, the decoder manifest stays in the DRAFT state.
- You can't create vehicles from vehicle models that are associated with a DRAFT decoder manifest. Use the `UpdateDecoderManifest` API operation to change the decoder manifest to the ACTIVE state.
- You can't edit decoder manifests that are in the ACTIVE state.

Topics

- [Configure network interfaces and decoder signals](#)
- [Create a decoder manifest](#)
- [Update a decoder manifest \(AWS CLI\)](#)
- [Delete a decoder manifest](#)
- [Get decoder manifest information \(AWS CLI\)](#)

Configure network interfaces and decoder signals

Every decoder manifest has at least a network interface and decoder signals paired with signals specified in the associated vehicle model.

If the decoder manifest contains CAN network interfaces, it must contain CAN decoder signals. If the decoder manifest contains OBD network interfaces, it must contain OBD decoder signals.

Topics

- [Configure network interfaces](#)
- [Configure decoder signals](#)

Configure network interfaces

To configure a CAN network interface, specify the following information.

- `name` – The CAN interface's name.

The interface name must be unique and can have 1–100 characters.

- (Optional) `protocolName` – The protocol's name.

Valid values: `CAN-FD` and `CAN`

- (Optional) `protocolVersion` – AWS IoT FleetWise currently supports `CAN-FD` and `CAN 2.0b`.

Valid values: `1.0` and `2.0b`

To configure an OBD network interface, specify the following information.

- `name` – The OBD interface's name.

The interface name must be unique and can have 1–100 characters.

- `requestMessageId` – The ID of the message that is requesting data.
- (Optional) `dtcRequestIntervalSeconds` – How often to request diagnostic trouble codes (DTCs) from the vehicle in seconds. For example, if the specified value is 120, the Edge Agent software collects stored DTCs once every 2 minutes.
- (Optional) `hasTransmissionEcu` – Whether the vehicle has a transmission control module (TCM).

Valid values: `true` and `false`

- (Optional) `obdStandard` – The OBD standard that AWS IoT FleetWise supports. AWS IoT FleetWise currently supports the World Wide Harmonization On-Board Diagnostics (WWH-OBD) ISO15765-4 standard.
- (Optional) `pidRequestIntervalSeconds` – How often to request OBD II PIDs from the vehicle. For example, if the specified value is 120, the Edge Agent software collects OBD II PIDs once every 2 minutes.
- (Optional) `useExtendedIds` – Whether to use extended IDs in the message.

Valid values: `true` and `false`

To configure a vehicle middleware network interface, specify the following information.

- `name` – The vehicle middleware interface's name.

The interface name must be unique and can have 1–100 characters.

- `protocolName` – The protocol's name.

Valid values: `ROS_2`

Configure decoder signals

To configure a CAN decoder signal, specify the following information.

- `factor` – The multiplier used to decode the message.
- `isBigEndian` – Whether the byte ordering of the message is big-endian. If it's big-endian, the most significant value in the sequence is stored first, at the lowest storage address.
- `isSigned` – Whether the message is signed. If it's signed, the message can represent both positive and negative numbers.
- `length` – The length of the message in bytes.
- `messageId` – The ID of the message.
- `offset` – The offset used to calculate the signal value. Combined with `factor`, the calculation is $value = raw_value * factor + offset$.
- `startBit` – Indicates the location of the first bit of the message.
- (Optional) `name` – The name of the signal.

To configure an OBD decoder signal, specify the following information.

- `byteLength` – The length of the message in bytes.
- `offset` – The offset used to calculate the signal value. Combined with `scaling`, the calculation is $value = raw_value * scaling + offset$.
- `pid` – The diagnostic code used to request a message from a vehicle for this signal.
- `pidResponseLength` – The length of the requested message.
- `scaling` – The multiplier used to decode the message.
- `serviceMode` – The mode of operation (diagnostic service) in a message.
- `startByte` – Indicates the beginning of the message.
- (Optional) `bitMaskLength` – The number of bits that are masked in a message.

- (Optional) `bitRightShift` – The number of positions shifted to the right.

To configure a message decoder signal, specify the following information.

- `topicName` – The topic name for the message signal. It corresponds to topics in ROS 2. For more information about the structured message object, see [StructuredMessage](#).
- `structuredMessage` – The structured message for the message signal. It can be defined with either a `primitiveMessageDefinition`, `structuredMessageListDefinition`, or `structuredMessageDefinition` recursively.

Create a decoder manifest

You can use the AWS IoT FleetWise console or API to create a decoder manifest for your vehicle model.

Important

You must have a vehicle model before you can create a decoder manifest. Every decoder manifest must be associated with a vehicle model. For more information, see [Create and manage vehicle models](#).

Topics

- [Create a decoder manifest \(console\)](#)
- [Create a decoder manifest \(AWS CLI\)](#)

Create a decoder manifest (console)

You can use the AWS IoT FleetWise console to create a decoder manifest that's associated with your vehicle model.

Important

You can't configure vision system data signals in decoder manifests using the AWS IoT FleetWise console. Instead, use the AWS CLI. Vision system data is in preview release and is subject to change.

To create a decoder manifest

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Vehicle models**.
3. Choose the target vehicle model.
4. On the vehicle model summary page, choose **Create decoder manifest**, and then do the following.

Topics

- [Step 1: Configure decoder manifest](#)
- [Step 2: Add network interfaces](#)
- [Step 3: Review and create](#)

Step 1: Configure decoder manifest

In **General information**, do the following.

1. Enter a unique name for the decoder manifest.
2. (Optional) Enter a description.
3. Choose **Next**.

Step 2: Add network interfaces

Each decoder manifest must have at least one network interface. You can add multiple network interfaces to a decoder manifest.

To add a network interface

- In **Network interface**, do the following.
 - a. For **Network interface type**, choose the **CAN_INTERFACE** or **OBD_INTERFACE**.
 - b. Enter a unique name for your network interface.
 - c. Enter a unique network interface ID. You can use the ID generated by AWS IoT FleetWise.
 - d. Select one or more signals specified in your vehicle model to pair with decoder signals.
 - e. To provide decoding information, upload a .dbc file. AWS IoT FleetWise parses the .dbc file to retrieve decoder signals.

- f. In the **Paired signals** section, make sure that every signal is paired with a decoder signal.
- g. Choose **Next**.

Note

- You can upload only one .dbc file for each network interface.
- Make sure that every signal specified in your vehicle model is paired with a decoder signal.
- After you choose to add another network interface, you can't edit the one that you're editing. You can delete any existing network interfaces.

Step 3: Review and create

Verify the configurations for the decoder manifest, and then choose **Create**.

Create a decoder manifest (AWS CLI)

You can use the [CreateDecoderManifest](#) API operation to create decoder manifests. The following example uses the AWS CLI.

Important

Before you create a decoder manifest, create a vehicle model first. For more information, see [Create a vehicle model](#).

To create a decoder manifest, run the following command.

Replace *decoder-manifest-configuration* with the name of the JSON file that contains the configuration.

```
aws iotfleetwise create-decoder-manifest --cli-input-json file://decoder-manifest-configuration.json
```

- Replace *decoder-manifest-name* with the name of the decoder manifest that you're creating.
- Replace *vehicle-model-ARN* with the Amazon Resource Name (ARN) of the vehicle-model.

- (Optional) Replace *description* with a description to help you identify the decoder manifest.

For more information about how to configure branches, attributes, sensors, and actuators, see [Configure network interfaces and decoder signals](#).

```
{
  "name": "decoder-manifest-name",
  "modelManifestArn": "vehicle-model-arn",
  "description": "description",
  "networkInterfaces": [
    {
      "canInterface": {
        "name": "myNetworkInterface",
        "protocolName": "CAN",
        "protocolVersion": "2.0b"
      },
      "interfaceId": "Qq1acaenBy0B3sSM39SYm",
      "type": "CAN_INTERFACE"
    }
  ],
  "signalDecoders": [
    {
      "canSignal": {
        "name": "Engine_Idle_Time",
        "factor": 1,
        "isBigEndian": true,
        "isSigned": false,
        "length": 24,
        "messageId": 271343712,
        "offset": 0,
        "startBit": 16
      },
      "fullyQualifiedName": "Vehicle.EngineIdleTime",
      "interfaceId": "Qq1acaenBy0B3sSM39SYm",
      "type": "CAN_SIGNAL"
    },
    {
      "canSignal": {
        "name": "Engine_Run_Time",
        "factor": 1,
        "isBigEndian": true,
        "isSigned": false,
        "length": 24,
```

```

        "messageId": 271343712,
        "offset": 0,
        "startBit": 40
    },
    "fullyQualified_name": "Vehicle.EngineRunTime",
    "interfaceId": "Qq1acaenBy0B3sSM39SYm",
    "type": "CAN_SIGNAL"
}
]
}

```

- Replace *decoder-manifest-name* with the name of the decoder manifest that you're creating.
- Replace *vehicle-model-ARN* with the Amazon Resource Name (ARN) of the vehicle-model.
- (Optional) Replace *description* with a description to help you identify the decoder manifest.

The order of property nodes within a structure (struct) must remain consistent as defined in the signal catalog and vehicle model (model manifest). For more information about how to configure branches, attributes, sensors, and actuators, see [Configure network interfaces and decoder signals](#).

```

{
  "name": "decoder-manifest-name",
  "modelManifestArn": "vehicle-model-arn",
  "description": "description",
  "networkInterfaces": [{
    "canInterface": {
      "name": "myNetworkInterface",
      "protocolName": "CAN",
      "protocolVersion": "2.0b"
    },
    "interfaceId": "Qq1acaenBy0B3sSM39SYm",
    "type": "CAN_INTERFACE"
  }, {
    "type": "VEHICLE_MIDDLEWARE",
    "interfaceId": "G1KzxkdnmV5Hn7wkV3ZL9",
    "vehicleMiddleware": {
      "name": "ROS2_test",
      "protocolName": "ROS_2"
    }
  }
  ]],
  "signalDecoders": [{
    "canSignal": {

```

```

    "name": "Engine_Idle_Time",
    "factor": 1,
    "isBigEndian": true,
    "isSigned": false,
    "length": 24,
    "messageId": 271343712,
    "offset": 0,
    "startBit": 16
  },
  "fullyQualifiedName": "Vehicle.EngineIdleTime",
  "interfaceId": "Qq1acaenByOB3sSM39SYm",
  "type": "CAN_SIGNAL"
},
{
  "canSignal": {
    "name": "Engine_Run_Time",
    "factor": 1,
    "isBigEndian": true,
    "isSigned": false,
    "length": 24,
    "messageId": 271343712,
    "offset": 0,
    "startBit": 40
  },
  "fullyQualifiedName": "Vehicle.EngineRunTime",
  "interfaceId": "Qq1acaenByOB3sSM39SYm",
  "type": "CAN_SIGNAL"
},
{
  "fullyQualifiedName": "Vehicle.CompressedImageTopic",
  "type": "MESSAGE_SIGNAL",
  "interfaceId": "G1KzxkdnmV5Hn7wkV3ZL9",
  "messageSignal": {
    "topicName": "CompressedImageTopic:sensor_msgs/msg/CompressedImage",
    "structuredMessage": {
      "structuredMessageDefinition": [{
        "fieldName": "header",
        "dataType": {
          "structuredMessageDefinition": [{
            "fieldName": "stamp",
            "dataType": {
              "structuredMessageDefinition": [{
                "fieldName": "sec",
                "dataType": {

```

```
    "primitiveMessageDefinition": {
      "ros2PrimitiveMessageDefinition": {
        "primitiveType": "INT32"
      }
    }
  },
  {
    "fieldName": "nanosec",
    "dataType": {
      "primitiveMessageDefinition": {
        "ros2PrimitiveMessageDefinition": {
          "primitiveType": "UINT32"
        }
      }
    }
  }
]
},
{
  "fieldName": "frame_id",
  "dataType": {
    "primitiveMessageDefinition": {
      "ros2PrimitiveMessageDefinition": {
        "primitiveType": "STRING"
      }
    }
  }
}
],
{
  "fieldName": "format",
  "dataType": {
    "primitiveMessageDefinition": {
      "ros2PrimitiveMessageDefinition": {
        "primitiveType": "STRING"
      }
    }
  }
},
{
```



```
--name decoder-manifest-name /  
--status ACTIVE
```

Important

After you activate the decoder manifest, you can't edit it.

Delete a decoder manifest

You can use the AWS IoT FleetWise console or API to delete a decoder manifest.

Important

Vehicles associated with the decoder manifest must be deleted first. For more information, see [Delete a vehicle](#).

Topics

- [Delete a decoder manifest \(console\)](#)
- [Delete a decoder manifest \(AWS CLI\)](#)

Delete a decoder manifest (console)

You can use the AWS IoT FleetWise console to delete a decoder manifest.

To delete a decoder manifest

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Vehicle models**.
3. Choose the target vehicle model.
4. On the vehicle model summary page, choose the **Decoder manifests** tab.
5. Choose the target decoder manifest, and then choose **Delete**.
6. In **Delete decoder-manifest-name?**, enter the name of the decoder manifest to delete, and then choose **Confirm**.

Delete a decoder manifest (AWS CLI)

You can use the [DeleteDecoderManifest](#) API operation to delete a decoder manifest. The following example uses AWS CLI.

Important

Before you delete the decoder manifest, delete the associated vehicles first. For more information, see [Delete a vehicle](#).

To delete a decoder manifest, run the following command.

Replace *decoder-manifest-name* with the name of the decoder manifest that you're deleting.

```
aws iotfleetwise delete-decoder-manifest --name decoder-manifest-name
```

Get decoder manifest information (AWS CLI)

You can use the [ListDecoderManifests](#) API operation to verify if a decoder manifest has been deleted. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all decoder manifests, run the following command.

```
aws iotfleetwise list-decoder-manifests
```

You can use the [ListDecoderManifestSignals](#) API operation to verify if decoder signals in the decoder manifest have been updated. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all decoder signals (nodes) in a given decoder manifest, run the following command.

Replace *decoder-manifest-name* with the name of the decoder manifest that you're checking.

```
aws iotfleetwise list-decoder-manifest-signals /  
    --name decoder-manifest-name
```

You can use the [ListDecoderManifestNetworkInterfaces](#) API operation to verify if network interfaces in the decoder manifest have been updated. The following example uses AWS CLI.

To retrieve a paginated list of summaries of all network interfaces in a given decoder manifest, run the following command.

Replace *decoder-manifest-name* with the name of the decoder manifest that you're checking.


```
aws iotfleetwise list-decoder-manifest-network-interfaces /  
    --name decoder-manifest-name
```

You can use the [GetDecoderManifest](#) API operation to verify if network interfaces and decoder signals in the decoder manifest have been updated. The following example uses AWS CLI.

To retrieve information about a decoder manifest, run the following command.

Replace *decoder-manifest* with the name of the decoder manifest that you want to retrieve.

```
aws iotfleetwise get-decoder-manifest --name decoder-manifest
```

 **Note**

This operation is [eventually consistent](#). In other words, changes to the decoder manifest might not be reflected immediately.

Create, provision, and manage vehicles

Vehicles are instances of vehicle models. Vehicles must be created from a vehicle model and associated with a decoder manifest. Vehicles uploads one or more data streams to the cloud. For example, a vehicle can send mileage, engine temperature, and state of heater data to the cloud. Every vehicle contains the following information:

vehicleName

An ID that identifies the vehicle.

Do not add personally identifiable information (PII) or other confidential or sensitive information in your vehicle name. Vehicle names are accessible by other AWS services, including Amazon CloudWatch. Vehicle names aren't intended to be used for private or sensitive data.

modelManifestARN

The Amazon Resource Name (ARN) of a vehicle model (model manifest). Every vehicle is created from a vehicle model. Vehicles created from the same vehicle model consist of the same group of signals inherited from the vehicle model. These signals are defined and standardized in the signal catalog.

decoderManifestArn

The ARN of the decoder manifest. A decoder manifest provides decoding information that AWS IoT FleetWise can use to transform raw signal data (binary data) into human-readable values. A decoder manifest must be associated with a vehicle model. AWS IoT FleetWise uses the same decoder manifest to decode raw data from vehicles created based on the same vehicle model.

attributes

Attributes are key-value pairs that contain static information. Vehicles can contain attributes inherited from the vehicle model. You can add additional attributes to distinguish an individual vehicle from other vehicles created from the same vehicle model. For example, if you have a black car, you can specify the following value for an attribute: `{"color": "black"}`.

Important

Attributes must be defined in the associated vehicle model before you can add them to individual vehicles.

For more information about vehicle models, decoder manifests, and attributes, see [Modeling vehicles](#).

AWS IoT FleetWise provides the following API operations that you can use to create and manage vehicles.

- [CreateVehicle](#) – Creates a new vehicle.
- [BatchCreateVehicle](#) – Creates one or more new vehicles.
- [UpdateVehicle](#) – Updates an existing vehicle.
- [BatchUpdateVehicle](#) – Updates one or more existing vehicles.
- [DeleteVehicle](#) – Deletes an existing vehicle.
- [ListVehicles](#) – Retrieves a paginated list of summaries of all vehicles.
- [GetVehicle](#) – Retrieves information about a vehicle.

Tutorials

- [Provision vehicles](#)
- [Reserved topics](#)
- [Create a vehicle](#)
- [Update a vehicle \(AWS CLI\)](#)
- [Update multiple vehicles \(AWS CLI\)](#)
- [Delete a vehicle](#)
- [Get vehicle information \(AWS CLI\)](#)

Provision vehicles

The Edge Agent for AWS IoT FleetWise software running in your vehicle collects and transfers data to the cloud. AWS IoT FleetWise integrates with AWS IoT Core to support secure communication between the Edge Agent software and the cloud through MQTT. Each vehicle corresponds to an AWS IoT thing. You can use an existing AWS IoT thing to create a vehicle or set AWS IoT FleetWise to automatically create an AWS IoT thing for your vehicle. For more information, see [Create a vehicle \(AWS CLI\)](#).

AWS IoT Core supports [authentication](#) and [authorization](#) that help securely control access to AWS IoT FleetWise resources. Vehicles can use X.509 certificates to get authenticated (signed in) to use

AWS IoT FleetWise and AWS IoT Core policies to get authorized (have permissions) to perform specified actions.

Authenticate vehicles

You can create AWS IoT Core policies to authenticate your vehicles.

To authenticate your vehicle

- To create an AWS IoT Core policy, run the following command.
 - Replace *policy-name* with the name of the policy that you want to create.
 - Replace *file-name* with the name of the JSON file that contains the AWS IoT Core policy.

```
aws iot create-policy --policy-name policy-name --policy-document file://file-name.json
```

Before you use the example policy, do the following:

- Replace *region* with the AWS Region where you created AWS IoT FleetWise resources.
- Replace *awsAccount* with your AWS account ID.

This example includes topics reserved by AWS IoT FleetWise. You must add the topics to the policy. For more information, see [Reserved topics](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:region:awsAccount:client/
${iot:Connection.Thing.ThingName}"
      ]
    },
    {
```

```

        "Effect": "Allow",
        "Action": [
            "iot:Publish"
        ],
        "Resource": [
            "arn:aws:iot:region:awsAccount:topic/$aws/iotfleetwise/vehicles/
            ${iot:Connection.Thing.ThingName}/checkins",
            "arn:aws:iot:region:awsAccount:topic/$aws/iotfleetwise/vehicles/
            ${iot:Connection.Thing.ThingName}/signals"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "iot:Subscribe"
        ],
        "Resource": [
            "arn:aws:iot:region:awsAccount:topicfilter/$aws/iotfleetwise/
            vehicles/${iot:Connection.Thing.ThingName}/collection_schemes",
            "arn:aws:iot:region:awsAccount:topicfilter/$aws/iotfleetwise/
            vehicles/${iot:Connection.Thing.ThingName}/decoder_manifests"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "iot:Receive"
        ],
        "Resource": [
            "arn:aws:iot:region:awsAccount:topic/$aws/iotfleetwise/vehicles/
            ${iot:Connection.Thing.ThingName}/collection_schemes",
            "arn:aws:iot:region:awsAccount:topic/$aws/iotfleetwise/vehicles/
            ${iot:Connection.Thing.ThingName}/decoder_manifests"
        ]
    }
]
}

```

Authorize vehicles

You can create X.509 certificates to authorize your vehicles.

To authorize your vehicle

Important

We recommend that you create a new certificate for each vehicle.

1. To create an RSA key pair and issue an X.509 certificate, run the following command.
 - Replace *cert* with the name of the file that saves the command output contents of `certificatePem`.
 - Replace *public-key* with the name of the file that saves the command output contents of `keyPair.PublicKey`.
 - Replace *private-key* with the name of the file that saves the command output contents of `keyPair.PrivateKey`.

```
aws iot create-keys-and-certificate \  
  --set-as-active \  
  --certificate-pem-outfile cert.pem \  
  --public-key-outfile public-key.key" \  
  --private-key-outfile private-key.key"
```

2. Copy the Amazon Resource Name (ARN) of the certificate from the output.
3. To attach the policy to the certificate, run the following command.
 - Replace *policy-name* with the name of the AWS IoT Core policy that you created.
 - Replace *certificate-arn* with the ARN of the certificate that you copied.

```
aws iot attach-policy \  
  --policy-name policy-name \  
  --target "certificate-arn"
```

4. To attach the certificate to the thing, run the following command.
 - Replace *thing-name* with the name of your AWS IoT thing or the ID of your vehicle.
 - Replace *certificate-arn* with the ARN of the certificate that you copied.

```
aws iot attach-thing-principal \
  --thing-name thing-name \
  --principal "certificate-arn"
```

Reserved topics

AWS IoT FleetWise reserves the use of the following topics. If the reserved topic allows, you can subscribe or publish to it. However, you can't create new topics that begin with a dollar sign (\$). If you use unsupported publish or subscribe operations with reserved topics, it can result in the connection ending.

Topic	Client operation allowed	Description
\$aws/iotfleetwise/vehicles/ <i>vehicleName</i> /checkins	Publish	The Edge Agent software publishes vehicle status information to this topic. Vehicle status information is exchanged in protocol buffers (protobuf) format. For more information, see the Edge Agent for AWS IoT FleetWise software Developer Guide .
\$aws/iotfleetwise/vehicles/	Publish	The Edge Agent software publishes signals to this topic.

Topic	Client operation allowed	Description
<code>vehicleName / signals</code>		Signal information is exchanged in protocol buffers (protobuf) format. For more information, see the Edge Agent for AWS IoT FleetWise software Developer Guide .
<code>\$aws/iotfleetwise/vehicles/vehicleName /collection_schemes</code>	Subscribe	AWS IoT FleetWise publishes data collection schemes to this topic. Vehicles consume these data collection schemes.
<code>\$aws/iotfleetwise/vehicles/vehicleName /decoder_manifests</code>	Subscribe	AWS IoT FleetWise publishes decoder manifests to this topic. Vehicles consume these decoder manifests.

Create a vehicle

You can use the AWS IoT FleetWise console or API to create a vehicle.

Important

Before you start, check the following:

- You must have a vehicle model and the status of the vehicle model must be ACTIVE. For more information, see [Create and manage vehicle models](#).

- Your vehicle model must be associated with a decoder manifest, and the status of the decoder manifest must be ACTIVE. For more information, see [Create and manage decoder manifests](#).

Topics

- [Create a vehicle \(console\)](#)
- [Create a vehicle \(AWS CLI\)](#)
- [Create multiple vehicles \(AWS CLI\)](#)

Create a vehicle (console)

You can use the AWS IoT FleetWise console to create a vehicle.

Important

Before you start, check the following:

- You must have a vehicle model and the status of the vehicle model must be ACTIVE. For more information, see [Create and manage vehicle models](#).
- Your vehicle model must be associated with a decoder manifest, and the status of the decoder manifest must be ACTIVE. For more information, see [Create and manage decoder manifests](#).

To create a vehicle

1. Open the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Vehicles**.
3. On the vehicle summary page, choose **Create vehicle**, and then do the following steps.

Topics

- [Step 1: Define vehicle properties](#)
- [Step 2: Configure vehicle certificate](#)
- [Step 3: Attach policies to certificate](#)

- [Step 4: Review and create](#)

Step 1: Define vehicle properties

In this step, you name the vehicle and associate it with the model manifest and decoder manifest.

1. Enter a unique name for the vehicle.

Important

A vehicle corresponds to an AWS IoT thing. If a thing already exists with that name, choose **Associate the vehicle with an IoT thing** to update the thing with the vehicle. Or, choose a different vehicle name and AWS IoT FleetWise will automatically create a new thing for the vehicle.

2. Choose a vehicle model (model manifest) from the list.
3. Choose a decoder manifest from the list. The decoder manifest is associated with the vehicle model.
4. (Optional) To associate vehicle attributes, choose **Add attributes**. If you skip this step, you must add attributes after the vehicle is created before you can deploy it to campaigns.
5. (Optional) To associate tags with the vehicle, choose **Add new tag**. You can also add tags after the vehicle is created.
6. Choose **Next**.

Step 2: Configure vehicle certificate

To use your vehicle as an AWS IoT thing, you must configure a vehicle certificate with an attached policy. If you skip this step, you must configure a certificate after the vehicle is created before you can deploy it to campaigns.

1. Choose **Auto-generate a new certificate (recommended)**.
2. Choose **Next**.

Step 3: Attach policies to certificate

Attach a policy to the certificate you configured in the previous step.

1. For **Policies**, enter an existing policy name. To create a new policy, choose **Create policy**.
2. Choose **Next**.

Step 4: Review and create

Verify the configurations for the vehicle, and then choose **Create vehicle**.

Important

After the vehicle is created, you must download the certificate and keys. You'll use the certificate and private key to connect the vehicle in the Edge Agent for AWS IoT FleetWise software.

Create a vehicle (AWS CLI)

When you create a vehicle, you must use a vehicle model that is associated with a decoder manifest. You can use the [CreateVehicle](#) API operation to create a vehicle. The following example uses the AWS CLI.

Important

Before you start, check the following:

- You must have a vehicle model and the status of the vehicle model must be ACTIVE. For more information, see [Create and manage vehicle models](#).
- Your vehicle model must be associated with a decoder manifest, and the status of the decoder manifest must be ACTIVE. For more information, see [Create and manage decoder manifests](#).

To create a vehicle, run the following command.

Replace *file-name* with the name of the JSON file that contains the vehicle configuration.

```
aws iotfleetwise create-vehicle --cli-input-json file://file-name.json
```

Example vehicle configuration

- (Optional) The `associationBehavior` value can be one of the following:
 - `CreateIotThing` – When your vehicle is created, AWS IoT FleetWise automatically creates an AWS IoT thing with the name of your vehicle ID for your vehicle.
 - `ValidateIotThingExists` – Use an existing AWS IoT thing to create a vehicle.

To create an AWS IoT thing, run the following command. Replace *thing-name* with the name of the thing you want to create.

```
aws iot create-thing --thing-name thing-name
```

If it's not specified, AWS IoT FleetWise automatically creates an AWS IoT thing for your vehicle.

Important

Make sure that the AWS IoT thing is provisioned after the vehicle is created. For more information, see [Provision vehicles](#).

- Replace *vehicle-name* with one of the following.
 - The name of your AWS IoT thing if `associationBehavior` is configured to `ValidateIotThingExists`.
 - The ID of the vehicle to create if `associationBehavior` is configured to `CreateIotThing`.

The vehicle ID can have 1–100 characters. Valid characters: a–z, A–Z, 0–9, dash (-), underscore (_), and colon (:).

- Replace *model-manifest-ARN* with the ARN of your vehicle model (model manifest).
- Replace *decoder-manifest-ARN* with the ARN of the decoder manifest associated with the specified vehicle model.
- (Optional) You can add additional attributes to distinguish this vehicle from other vehicles created from the same vehicle model. For example, if you have an electric car, you can specify the following value for an attribute: `{"fuelType": "electric"}`.

⚠ Important

Attributes must be defined in the associated vehicle model before you can add them to individual vehicles.

```
{
  "associationBehavior": "associationBehavior",
  "vehicleName": "vehicle-name",
  "modelManifestArn": "model-manifest-ARN",
  "decoderManifestArn": "decoder-manifest-ARN",
  "attributes": {
    "key": "value"
  }
}
```

Create multiple vehicles (AWS CLI)

You can use the [BatchCreateVehicle](#) API operation to create multiple vehicles at one time. The following example uses the AWS CLI.

To create multiple vehicles, run the following command.

Replace *file-name* with the name of the JSON file that contains the configurations of multiple vehicles.

```
aws iotfleetwise batch-create-vehicle --cli-input-json file://file-name.json
```

Example vehicle configurations

```
{
  "vehicles": [
    {
      "associationBehavior": "associationBehavior",
      "vehicleName": "vehicle-name",
      "modelManifestArn": "model-manifest-ARN",
      "decoderManifestArn": "decoder-manifest-ARN",
      "attributes": {
        "key": "value"
      }
    }
  ]
}
```

```
    },
    {
      "associationBehavior": "associationBehavior",
      "vehicleName": "vehicle-name",
      "modelManifestArn": "model-manifest-ARN",
      "decoderManifestArn": "decoder-manifest-ARN",
      "attributes": {
        "key": "value"
      }
    }
  ]
}
```

You can create up to 10 vehicles for each batch operation. For more information about the vehicle configuration, see [Create a vehicle \(AWS CLI\)](#).

Update a vehicle (AWS CLI)

You can use the [UpdateVehicle](#) API operation to update an existing vehicle. The following example uses the AWS CLI.

To update a vehicle, run the following command.

Replace *file-name* with the name of the JSON file that contains the configuration of your vehicle.

```
aws iotfleetwise update-vehicle --cli-input-json file://file-name.json
```

Example vehicle configuration

- Replace *vehicle-name* with the ID of the vehicle that you want to update.
- (Optional) Replace *model-manifest-ARN* with the ARN of the vehicle model (model manifest) that you use to replace the vehicle model in use.
- (Optional) Replace *decoder-manifest-ARN* with the ARN of your decoder manifest associated with the new vehicle model that you specified.
- (Optional) Replace *attribute-update-mode* with vehicle attributes.
 - Merge – Merge new attributes into existing attributes by updating existing attributes with new values and adding new attributes if they don't exist.

For example, if a vehicle has the following attributes: {"color": "black", "fuelType": "electric"}, and you update the vehicle with the following attributes: {"color": "", "fuelType": "gasoline", "model": "x"}, the updated vehicle has the following attributes: {"fuelType": "gasoline", "model": "x"}.

- **Overwrite** – Replace existing attributes with new attributes.

For example, if a vehicle has the following attributes: {"color": "black", "fuelType": "electric"}, and you update the vehicle with the {"model": "x"} attribute, the updated vehicle has the {"model": "x"} attribute.

This is required if attributes are present in the input.

- (Optional) To add new attributes or update existing ones with new values, configure attributes. For example, if you have an electric car, you can specify the following value for an attribute: {"fuelType": "electric"}.

To delete attributes, configure `attributeUpdateMode` to `Merge`.

Important

Attributes must be defined in the associated vehicle model before you can add them to individual vehicles.

```
{
  "vehicleName": "vehicle-name",
  "modelManifestArn": "model-manifest-arn",
  "decoderManifestArn": "decoder-manifest-arn",
  "attributeUpdateMode": "attribute-update-mode"
}
```

Update multiple vehicles (AWS CLI)

You can use the [BatchUpdateVehicle](#) API operation to update multiple existing vehicles at one time. The following example uses the AWS CLI.

To update multiple vehicles, run the following command.

Replace *file-name* with the name of the JSON file that contains the configurations of multiple vehicles.

```
aws iotfleetwise batch-update-vehicle --cli-input-json file://file-name.json
```

Example vehicle configurations

```
{
  "vehicles": [
    {
      "vehicleName": "vehicle-name",
      "modelManifestArn": "model-manifest-arn",
      "decoderManifestArn": "decoder-manifest-arn",
      "mergeAttributes": true,
      "attributes": {
        "key": "value"
      }
    },
    {
      "vehicleName": "vehicle-name",
      "modelManifestArn": "model-manifest-arn",
      "decoderManifestArn": "decoder-manifest-arn",
      "mergeAttributes": true,
      "attributes": {
        "key": "value"
      }
    }
  ]
}
```

You can update up to 10 vehicles for each batch operation. For more information about the configuration of each vehicle, see [Update a vehicle \(AWS CLI\)](#).

Delete a vehicle

You can use the AWS IoT FleetWise console or API to delete vehicles.

Important

After a vehicle is deleted, AWS IoT FleetWise automatically removes the vehicle from the associated fleets and campaigns. For more information, see [Create and manage fleets](#) and

[Collect and transfer data with campaigns](#). However, the vehicle still exists as a thing or is still associated with a thing in AWS IoT Core. For instructions on deleting a thing, see [Delete a thing](#) in the *AWS IoT Core Developer Guide*.

Delete a vehicle (console)

You can use the AWS IoT FleetWise console to delete a vehicle.

To delete a vehicle

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Vehicles**.
3. On the **Vehicles** page, select the button next to the vehicle you want to delete.
4. Choose **Delete**.
5. In **Delete vehicle-name**, enter the name of the vehicle, and then choose **Delete**.

Delete a vehicle (AWS CLI)

You can use the [DeleteVehicle](#) API operation to delete a vehicle. The following example uses AWS CLI.

To delete a vehicle, run the following command.

Replace *vehicle-name* with the ID of the vehicle that you want to delete.

```
aws iotfleetwise delete-vehicle --vehicle-name vehicle-name
```

Get vehicle information (AWS CLI)

You can use the [ListVehicles](#) API operation to verify if a vehicle has been deleted. The following example uses the AWS CLI.

To retrieve a paginated list of summaries of all vehicles, run the following command.


```
aws iotfleetwise list-vehicles
```


You can use the [GetVehicle](#) API operation to retrieve vehicle information. The following example uses the AWS CLI.

To retrieve the metadata of a vehicle, run the following command.

Replace *vehicle-name* with the ID of the vehicle that you want to retrieve.

```
aws iotfleetwise get-vehicle --vehicle-name vehicle-name
```

 **Note**

This operation is [eventually consistent](#). In other words, changes to the vehicle might not be reflected immediately.

Create and manage fleets

A fleet represents a group of vehicles. A fleet without associated vehicles is an empty entity. Before you can use the fleet to manage multiple vehicles at the same time, you must associate vehicles with the fleet. A vehicle can belong to multiple fleets. You can control what data to collect from a fleet of vehicles and when to collect data by deploying a campaign. For more information, see [Collect and transfer data with campaigns](#).

A fleet contains the following information.

`fleetId`

The ID of the fleet.

(Optional) `description`

A description that helps you find the fleet.

`signalCatalogArn`

The Amazon Resource Name (ARN) of the signal catalog.

AWS IoT FleetWise provides the following API operations that you can use to create and manage fleets.

- [CreateFleet](#) – Creates a group of vehicles that contain the same group of signals.
- [AssociateVehicleFleet](#) – Associates a vehicle to a fleet.
- [DisassociateVehicleFleet](#) – Disassociates a vehicle from a fleet.
- [UpdateFleet](#) – Updates the description for an existing fleet.
- [DeleteFleet](#) – Deletes an existing fleet.
- [ListFleets](#) – Retrieves a paginated list of summaries of all fleets.
- [ListFleetsForVehicle](#) – Retrieves a paginated list of IDs of all fleets that the vehicle belongs to.
- [ListVehiclesInFleet](#) – Retrieves a paginated list of summaries of all vehicles in a fleet.
- [GetFleet](#) – Retrieves information about a fleet.

Topics

- [Create a fleet \(AWS CLI\)](#)

- [Associate a vehicle with a fleet \(AWS CLI\)](#)
- [Disassociate a vehicle from a fleet \(AWS CLI\)](#)
- [Update a fleet \(AWS CLI\)](#)
- [Delete a fleet \(AWS CLI\)](#)
- [Get fleet information \(AWS CLI\)](#)

Create a fleet (AWS CLI)

You can use the [CreateFleet](#) API operation to create a vehicle fleet. The following example uses AWS CLI.

Important

You must have a signal catalog before you can create a fleet. For more information, see [Create a signal catalog \(AWS CLI\)](#).

To create a fleet, run the following command.

- Replace *fleet-id* with the ID of the fleet that you're creating.

The fleet ID must be unique and have 1-100 characters. Valid characters: letters (A-Z and a-z), numbers (0-9), colons (:), dashes (-), and underscores (_).

- (Optional) Replace *description* with a description.

The description can have 1-2048 characters.

- Replace *signal-catalog-arn* with the ARN of the signal catalog.

```
aws iotfleetwise create-fleet \  
  --fleet-id fleet-id \  
  --description description \  
  --signal-catalog-arn signal-catalog-arn
```

Associate a vehicle with a fleet (AWS CLI)

You can use the [AssociateVehicleFleet](#) API operation to associate a vehicle with a fleet. The following example uses AWS CLI.

Important

- You must have a vehicle and a fleet before you can associate a vehicle with a fleet. For more information, see [Create, provision, and manage vehicles](#).
- If you associate a vehicle with a fleet that is targeted by a campaign, AWS IoT FleetWise automatically deploys the campaign to the vehicle.

To associate a vehicle with a fleet, run the following command.

- Replace *fleet-id* with the ID of the fleet.
- Replace *vehicle-name* with the ID of the vehicle.

```
aws iotfleetwise associate-vehicle-fleet --fleet-id fleet-id --vehicle-name vehicle-name
```

Disassociate a vehicle from a fleet (AWS CLI)

You can use the [DisassociateVehicleFleet](#) API operation to disassociate a vehicle from a fleet. The following example uses AWS CLI.

To disassociate a vehicle with a fleet, run the following command.

- Replace *fleet-id* with the ID of the fleet.
- Replace *vehicle-name* with the ID of the vehicle.

```
aws iotfleetwise disassociate-vehicle-fleet --fleet-id fleet-id --vehicle-name vehicle-name
```

Update a fleet (AWS CLI)

You can use the [UpdateFleet](#) API operation to update the description for a fleet. The following example uses AWS CLI.

To update a fleet, run the following command.

- Replace *fleet-id* with the ID of the fleet that you're updating.
- Replace *description* with a new description.

The description can have 1-2048 characters.

```
aws iotfleetwise update-fleet --fleet-id fleet-id --description description
```

Delete a fleet (AWS CLI)

You can use the [DeleteFleet](#) API operation to delete a fleet. The following example uses AWS CLI.

Important

Before you delete a fleet, make sure it has no associated vehicles. For instructions on how to disassociate a vehicle from a fleet, see [Disassociate a vehicle from a fleet \(AWS CLI\)](#).

To delete a fleet, run the following command.

Replace *fleet-id* with the ID of the fleet that you're deleting.

```
aws iotfleetwise delete-fleet --fleet-id fleet-id
```

Get fleet information (AWS CLI)

You can use the [ListFleets](#) API operation to verify if a fleet has been deleted. The following example uses the AWS CLI.

To retrieve a paginated list of summaries of all fleets, run the following command.

```
aws iotfleetwise list-fleets
```

You can use the [ListFleetsForVehicle](#) API operation to retrieve a paginated list of IDs of all fleets that the vehicle belongs to. The following example uses the AWS CLI.

To retrieve a paginated list of IDs of all fleets that the vehicle belongs to, run the following command.

Replace *vehicle-name* with the ID of the vehicle.

```
aws iotfleetwise list-fleets-for-vehicle \  
  --vehicle-name vehicle-name
```

You can use the [ListVehiclesInFleet](#) API operation to retrieve a paginated list of summaries of all vehicles in a fleet. The following example uses the AWS CLI.

To retrieve a paginated list of summaries of all vehicles in a fleet, run the following command.

Replace *fleet-id* with the ID of the fleet.

```
aws iotfleetwise list-vehicles-in-fleet \  
  --fleet-id fleet-id
```

You can use the [GetFleet](#) API operation to retrieve fleet information. The following example uses the AWS CLI.

To retrieve the metadata of a fleet, run the following command.

Replace *fleet-id* with the ID of the fleet.

```
aws iotfleetwise get-fleet \  
  --fleet-id fleet-id
```

 **Note**

This operation is [eventually consistent](#). In other words, changes to the fleet might not be reflected immediately.

Collect and transfer data with campaigns

A campaign is an orchestration of data collection rules. Campaigns give the Edge Agent for AWS IoT FleetWise software instructions on how to select, collect, and transfer data to the cloud.

You create campaigns in the cloud. After you or your team has approved a campaign, AWS IoT FleetWise automatically deploys it to vehicles. You can choose to deploy a campaign to a vehicle or a fleet of vehicles. The Edge Agent software doesn't start collecting data until a running campaign is deployed to the vehicle.

Note

Campaigns won't work until you have the following.

- The Edge Agent software is running in your vehicle. For more information about how to develop, install, and work with the Edge Agent software, do the following.
 1. Navigate to the [AWS IoT FleetWise console](#).
 2. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.
- You've set up AWS IoT Core to provision your vehicle. For more information, see [Provision vehicles](#).

Each campaign contains the following information.

`signalCatalogArn`

The Amazon Resource Name (ARN) of the signal catalog associated with the campaign.

(Optional) tags

Tags are metadata that can be used to manage the campaign. You can assign the same tag to resources from different services to indicate that the resources are related.

`TargetArn`

The ARN of a vehicle or fleet to which the campaign is deployed.

`name`

A unique name that helps identify the campaign.

collectionScheme

The data collection schemes give Edge Agent software instructions on what data to collect or when to collect it. AWS IoT FleetWise currently supports the condition-based collection scheme and the time-based collection scheme.

conditionBasedCollectionScheme

The condition-based collection scheme uses a logical expression to recognize what data to collect. The Edge Agent software collects data when the condition is met.

expression

The logical expression used to recognize what data to collect. For example, if the `$variable.`myVehicle.InVehicleTemperature` > 50.0` expression is specified, the Edge Agent software collects temperature values that are greater than 50.0. For instructions on how to write expressions, see [Logical expressions for campaigns](#).

(Optional) `triggerMode` can be one of the following values.

- `RISING_EDGE` – The Edge Agent software collects data only when the condition is met for the first time. For example, `$variable.`myVehicle.AirBagDeployed` == true`.
- `ALWAYS` – Edge Agent software collects data whenever the condition is met.

(Optional) `minimumTriggerIntervalMs`

The minimum duration of time between two data collection events, in milliseconds. If a signal changes often, you might collect data at a slower rate.

(Optional) `conditionLanguageVersion`

The version of the conditional expression language.

timeBasedCollectionScheme

When you define a time-based collection scheme, specify a time period in milliseconds. The Edge Agent software uses the time period to decide how often to collect data. For example, if the time period is 120,000 milliseconds, the Edge Agent software collects data once every two minutes.

(Optional) `compression`

To save wireless bandwidth and reduce network traffic, you can specify [SNAPPY](#) to compress data in vehicles.

By default (OFF), the Edge Agent software doesn't compress data.

`dataDestinationConfigs`

Choose the destination where the campaign will transfer vehicle data. You can choose to store data in Amazon S3 or Amazon Timestream.

S3 is a cost-effective data storage mechanism that offers durable data management capabilities and downstream data services. You can use S3 for data related to driving behaviors or analyzing long-term maintenance.

Timestream is a data persistence mechanism that can help you identify trends and patterns in near real time. You can use Timestream for time-series data, such as to analyze historical trends in vehicle speed or braking.

(Optional) `dataExtraDimensions`

You can add one or more attributes to provide additional information for a signal.

(Optional) `description`

You can add a description to help identify the campaign's purpose.

(Optional) `diagnosticsMode`

When the diagnostics mode is configured to `SEND_ACTIVE_DTCS`, the campaign sends stored, standard diagnostic trouble codes (DTCs) that help identify what is wrong with your vehicle. For example, P0097 indicates the engine control module (ECM) has determined that the intake air temperature sensor 2 (IAT2) input is lower than the normal sensor range.

By default (OFF), the Edge Agent software doesn't send diagnostic codes.

(Optional) `expiryTime`

You can define the expiration date for your campaign. When the campaign expires, the Edge Agent software stops collecting data as specified in this campaign. If multiple campaigns are deployed to the vehicle, the Edge Agent software uses other campaigns to collect data.

Default value: 253402243200 (December 31, 9999, 00:00:00 UTC)

(Optional) `postTriggerCollectionDuration`

You can define a post-trigger collection duration, so that the Edge Agent software continues collecting data for a specified period after a scheme is invoked. For example,

if a condition-based collection scheme with the following expression is invoked:
`$variable.`myVehicle.Engine.RPM` > 7000.0`, the Edge Agent software continues to collect revolutions per minute (RPM) values for the engine. Even if the RPM only goes higher than 7000 once, it might indicate that there's a mechanical issue. In this case, you might want the Edge Agent software to continue collecting data to help monitor the condition.

Default value: 0

(Optional) `priority`

You can specify an integer to indicate the priority level of the campaign. Campaigns with a smaller number are higher priorities. If you deploy multiple campaigns to a vehicle, the campaigns that are higher priorities are initiated first.

Default value: 0

(Optional) `signalsToCollect`

A list of signals from which data is collected when the data collection scheme is invoked.

 **Important**

Signals used in the expression for the condition-based collection scheme must be specified in this field.

`name`

The name of the signal from which data is collected when the data collection scheme is invoked.

(Optional) `maxSampleCount`

The maximum number of data samples that the Edge Agent software collects and transfers to the cloud when the data collection scheme is invoked.

(Optional) `minimumSamplingIntervalMs`

The minimum duration of time between two data sample collection events, in milliseconds. If a signal changes often, you can use this parameter to collect data at a slower rate.

Valid range: 0-4294967295

(Optional) `spoolingMode`

If `spoolingMode` is configured to `T0_DISK`, the Edge Agent software temporarily stores data locally when a vehicle isn't connected to the cloud. After the connection is reestablished, the data stored locally is automatically transferred to the cloud.

Default value: `OFF`

(Optional) `startTime`

An approved campaign is activated at the start time.

Default value: `0`

The status of a campaign can be one of the following values.

- `CREATING` – AWS IoT FleetWise is processing your request to create the campaign.
- `WAITING_FOR_APPROVAL` – After a campaign is created, it enters the `WAITING_FOR_APPROVAL` state. To approve the campaign, use the `UpdateCampaign` API operation. After the campaign is approved, AWS IoT FleetWise automatically deploys the campaign to the target vehicle or fleet. For more information, see [Update a campaign \(AWS CLI\)](#).
- `RUNNING` – The campaign is active.
- `SUSPENDED` – The campaign is suspended. To resume the campaign, use the `UpdateCampaign` API operation.

AWS IoT FleetWise provides the following API operations that you can use to create and manage campaigns.

- [CreateCampaign](#) – Creates a new campaign.
- [UpdateCampaign](#) – Updates an existing campaign. After a campaign is created, you must use this API operation to approve the campaign.
- [DeleteCampaign](#) – Deletes an existing campaign.
- [ListCampaigns](#) – Retrieves a paginated list of summaries for all campaigns.
- [GetCampaign](#) – Retrieves information about a campaign.

Tutorials

- [Create a campaign](#)

- [Update a campaign \(AWS CLI\)](#)
- [Delete a campaign](#)
- [Get campaign information \(AWS CLI\)](#)

Create a campaign

You can use the AWS IoT FleetWise console or API to create campaigns to collect vehicle data.

Important

For your campaign to work, you must have the following:

- The Edge Agent software is running in your vehicle. For more information about how to develop, install, and work with the Edge Agent software, do the following:
 1. Navigate to the [AWS IoT FleetWise console](#).
 2. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.
- You've set up AWS IoT Core to provision your vehicle. For more information, see [Provision vehicles](#).

Topics

- [Create a campaign \(console\)](#)
- [Create a campaign \(AWS CLI\)](#)
- [Logical expressions for campaigns](#)

Create a campaign (console)

You can use the AWS IoT FleetWise console to create a campaign to select, collect, and transfer vehicle data to the cloud.

To create a campaign

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Campaigns**.

3. On the **Campaigns** page, choose **Create campaign**, and then complete the steps in the following topics.

Topics

- [Step 1: Configure campaign](#)
- [Step 2: Define storage destination](#)
- [Step 3: Add vehicles](#)
- [Step 4: Review and create](#)
- [Step 5: Deploy a campaign](#)

Important

- You must have a signal catalog and a vehicle before you create a campaign. For more information, see [Create and manage signal catalogs](#) and [Create, provision, and manage vehicles](#).
- After a campaign is created, you must approve the campaign. For more information, see [Step 5: Deploy a campaign](#).

Step 1: Configure campaign

In **General information**, do the following:

1. Enter a name for the campaign.
2. (Optional) Enter a description.

Configure the campaign's data collection scheme. A data collection scheme gives the Edge Agent software instructions on what data to collect or when to collect it. In the AWS IoT FleetWise console, you can configure a data collection scheme in the following ways:


- Manually define the data collection scheme.
- Upload a file to automatically define the data collection scheme.

In **Configuration option**, choose one of the following:

- To manually specify the type of data collection scheme and define options to customize the scheme, choose **Define data collection scheme**.

Manually specify the type of data collection scheme and define options to customize the scheme.

1. In the **Data collection scheme details** section, choose the type of data collection scheme you want this campaign to use. To use a logical expression to recognize what vehicle data to collect, choose **Condition-based**. To use a specific time period to decide how often to collect vehicle data, choose **Time-based**.
2. Define the duration of time the campaign collects data.

 **Note**

By default, an approved campaign is activated immediately and doesn't have a set end time. To avoid extra charges, you must specify a time range.

3. If you specified a condition-based data collection scheme, you must define a logical expression to recognize what data to collect. AWS IoT FleetWise uses a logical expression to recognize what data to collect for a condition-based scheme. The expression must specify a signal's fully qualified name as a variable, a comparison operator, and a comparison value.

For example, if you specify the `$variable.`myVehicle.InVehicleTemperature` > 50.0` expression, AWS IoT FleetWise collects temperature values that are greater than 50.0. For instructions about how to write expressions, see [Logical expressions for campaigns](#).

Enter the logical expression used to recognize what data to collect.

4. (Optional) You can specify the language version of the conditional expression. The default value is 1.
5. (Optional) You can specify the minimum trigger interval, which is the smallest duration of time between two data collection events. For example, if a signal changes often, you might want to collect data at a slower rate.
6. Specify the **Trigger mode** condition for the Edge Agent software to collect data. By default, the Edge Agent for AWS IoT FleetWise software **Always** collects data whenever the condition is met. Or, it can collect data only when the condition is met for the first time, **On first trigger**.

7. If you specified a time-based data collection scheme, you must specify a time **Period**, in milliseconds, from 10,000 - 60,000 milliseconds. The Edge Agent software uses the time period to decide how often to collect data.
8. (Optional) You can edit the scheme's **Advanced scheme options**.
 - a. To save wireless bandwidth and reduce network traffic by compressing data, choose **Snappy**.
 - b. (Optional) To define how long, in milliseconds, to continue collecting data after a data collection event, you can specify the **Post trigger collection duration**.
 - c. (Optional) To indicate the priority level of the campaign, you can specify the campaign **Priority**. Campaigns with a smaller number for priority are deployed first and are considered to have a higher priority.
 - d. The Edge Agent software can temporarily store data locally when a vehicle isn't connected to the cloud. After the connection is reestablished, the data stored locally is automatically transferred to the cloud. Specify if you want the Edge Agent to **Store data locally** during a lost connection.
 - e. (Optional) To provide additional information for a signal, add up to five attributes as **Extra data dimensions**.
- To upload a file to define the data collection scheme, select **Upload a .json file from your local device**. AWS IoT FleetWise automatically defines which options that you can define in the file. You can review and update the selected options.

Upload a .json file with details about the data collection scheme.

1. To import information about the data collection scheme, choose **Choose files**. For more information about the required file format, see the [CreateCampaign](#) API documentation.

 **Note**

AWS IoT FleetWise currently supports the .json file format extension.

2. AWS IoT FleetWise automatically defines the data collection scheme based on the information in your file. Review the options that AWS IoT FleetWise selected for you. You can update the options, if needed.

Specify signals

You can specify the signals to collect data from when the data collection scheme is invoked.

Important

Signals used in the expression for the condition-based collection scheme must be specified in this field.

To specify the signals to collect data from

1. Search for the fully qualified **Name** of the signal.

Note

The fully qualified name of the signal is the path to the signal plus the signal's name. Use a dot(.) to refer to a child signal.

For example,

`Vehicle.Chassis.SteeringWheel.HandsOff.HandsOffSteeringState` is the fully qualified name for the `HandsOffSteeringState` actuator.

`Vehicle.Chassis.SteeringWheel.HandsOff.` is the path to this actuator.

2. (Optional) For **Max sample count**, enter the maximum number of data samples that the Edge Agent software collects and transfers to the cloud when the data collection scheme is invoked.
3. (Optional) For **Min sampling interval**, enter the minimum duration of time between two data sample collection events, in milliseconds. If a signal changes often, you can use this parameter to collect data at a slower rate.
4. To add another signal, choose **Add more signals**. You can add up to 999 signals.
5. Choose **Next**.

Step 2: Define storage destination

Note

You can only transfer vehicle data to Amazon S3 if the campaign contains vision system data signals.

Vision system data is in preview release and is subject to change.

Choose the destination where you want to store data collected by the campaign. You can transfer vehicle data to Amazon S3 or Amazon Timestream.

In **Destination settings**, do the following:

- Choose S3 or Timestream from the dropdown list.

To store vehicle data in an S3 bucket, choose **Amazon S3**. S3 is an object storage service that stores data as objects within buckets. For more information, see [Creating, configuring, and working with Amazon S3 buckets](#) in the *Amazon Simple Storage Service User Guide*.

S3 optimizes the cost of data storage and provides additional mechanisms to use vehicle data, such as data lakes, centralized data storage, data processing pipelines, and analytics. You can use S3 to store data for batch processing and analysis. For example, you can create reports of hard-braking events for your machine learning (ML) model. Incoming vehicle data is buffered for 10 minutes before delivery.

Amazon S3

Important

You can only transfer data to S3 if AWS IoT FleetWise has permissions to write into the S3 bucket. For more information about granting access, see [Controlling access with AWS IoT FleetWise](#).

In **S3 destination settings**, do the following:

1. For **S3 bucket**, choose a bucket that AWS IoT FleetWise has permissions to.
2. (Optional) Enter a custom prefix that you can use to organize data stored in the S3 bucket.
3. Choose the output format, which is the format files that are saved as in the S3 bucket.
4. Choose if you want to compress data stored in the S3 bucket as a .gzip file. We recommend compressing data because it minimizes storage costs.
5. The options you select in **S3 destination settings** change the **Example S3 object URI**. This is an example of what files are saved as in S3.

To store vehicle data in a Timestream table, choose **Amazon Timestream**. You can use Timestream to query vehicle data so that you can identify trends and patterns. For example, you can use Timestream to create an alarm for vehicle fuel level. Incoming vehicle data is transferred to Timestream in near real time. For more information, see [What is Amazon Timestream?](#) in the *Amazon Timestream Developer Guide*.

Amazon Timestream

Important

You can only transfer data to a table if AWS IoT FleetWise has permissions to write data into Timestream. For more information about granting access, see [Controlling access with AWS IoT FleetWise](#).

In **Timestream table settings**, do the following:

1. For **Timestream database name**, choose the name of your Timestream database from the dropdown list.
2. For **Timestream table name**, choose the name of your Timestream table from the dropdown list.

In **Service access for Timestream**, do the following:

- Choose an IAM role from the dropdown list.
- Choose **Next**.

Step 3: Add vehicles

To choose which vehicles to deploy your campaign to, select them in the vehicles list. Filter vehicles by searching for the attributes and their values that you added when creating the vehicles, or by vehicle name.

In **Filter vehicles**, do the following:

1. In the search box, find the attribute or vehicle name and choose it from the list.

Note

Each attribute can be used only once.

2. Enter the value of the attribute or the vehicle name that you want to deploy the campaign to. For example, if the fully qualified name of the attribute is `fuelType`, enter `gasoline` as its value.
3. To search for another vehicle attribute, repeat the preceding steps. You can search for up to five vehicle attributes and an unlimited number of vehicle names.
4. Vehicles that match your search are listed under **Vehicle name**. Choose the vehicles that you want the campaign to deploy to.

Note

Up to 100 vehicles are displayed in search results. Choose **Select all** to add all vehicles to the campaign.

5. Choose **Next**.

Step 4: Review and create

Verify the configurations for the campaign, and then choose **Create campaign**.

Note

After a campaign is created, you or your team must deploy the campaign to vehicles.

Step 5: Deploy a campaign

After you create a campaign, you or your team must deploy the campaign to vehicles.

To deploy a campaign

1. On the **Campaign summary** page, choose **Deploy**.
2. Review and confirm that you want to start the deployment and begin collecting data from vehicles connected to the campaign.

3. Choose **Deploy**.

If you want to pause collecting data from vehicles connected to the campaign, on the **Campaign summary** page, choose **Suspend**. To resume collecting data from vehicles connected to the campaign, choose **Resume**.

Create a campaign (AWS CLI)

You can use the [CreateCampaign](#) API operation to create a campaign. The following example uses the AWS CLI.

When you create a campaign, data collected from vehicles can be stored in either Amazon S3 (S3) or Amazon Timestream. Choose Timestream for a fast, scalable, and server-less time series database, such as to store data that requires near real time processing. Choose S3 for a object storage with industry-leading scalability, data availability, security, and performance.

Important

You can only transfer vehicle data if AWS IoT FleetWise has permissions to write data into S3 or Timestream. For more information about granting access, see [Controlling access with AWS IoT FleetWise](#).

Create campaign

Important

- You must have a signal catalog and a vehicle or fleet before you create a campaign. For more information, see [Create and manage signal catalogs](#), [Create, provision, and manage vehicles](#), and [Create and manage fleets](#).
- After a campaign is created, you must use the UpdateCampaign API operation to approve the campaign. For more information, see [Update a campaign \(AWS CLI\)](#)

To create a campaign, run the following command.

Replace *file-name* with the name of the JSON file that contains the campaign configuration.

```
aws iotfleetwise create-campaign --cli-input-json file://file-name.json
```

- Replace *campaign-name* with the name of the campaign that you're creating.
- Replace *signal-catalog-arn* with the Amazon Resource Name (ARN) of the signal catalog.
- Replace *target-arn* with the ARN of a fleet or vehicle that you created.
- Replace *bucket-arn* with the ARN of the S3 bucket.

```
{
  "name": "campaign-name",
  "targetArn": "target-arn",
  "signalCatalogArn": "signal-catalog-arn",
  "collectionScheme": {
    "conditionBasedCollectionScheme": {
      "conditionLanguageVersion": 1,
      "expression": "$variable.`Vehicle.DemoBrakePedalPressure` > 7000",
      "minimumTriggerIntervalMs": 1000,
      "triggerMode": "ALWAYS"
    }
  },
  "compression": "SNAPPY",
  "diagnosticsMode": "OFF",
  "postTriggerCollectionDuration": 1000,
  "priority": 0,
  "signalsToCollect": [
    {
      "maxSampleCount": 100,
      "minimumSamplingIntervalMs": 0,
      "name": "Vehicle.DemoEngineTorque"
    },
    {
      "maxSampleCount": 100,
      "minimumSamplingIntervalMs": 0,
      "name": "Vehicle.DemoBrakePedalPressure"
    }
  ],
  "spoolingMode": "TO_DISK",
  "dataDestinationConfigs": [
    {
      "s3Config": {
        "bucketArn": "bucket-arn",

```

```

        "dataFormat": "PARQUET",
        "prefix": "campaign-name",
        "storageCompressionFormat": "GZIP"
    }
}
]
}

```

- Replace *campaign-name* with the name of the campaign that you're creating.
- Replace *signal-catalog-arn* with the Amazon Resource Name (ARN) of the signal catalog.
- Replace *target-arn* with the ARN of a fleet or vehicle that you created.
- Replace *role-arn* with the ARN of the task execution role that grants AWS IoT FleetWise permission to deliver data to the Timestream table.
- Replace *table-arn* with the ARN of the Timestream table.

```

{
  "name": "campaign-name",
  "targetArn": "target-arn",
  "signalCatalogArn": "signal-catalog-arn",
  "collectionScheme": {
    "conditionBasedCollectionScheme": {
      "conditionLanguageVersion": 1,
      "expression": "$variable.`Vehicle.DemoBrakePedalPressure` > 7000",
      "minimumTriggerIntervalMs": 1000,
      "triggerMode": "ALWAYS"
    }
  },
  "compression": "SNAPPY",
  "diagnosticsMode": "OFF",
  "postTriggerCollectionDuration": 1000,
  "priority": 0,
  "signalsToCollect": [
    {
      "maxSampleCount": 100,
      "minimumSamplingIntervalMs": 0,
      "name": "Vehicle.DemoEngineTorque"
    },
    {
      "maxSampleCount": 100,
      "minimumSamplingIntervalMs": 0,

```

```

    "name": "Vehicle.DemoBrakePedalPressure"
  }
],
"spoolingMode": "TO_DISK",
"dataDestinationConfigs": [
  {
    "timestreamConfig": {
      "executionRoleArn": "role-arn",
      "timestreamTableArn": "table-arn"
    }
  }
]
}

```

Logical expressions for campaigns

AWS IoT FleetWise uses a logical expression to recognize what data to collect as part of a campaign. For more information about expressions, see [Expressions](#) in the *AWS IoT Events Developer Guide*.

The expression variable should be constructed to comply with the rules for the type of data being collected. For telemetry system data, the expression variable should be the signal's fully qualified name. For vision system data, the expression combines the signal's fully qualified name with the path leading from the signal's data type to one of its properties.

For example, if the signal catalog contains the following nodes:

```

{
  myVehicle.ADAS.Camera:
    type: sensor
    datatype: Vehicle.ADAS.CameraStruct
    description: "A camera sensor"

  myVehicle.ADAS.CameraStruct:
    type: struct
    description: "An obstacle detection camera output struct"
}

```

If the nodes follow the ROS 2 definition:

```

{

```

```

Vehicle.ADAS.CameraStruct.msg:
boolean obstaclesExists
uint8[] image
Obstacle[30] obstacles
}
{
Vehicle.ADAS.Obstacle.msg:
float32: probability
uint8 o_type
float32: distance
}

```

The following are all possible event expression variables:

```

{
...
$variable.`myVehicle.ADAS.Camera.obstaclesExists`
$variable.`myVehicle.ADAS.Camera.Obstacle[0].probability`
$variable.`myVehicle.ADAS.Camera.Obstacle[1].probability`
...
$variable.`myVehicle.ADAS.Camera.Obstacle[29].probability`
$variable.`myVehicle.ADAS.Camera.Obstacle[0].o_type`
$variable.`myVehicle.ADAS.Camera.Obstacle[1].o_type`
...
$variable.`myVehicle.ADAS.Camera.Obstacle[29].o_type`
$variable.`myVehicle.ADAS.Camera.Obstacle[0].distance`
$variable.`myVehicle.ADAS.Camera.Obstacle[1].distance`
...
$variable.`myVehicle.ADAS.Camera.Obstacle[29].distance`
}

```

Update a campaign (AWS CLI)

You can use the [UpdateCampaign](#) API operation to update an existing campaign. The following command uses AWS CLI.

- Replace *campaign-name* with the name of the campaign that you're updating.
- Replace *action* with one of the following:
 - APPROVE – Approves the campaign to allow AWS IoT FleetWise to deploy it to a vehicle or fleet.

- SUSPEND – Suspends the campaign. The campaign is deleted from vehicles and all vehicles in the suspended campaign will stop sending data.
- RESUME – Reactivates the SUSPEND campaign. The campaign is redeployed to all vehicles and the vehicles will resume sending data.
- UPDATE – Updates the campaign by defining attributes and associating them with a signal.

```
aws iotfleetwise update-campaign \  
    --name campaign-name \  
    --action action
```

Delete a campaign

You can use the AWS IoT FleetWise console or API to delete campaigns.

Delete a campaign (console)

To delete a campaign, use the AWS IoT FleetWise console.

To delete a campaign

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the navigation pane, choose **Campaigns**.
3. On the **Campaigns** page, choose the target campaign.
4. Choose **Delete**.
5. In **Delete campaign-name?**, enter the name of the campaign to delete, and then choose **Confirm**.

Delete a campaign (AWS CLI)

You can use the [DeleteCampaign](#) API operation to delete a campaign. The following example uses AWS CLI.

To delete a campaign, run the following command.

Replace *campaign-name* with the name of the vehicle that you're deleting.

```
aws iotfleetwise delete-campaign --name campaign-name
```

Get campaign information (AWS CLI)

You can use the [ListCampaigns](#) API operation to verify if a campaign has been deleted. The following example uses the AWS CLI.

To retrieve a paginated list of summaries for all campaigns, run the following command.

```
aws iotfleetwise list-campaigns
```

You can use the [GetCampaign](#) API operation to retrieve vehicle information. The following example uses the AWS CLI.

To retrieve the metadata of a campaign, run the following command.

Replace *campaign-name* with the name of the campaign to you want to retrieve.

```
aws iotfleetwise get-campaign --name campaign-name
```

Note

This operation is [eventually consistent](#). In other words, changes to the campaign might not be reflected immediately.

Processing and visualizing vehicle data

The Edge Agent for AWS IoT FleetWise software transfers selected vehicle data to Amazon Timestream or Amazon Simple Storage Service (Amazon S3). After your data arrives in the data destination, you can use other AWS services to visualize and share it.

Processing vehicle data in Timestream

Timestream is a fully managed time series database that can store and analyze trillions of time series data points per day. Your data is stored in a customer managed Timestream table. You can use Timestream to query vehicle data so that you can gain insights about your vehicles. For more information, see [What is Amazon Timestream?](#)

The default schema of data that is transferred to Timestream contains the following fields.

Field name	Data type	Description
eventId	varchar	The ID of the data collection event.
vehicleName	varchar	The ID of the vehicle from which the data was collected.
name	varchar	The name of the campaign that the Edge Agent software uses to collect data.
time	timestamp	The timestamp of the data point.
measure_name	varchar	The name of the signal.
measure_value::bigint	bigint	Signal values of type Integer.

Field name	Data type	Description
measure_value::double	double	Signal values of type Double.
measure_value::boolean	boolean	Signal values of type Boolean.

Visualizing vehicle data stored in Timestream

After your vehicle data is transferred to Timestream, you can use the following AWS services to visualize, monitor, analyze, and share your data.

- Visualize and monitor data in dashboards by using [Grafana or Amazon Managed Grafana](#). You can visualize data from multiple AWS sources (such as Amazon CloudWatch and Timestream) and other data sources with a single Grafana dashboard.
- Analyze and visualize data in dashboards by using [Amazon QuickSight](#).

Processing vehicle data in S3

Amazon S3 is an object storage service that stores and protects any amount of data. You can use S3 for a variety of use cases, such as data lakes, backup and restore, archive, enterprise applications, AWS IoT devices, and big data analytics. Your data is stored in S3 as objects in buckets. For more information, see [What is Amazon S3?](#)

The default schema of data that is transferred to Amazon S3 contains the following fields.

Field name	Data type	Description
eventId	varchar	The ID of the data collection event.
vehicleName	varchar	The ID of the vehicle from which the data was collected.

Field name	Data type	Description
name	varchar	The name of the campaign that the Edge Agent software uses to collect data.
time	timestamp	The timestamp of the data point.
measure_name	varchar	The name of the signal.
measure_value_BIGINT	bigint	Signal values of type Integer.
measure_value_DOUBLE	double	Signal values of type Double.
measure_value_BOOLEAN	boolean	Signal values of type Boolean.
measure_value_STRUCT	struct	Signal values of type Struct.

S3 object format

AWS IoT FleetWise transfers vehicle data to S3 where it's saved as an object. You can use the object URI that uniquely identifies the data to find data from the campaign. The S3 object URI format depends on if the collected data is unstructured or processed data.

Unstructured data

Unstructured data is stored in S3 in a not pre-defined manner. It can be in various formats, such as images or videos.

Vehicle messages passed to AWS IoT FleetWise with signal data from Amazon Ion files are decoded and transferred to S3 as objects. The S3 objects represent each signal and are binary encoded.

The unstructured data S3 object URI uses the following format:

```
s3://bucket-name/prefix/unstructured-data/random-ID-yyyy-MM-dd-HH-mm-ss-SSS-vehicleName-signalName-fieldName
```

Processed data

Processed data is stored in S3 and undergoes processing steps that validate, enrich, and transform messages. Object lists and velocity are examples of processed data.

Data transferred to S3 are stored as objects that represent records that were buffered for a period of about 10 minutes. By default, AWS IoT FleetWise adds a UTC time prefix in the format year=YYYY/month=MM/date=DD/hour=HH before writing objects to S3. This prefix creates a logical hierarchy in the bucket where each forward slash (/) creates a level in the hierarchy. The processed data also contains the S3 object URI to unstructured data.

The processed data S3 object URI uses the following format:

```
s3://bucket-name/prefix/processed-data/year=YYYY/month=MM/day=DD/hour=HH/part-0000-random-ID.gz.parquet
```

Raw data

Raw data, also known as primary data, are data collected from Amazon Ion files. You can use raw data to troubleshoot any issues or to root cause errors.

The raw data S3 object URI uses the following format:

```
s3://bucket-name/prefix/raw-data/vehicle-name/eventID-timestamp.10n
```

Analyzing vehicle data stored in S3

After your vehicle data is transferred to S3, you can use the following AWS services to monitor, analyze, and share your data.

Extract and analyze data using Amazon SageMaker for downstream labeling and machine learning (ML) workflows.

For more information, see the following topics in the *Amazon SageMaker Developer Guide*:

- [Process data](#)
- [Train machine learning models](#)
- [Label Images](#)

Catalog your data using AWS Glue crawler and analyze it in Amazon Athena. By default, objects written to S3 have Apache Hive style time partitions, with data paths that contain key-value pairs connected by equal signs.

For more information, see the following topics in the *Amazon Athena User Guide*:

- [Partitioning data in Athena](#)
- [Using AWS Glue to connect to data sources in Amazon S3](#)
- [Best practices when using Athena with AWS Glue](#)

Visualize data using Amazon QuickSight by either reading your Athena table or S3 bucket directly.

 **Tip**

If you're reading from S3 directly, confirm that your vehicle data is in JSON format because Amazon QuickSight doesn't support Apache Parquet format.

For more information, see the following topics in the *Amazon QuickSight User Guide*:

- [Supported data sources](#)
- [Creating a data source](#)

AWS CLI and AWS SDKs

This section provides information about making AWS IoT FleetWise API requests. For more information about AWS IoT FleetWise [operations and data types](#), see the *AWS IoT FleetWise API Reference*.

To use AWS IoT FleetWise with a variety of programming languages, use the [AWS SDKs](#), which contain the following automatic functionality:

- Cryptographically signing your service requests
- Retrying requests
- Handling error responses

For command line access, use AWS IoT FleetWise with the [AWS CLI](#). You can control AWS IoT FleetWise, and your other services, from the command line, and automate them through scripts.

Troubleshooting AWS IoT FleetWise

Use the troubleshooting information and solutions in this section to help resolve issues with AWS IoT FleetWise.

The following information might help you troubleshoot common issues with AWS IoT FleetWise.

Topics

- [Decoder manifest issues](#)
- [Edge Agent for AWS IoT FleetWise software issues](#)

Decoder manifest issues

Troubleshoot decoder manifest issues.

Diagnosing decoder manifest API calls

Error	Troubleshooting guidelines
<code>UpdateOperationFailure.ConflictingDecoderUpdate</code>	The same decoder manifest has multiple update requests. Wait and try again.
<code>UpdateOperationFailure.InternalFailure</code>	<code>InternalFailure</code> is launched as an encapsulated exception. The problem itself depends on the exception encapsulated.
<code>UpdateOperationFailure.ActiveDecoderUpdate</code>	The decoder manifest is in an <code>Active</code> state and can't be updated. Change the decoder manifest state to <code>DRAFT</code> , and then try again.
<code>UpdateOperationFailure.ConflictingModelUpdate</code>	AWS IoT FleetWise is trying to validate against a vehicle model (model manifest) that's being modified by someone else. Wait and try again.
<code>UpdateOperationFailure.ModelManifestValidationResponse : FailureReason.MODEL_DATA_ENTRIES_NOT_FOUND</code>	The vehicle model doesn't have any signals associated with it. Add signals to the vehicle model and verify that the signals can be found in the associated signal catalog.

Error	Troubleshooting guidelines
<pre>UpdateOperationFailure.Mode lManifestValidationResponse : FailureReason.MODEL_NOT_ACTIVE</pre>	<p>Update the vehicle model so that it's in ACTIVE state, and then try again.</p>
<pre>UpdateOperationFailure.Mode lManifestValidationResponse : FailureReason.MODEL_NOT_FOUND</pre>	<p>AWS IoT FleetWise can't find the vehicle model associated with the decoder manifest. Verify the Amazon Resource Name (ARN) of the vehicle model and try again.</p>
<pre>UpdateOperationFailure.Mode lManifestValidationResponse (FailureReason.MODEL_DATA_E NTRIES_READ_FAILURE</pre>	<p>The validation of the vehicle model failed because signal names from the vehicle model weren't found in the signal catalog. Verify that the signals in the vehicle model are all included in the associated signal catalog.</p>
<pre>UpdateOperationFailure.Vali dationFailure</pre>	<p>Signals or network interfaces that aren't valid were found in the request to update the decoder manifest. Verify that all signals and network interfaces returned by the exception exist, that all signals used are associated with an available interface, and that you won't remove an interface that has signals associated with it.</p>
<pre>UpdateOperationFailure.KmsK eyAccessDenied</pre>	<p>There's a permission issue on the AWS Key Management Service (AWS KMS) key used for the operation. Verify that you're using a role that has access to the key and try again.</p>
<pre>UpdateOperationFailure.Deco derDoesNotExist</pre>	<p>The decoder manifest doesn't exist. Verify the decoder manifest name and try again.</p>

Vision system data error messages with the `SIGNAL_DECODER_INCOMPATIBLE_WITH_SIGNAL_CATALOG` reason will include a hint in

the response that provides information about why the request failed. You can use the hint to determine which troubleshooting guidelines to follow.

 **Note**

Vision system data is in preview release and is subject to change.

Diagnosing decoder manifest vision system data validation

Error	Troubleshooting guidelines
<code>InvalidSignalDecoder.withReason(SignalDecoderFailureReason.NO_SIGNAL_IN_CATALOG_FOR_DECODER_SIGNAL)</code>	AWS IoT FleetWise didn't find the root signal structure used in the signal decoder using the signal catalog. Verify that the root signal of the structure is properly defined in the signal catalog.
<code>InvalidSignalDecoder.withReason(SignalDecoderFailureReason.SIGNAL_DECODER_TYPE_INCOMPATIBLE_WITH_MESSAGE_SIGNAL_TYPE)</code>	A primitive message in the signal catalog wasn't defined with the same data type in the decoder manifest update request. Verify that the primitive messages defined in the request match their corresponding signal catalog definition.
<code>InvalidSignalDecoder.withReason(SignalDecoderFailureReason.STRUCT_SIZE_MISMATCH)</code>	The number of properties defined in a struct in the signal catalog don't match the number of properties you're trying to decode in the decoder manifest. Verify that you have the correct number of signals to decode by comparing it with the signals defined in the signal catalog.
<code>InvalidSignalDecoder.withReason(SignalDecoderFailureReason.SIGNAL_DECODER_INCOMPATIBLE_WITH_SIGNAL_CATALOG)</code>	AWS IoT FleetWise found a signal defined as a STRUCT in the signal catalog without a <code>structuredMessageDefinition</code> defined in the decoder manifest request. Make sure that each struct is defined as a <code>structuredMessageD</code>

Error	Troubleshooting guidelines
<code>InvalidSignalDecoder.withReason(SignalDecoderFailureReason.SIGNAL_DECODER_INCOMPATIBLE_WITH_SIGNAL_CATALOG)</code>	<p>The root signal of the structure used in the decoder manifest is not properly defined as a structure in the signal catalog. The root signal structure used in the decoder manifest must have its field <code>structFullyQualifiedName</code> defined. It also needs a <code>STRUCT</code> node with that <code>fullyQualifiedName</code>.</p>
<code>InvalidSignalDecoder.withReason(SignalDecoderFailureReason.SIGNAL_DECODER_INCOMPATIBLE_WITH_SIGNAL_CATALOG)</code>	<p>One of the leaf messages used in the decoder manifest request is not defined as a primitive message. Verify that all leaf objects in the request are defined as primitive messages.</p>
<code>InvalidSignalDecoder.withReason(SignalDecoderFailureReason.SIGNAL_DECODER_INCOMPATIBLE_WITH_SIGNAL_CATALOG)</code>	<p>An array object in the signal catalog wasn't defined as a <code>structuredMessageListDefinition</code> in the decoder manifest update request. Verify that all array properties are defined as <code>structuredMessageListDefinition</code> in the decoder manifest update request.</p>

Edge Agent for AWS IoT FleetWise software issues

Troubleshoot Edge Agent software issues.

Issues

- [Issue: The Edge Agent software doesn't start.](#)
- [Issue: \[ERROR\] \[IoTFleetWiseEngine::connect\]: \[Failed to init persistency library\]](#)
- [Issue: The Edge Agent software doesn't collect on-board diagnostics \(OBD\) II PIDs and diagnostic trouble codes \(DTCs\).](#)
- [Issue: The Edge Agent for AWS IoT FleetWise software doesn't collect data from the network or isn't able to apply data inspection rules.](#)

- [Issue: \[ERROR\] \[AwsIotConnectivityModule::connect\]: \[Connection failed with error\] or \[WARN\] \[AwsIotChannel::send\]: \[No alive MQTT Connection.\]](#)

Issue: The Edge Agent software doesn't start.

You might see the following errors when the Edge Agent software doesn't start.

- ```
Error from reader: * Line 1, Column 1
Syntax error: value, object or array expected.
```

**Solution:** Make sure the Edge Agent for AWS IoT FleetWise software configuration file is using valid JSON format. For example, make sure that commas are used correctly. For more information about the configuration file, do the following to download the *Edge Agent for AWS IoT FleetWise software Developer Guide*.

1. Navigate to the [AWS IoT FleetWise console](#).
2. On the service home page, in the **Get started with AWS IoT FleetWise** section, choose **Explore Edge Agent**.

- ```
[ERROR] [SocketCANBusChannel::connect]: [ SocketCan with name xxx is not accessible]
[ERROR] [IoTFleetWiseEngine::connect]: [ Failed to Bind Consumers to Producers ]
```

Solution: You might see this error when the Edge Agent software fails to establish socket communication with the network interfaces defined in the configuration file.

To check that every network interface defined in the configuration is available, run the following command.

```
ip link show
```

To bring a network interface online, run the following command. Replace *network-interface-id* with the ID of the network interface.

```
sudo ip link set network-interface-id up
```

- ```
[ERROR] [AwsIotConnectivityModule::connect]: [Connection failed with error]
[WARN] [AwsIotChannel::send]: [No alive MQTT Connection.]
or
```

```
[WARN] [AwsIotChannel::send]: [aws-c-common: AWS_ERROR_FILE_INVALID_PATH]
```

**Solution:** You might see this error when the Edge Agent software fails to establish an MQTT connection to AWS IoT Core. Check that the following are configured correctly and restart the Edge Agent software.

- `mqttConnection::endpointUrl` – AWS account's IoT device endpoint.
- `mqttConnection::clientId` – The ID of the vehicle in which the Edge Agent software is running.
- `mqttConnection::certificateFilename` – The path to the vehicle certificate file.
- `mqttConnection::privateKeyFilename` – The path to the vehicle private key file.
- You have used AWS IoT Core to provision the vehicle. For more information, see [Provision vehicles](#).

For more troubleshooting information, see [AWS IoT Device SDK for C++ Frequently Asked Questions](#).

## Issue: [ERROR] [IoTFleetWiseEngine::connect]: [Failed to init persistency library]

**Solution:** You might see this error when the Edge Agent software fails to locate the persistence storage. Check that the following is configured correctly and restart the Edge Agent software.

`persistency::persistencyPath` – A local path used to persist collection schemes, decoder manifests, and data snapshots.

## Issue: The Edge Agent software doesn't collect on-board diagnostics (OBD) II PIDs and diagnostic trouble codes (DTCs).

**Solution:** You might see this error if `obdInterface::pidRequestIntervalSeconds` or `obdInterface::dtcRequestIntervalSeconds` is configured to 0.

If the Edge Agent software is running in an automatic transmission vehicle, make sure `obdInterface::hasTransmissionEcu` is configured to `true`.

If your vehicle supports extended Controller Area Network (CAN bus) arbitration IDs, make sure `obdInterface::useExtendedIds` is configured to `true`.

## Issue: The Edge Agent for AWS IoT FleetWise software doesn't collect data from the network or isn't able to apply data inspection rules.

**Solution:** You might see this error when the default quotas are breached.

| Resource                                             | Quota                                              | Adjustable | Note                                                                                                                                                                                                |
|------------------------------------------------------|----------------------------------------------------|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Value of the signal ID                               | The signal ID must be less than or equal to 50,000 | Yes        | The Edge Agent software won't collect data from signals that have an ID greater than 50,000. We recommend that you check how many signals the signal catalog contains before you change this quota. |
| Number of active data collection schemes per vehicle | 256                                                | Yes        | We recommend that you check how many campaigns that you've created in the cloud and how many schemes each campaign contains before you change this quota.                                           |
| Size of the signal history buffer                    | 20 MB                                              | Yes        | If the quota is breached, the Edge Agent software stops collecting new data.                                                                                                                        |

## **Issue: [ERROR] [AwsIotConnectivityModule::connect]: [Connection failed with error] or [WARN] [AwsIotChannel::send]: [No alive MQTT Connection.]**

**Solution:** You might see this error when the Edge Agent software isn't connected to the cloud. By default, the Edge Agent software sends a ping request to AWS IoT Core every minute and waits for three minutes. If there's no response, the Edge Agent software automatically reestablishes the connection to the cloud.



# Security in AWS IoT FleetWise

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to AWS IoT FleetWise, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using AWS IoT FleetWise. It shows you how to configure AWS IoT FleetWise to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS IoT FleetWise resources.

## Contents

- [Data protection in AWS IoT FleetWise](#)
- [Controlling access with AWS IoT FleetWise](#)
- [Identity and Access Management for AWS IoT FleetWise](#)
- [Compliance Validation for AWS IoT FleetWise](#)
- [Resilience in AWS IoT FleetWise](#)
- [Infrastructure security in AWS IoT FleetWise](#)
- [Configuration and vulnerability analysis in AWS IoT FleetWise](#)
- [Security best practices for AWS IoT FleetWise](#)

## Data protection in AWS IoT FleetWise

The AWS [shared responsibility model](#) applies to data protection in AWS IoT FleetWise. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS IoT FleetWise or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

AWS IoT FleetWise is intended to be used with an Edge Agent that you develop and install on supported vehicle hardware in order to transmit vehicle data to the AWS Cloud. Extracting data from vehicles *might* be subject to data privacy regulations in certain jurisdictions. Before using

AWS IoT FleetWise and installing your Edge Agent, we strongly recommend that you assess your compliance obligations under applicable law. This includes any applicable legal requirements to provide legally adequate privacy notices and obtain any necessary consents for extracting vehicle data.

## Encryption at rest

The data collected from a vehicle is transmitted to the cloud through an AWS IoT Core message with the MQTT message protocol. AWS IoT FleetWise delivers the data to your Amazon Timestream database. In Timestream, your data is encrypted. All AWS services encrypt data at rest by default.

Encryption at rest integrates with AWS Key Management Service (AWS KMS) to manage the encryption key that's used to encrypt your data. You can choose to use a customer managed key to encrypt data collected by AWS IoT FleetWise. You can create, manage, and view your encryption key through AWS KMS. For more information, see [What is AWS Key Management Service?](#) in the *AWS Key Management Service Developer Guide*.

## Encryption in transit

All data exchanged with AWS IoT services is encrypted in transit by using Transport Layer Security (TLS). For more information, see [Transport security](#) in the *AWS IoT Developer Guide*.

Also, AWS IoT Core supports [authentication](#) and [authorization](#) to help securely control access to AWS IoT FleetWise resources. Vehicles can use X.509 certificates to get authenticated (signed in) to use AWS IoT FleetWise and use AWS IoT Core policies to get authorized (have permissions) to perform specified actions. For more information, see [the section called "Provision vehicles"](#).

## Data encryption

Data encryption refers to protecting data while in-transit (as it travels to and from AWS IoT FleetWise, and between gateways and servers), and at rest (while it's stored on local devices or in AWS services). You can protect data at rest using client-side encryption.

### Note

AWS IoT FleetWise edge processing exposes APIs that are hosted within AWS IoT FleetWise gateways and are accessible over the local network. These APIs are exposed over a TLS connection backed by a server-certificate owned by the AWS IoT FleetWise Edge connector. For client authentication, these APIs use an access-control password. The server-certificate

private-key and the access-control password are both stored on disk. AWS IoT FleetWise edge processing relies on file-system encryption for the security of these credentials at rest.

For more information about server-side encryption and client-side encryption, review the following topics.

## Contents

- [Encryption at rest](#)
- [Key management](#)

## Encryption at rest

AWS IoT FleetWise stores your data in the AWS Cloud and on gateways.

### Data at rest in the AWS Cloud

AWS IoT FleetWise stores data in other AWS services that encrypt data at rest by default. Encryption at rest integrates with [AWS Key Management Service \(AWS KMS\)](#) for managing the encryption key that is used to encrypt your asset property values and aggregate values in AWS IoT FleetWise. You can choose to use a customer managed key to encrypt asset property values and aggregate values in AWS IoT FleetWise. You can create, manage, and view your encryption key through AWS KMS.

You can choose an AWS owned key or a customer managed key to encrypt your data.

### How it works

Encryption at rest integrates with AWS KMS for managing the encryption key that is used to encrypt your data.

- **AWS owned key** – Default encryption key. AWS IoT FleetWise owns this key. You can't view, manage, or use this key in your AWS account. You also can't see operations on the key in AWS CloudTrail logs. You can use this key at no additional charge.
- **Customer managed key** – The key is stored in your account, which you create, own, and manage. You have full control over the KMS key. Additional AWS KMS charges apply.

## AWS owned keys

AWS owned keys aren't stored in your account. They're part of a collection of KMS keys that AWS owns and manages for use in multiple AWS accounts. AWS services can use AWS owned keys to protect your data.

You can't view, manage, or use AWS owned keys, or audit their use. However, you don't need to take any action or change any programs to protect keys that encrypt your data.

You won't be charged a fee if you use AWS owned keys, and they don't count against AWS KMS quotas for your account.

## Customer managed keys

Customer managed keys are KMS keys in your account that you create, own, and manage. You have full control over these KMS keys, such as the following:

- Establishing and maintaining their key policies, IAM policies, and grants
- Enabling and disabling them
- Rotating their cryptographic material
- Adding tags
- Creating aliases that refer to them
- Scheduling them for deletion

You can also use CloudTrail and Amazon CloudWatch Logs to track the requests that AWS IoT FleetWise sends to AWS KMS on your behalf.

If you're using customer managed keys, you must grant AWS IoT FleetWise access to the KMS key stored in your account. AWS IoT FleetWise uses envelope encryption and key hierarchy to encrypt data. Your AWS KMS encryption key is used to encrypt the root key of this key hierarchy. For more information, see [Envelope encryption](#) in the *AWS Key Management Service Developer Guide*.

The following example policy grants AWS IoT FleetWise permissions to a create customer managed key on your behalf.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
 {
 "Sid": "Stmt1603902045292",
 "Action": [
 "kms:GenerateDataKey*",
 "kms:Decrypt",
 "kms:DescribeKey",
 "kms:CreateGrant",
 "kms:RetireGrant",
 "kms:RevokeGrant"
],
 "Effect": "Allow",
 "Resource": "*"
 }
]
```

### Important

When you add the new sections to your KMS key policy, don't change any existing sections in the policy. AWS IoT FleetWise can't perform operations to your data if encryption is enabled for AWS IoT FleetWise and any of the following is true:

- The KMS key is disabled or deleted.
- The KMS key policy isn't correctly configured for the service.

## Using vision system data with encryption at rest

### Note

Vision system data is in preview release and is subject to change.

If you have customer managed encryption with AWS KMS keys enabled on your AWS IoT FleetWise account, and you want to use vision system data, reset your encryption settings to be compatible with complex data types. This enables AWS IoT FleetWise to establish additional permissions needed for vision system data.

**Note**

Your decoder manifest could be stuck in a validating status if you haven't reset your encryption settings for vision system data.

1. Use the [GetEncryptionConfiguration](#) API operation to check if AWS KMS encryption is enabled. No further action is needed if the encryption type is FLEETWISE\_DEFAULT\_ENCRYPTION.
2. If the encryption type is KMS\_BASED\_ENCRYPTION, use the [PutEncryptionConfiguration](#) API operation to reset the encryption type to FLEETWISE\_DEFAULT\_ENCRYPTION.

```
{
 aws iotfleetwise put-encryption-configuration --encryption-type
 FLEETWISE_DEFAULT_ENCRYPTION
}
```

3. Use the [PutEncryptionConfiguration](#) API operation to re-enable the encryption type to KMS\_BASED\_ENCRYPTION.

```
{
 aws iotfleetwise put-encryption-configuration \
 --encryption-type "KMS_BASED_ENCRYPTION"
 --kms-key-id kms_key_id
}
```

For more information about enabling encryption, see [Key management](#).

## Key management

### AWS IoT FleetWise cloud key management

By default, AWS IoT FleetWise uses AWS managed keys to protect your data in the AWS Cloud. You can update your settings to use a customer managed key to encrypt data in AWS IoT FleetWise. You can create, manage, and view your encryption key through AWS Key Management Service (AWS KMS).

AWS IoT FleetWise supports server-side encryption with customer managed keys stored in AWS KMS to encrypt data for the following resources.

| AWS IoT FleetWise resource     | Data type    | Fields that are encrypted at rest with customer managed keys                                                                        |
|--------------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------|
| Signal catalog                 |              | description                                                                                                                         |
|                                | Attribute    | description, allowedValues, defaultValue, min, max                                                                                  |
|                                | Actuator     | description, allowedValues, min, max                                                                                                |
|                                | Sensor       | description, allowedValues, min, max                                                                                                |
| Vehicle model (model manifest) |              | description                                                                                                                         |
| Decoder manifest               |              | description                                                                                                                         |
|                                | CanInterface | protocolName, protocolVersion                                                                                                       |
|                                | ObdInterface | requestMessageId, dtcReques<br>tIntervalSeconds, hasTransm<br>issionEcu, obdStandard, pidReques<br>tIntervalSeconds, useExtendedIds |
|                                | CanSignal    | factor, isBigEndian, isSigned, length,<br>messageId, offset, startBit                                                               |
|                                | ObdSignal    | byteLength, offset, pid, pidRespon<br>seLength, scaling, serviceMode,<br>startByte, bitMaskLength, bitRightS<br>hift                |
| Vehicle                        |              | attributes                                                                                                                          |
| Campaign                       |              | description                                                                                                                         |



| AWS IoT FleetWise resource | Data type                      | Fields that are encrypted at rest with customer managed keys                |
|----------------------------|--------------------------------|-----------------------------------------------------------------------------|
|                            | conditionBasedCollectionScheme | expression, conditionLanguageVersion, minimumTriggerIntervalMs, triggerMode |
|                            | TimeBasedCollectionScheme      | periodMs                                                                    |

### Note

Other data and resources are encrypted using the default encryption with keys managed by AWS IoT FleetWise. This key is created and stored in the AWS IoT FleetWise account.

For more information, see [What is AWS Key Management Service?](#) in the *AWS Key Management Service Developer Guide*.

## Enable encryption using KMS keys (console)

To use customer managed keys with AWS IoT FleetWise, you must update your AWS IoT FleetWise settings.


### To enable encryption using KMS keys (console)

1. Open the [AWS IoT FleetWise console](#).
2. Navigate to **Settings**.
3. In **Encryption**, choose **Edit** to open the **Edit encryption** page.
4. For **Encryption key type**, choose **Choose a different AWS KMS key**. This enables encryption with customer managed keys stored in AWS KMS.

### Note

You can only use customer managed key encryption for AWS IoT FleetWise resources. This includes the signal catalog, vehicle model (model manifest), decoder manifest, vehicle, fleet, and campaign.

5. Choose your KMS key with one of the following options:
  - **To use an existing KMS key** – Choose your KMS key alias from the list.
  - **To create a new KMS key** – Choose **Create an AWS KMS key**.

 **Note**

This opens the AWS KMS console. For more information about creating a KMS key, see [Creating keys](#) in the *AWS Key Management Service Developer Guide*.

6. Choose **Save** to update your settings.

### Enable encryption using KMS keys (AWS CLI)

You can use the [PutEncryptionConfiguration](#) API operation to enable encryption for your AWS IoT FleetWise account. The following example uses AWS CLI.

To enable encryption, run the following command.

- Replace *KMS key id* with the ID of the KMS key.

```
aws iotfleetwise put-encryption-configuration --kms-key-id KMS key id --encryption-type
KMS_BASED_ENCRYPTION
```

### Example response

```
{
 "kmsKeyId": "customer_kms_key_id",
 "encryptionStatus": "PENDING",
 "encryptionType": "KMS_BASED_ENCRYPTION"
}
```

### KMS key policy

After you create a KMS key, you must, at minimum, add the following statement to your KMS key policy for it to work with AWS IoT FleetWise.

```
{
```

```
"Sid": "Allow FleetWise to encrypt and decrypt data when customer managed KMS key
based encryption is enabled",
"Effect": "Allow",
"Principal": {
 "Service": "iotfleetwise.amazonaws.com"
},
"Action": [
 "kms:GenerateDataKey*",
 "kms:Decrypt",
 "kms:DescribeKey",
 "kms:CreateGrant",
 "kms:RetireGrant",
 "kms:RevokeGrant"
],
"Resource": "*"
}
```

For more information about editing a KMS key policy for use with AWS IoT FleetWise, see [Changing a key policy](#) in the *AWS Key Management Service Developer Guide*.

### Important

When you add the new sections to your KMS key policy, don't change any existing sections in the policy. AWS IoT FleetWise can't perform operations to your data if encryption is enabled for AWS IoT FleetWise and any of the following is true:

- The KMS key is disabled or deleted.
- The KMS key policy isn't correctly configured for the service.

## Controlling access with AWS IoT FleetWise

The following sections cover how to control access to and from your AWS IoT FleetWise resources. The information they cover includes how to grant your application access so AWS IoT FleetWise can transfer vehicle data during campaigns. They also describe how you can grant AWS IoT FleetWise access to your Amazon S3 (S3) bucket or Amazon Timestream database and table to store data.

The technology for managing all these forms of access is AWS Identity and Access Management (IAM). For more information about IAM, see [What is IAM?](#)

### Contents

- [Grant AWS IoT FleetWise access to an Amazon S3 destination](#)
- [Grant AWS IoT FleetWise access to a Amazon Timestream destination](#)

## Grant AWS IoT FleetWise access to an Amazon S3 destination

When you use an Amazon S3 destination, AWS IoT FleetWise delivers vehicle data to your S3 bucket and can optionally use an AWS KMS key that you own for data encryption. If error logging is enabled, AWS IoT FleetWise also sends data delivery errors to your CloudWatch log group and streams. You're required to have an IAM role when creating a delivery stream.

AWS IoT FleetWise uses a bucket policy with the service principal for the S3 destination. For more information about adding bucket policies, see [Adding a bucket policy by using the Amazon S3 console](#) in the *Amazon Simple Storage Service User Guide*.

Use the following access policy to enable AWS IoT FleetWise to access your S3 bucket. If you don't own the S3 bucket, add `s3:PutObjectACL` to the list of Amazon S3 actions. This grants the bucket owner full access to the objects delivered by AWS IoT FleetWise. For more information about how you can secure access to objects in your buckets, see [Bucket policy examples](#) in the *Amazon Simple Storage Service User Guide*.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": [
 "iotfleetwise.amazonaws.com"
]
 },
 "Action": [
 "s3:ListBucket"
],
 "Resource": "arn:aws:s3:::bucket-name"
 },
 {
 "Effect": "Allow",
 "Principal": {
 "Service": [
 "iotfleetwise.amazonaws.com"
]
 },
 "Action": [
 "s3:PutObject"
],
 "Resource": "arn:aws:s3:::bucket-name/*"
 }
]
}
```

```

]
 },
 "Action": [
 "s3:GetObject",
 "s3:PutObject"
],
 "Resource": "arn:aws:s3:::bucket-name/*",
 "Condition": {
 "StringEquals": {
 "aws:SourceArn": "campaign-arn",
 "aws:SourceAccount": "account-id"
 }
 }
}
]
}

```

The following bucket policy is for all campaigns in an account in an AWS Region.

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Principal": {
 "Service": [
 "iotfleetwise.amazonaws.com"
]
 },
 "Action": [
 "s3:ListBucket"
],
 "Resource": "arn:aws:s3:::bucket-name"
 },
 {
 "Effect": "Allow",
 "Principal": {
 "Service": [
 "iotfleetwise.amazonaws.com"
]
 },
 "Action": [
 "s3:GetObject",

```

```

 "s3:PutObject"
],
 "Resource": "arn:aws:s3:::bucket-name/*",
 "Condition": {
 "StringLike": {
 "aws:SourceArn": "arn:aws:iotfleetwise:region:account-id:campaign/*",
 "aws:SourceAccount": "account-id"
 }
 }
}
]
}

```

If you have a KMS key attached to your S3 bucket, the key will need the following policy. For information about key management, see [Protecting data using server-side encryption with AWS Key Management Service keys \(SSE-KMS\)](#) in the *Amazon Simple Storage Service User Guide*.

```

{
 "Version": "2012-10-17",
 "Effect": "Allow",
 "Principal": {
 "Service": "iotfleetwise.amazonaws.com"
 },
 "Action": [
 "kms:GenerateDataKey",
 "kms:Decrypt"
],
 "Resource": "key-arn"
}

```

### Important

When you create a bucket, S3 creates a default access control lists (ACL) that grants the resource owner full control over the resource. If AWS IoT FleetWise can't deliver data to S3, make sure you disable the ACL on the S3 bucket. For more information, see [Disabling ACLs for all new buckets and enforcing Object Ownership](#) in the *Amazon Simple Storage Service User Guide*.

## Grant AWS IoT FleetWise access to a Amazon Timestream destination

When you use a Timestream destination, AWS IoT FleetWise delivers vehicle data to a Timestream table. You must attach the policies to the IAM role to allow AWS IoT FleetWise to send data to Timestream.

If you use the console to [create a campaign](#), AWS IoT FleetWise automatically attaches the required policy to the role.

Before you start, check the following:

### Important

- You must use the same AWS Region when you create Timestream resources for AWS IoT FleetWise. If you switch AWS Regions, you might have issues accessing the Timestream resources.
  - AWS IoT FleetWise is available in US East (N. Virginia) and Europe (Frankfurt).
  - For the list of supported Regions, see [Timestream endpoints and quotas](#) in the *AWS General Reference*.
- 
- You must have a Timestream database. For a tutorial, see [Create a database](#) in the *Amazon Timestream Developer Guide*.
  - You must have a table created in the specified Timestream database. For a tutorial, see [Create a table](#) in the *Amazon Timestream Developer Guide*.

You can use the AWS CLI to create an IAM role with a trust policy for Timestream. To create an IAM role, run the following command.

### To create an IAM role with a trust policy

- Replace *TimestreamExecutionRole* with the name of the role you're creating.
- Replace *trust-policy* with the JSON file that contains the trust policy.

```
aws iam create-role --role-name TimestreamExecutionRole --assume-role-policy-document
file://trust-policy.json
```

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "timestreamTrustPolicy",
 "Effect": "Allow",
 "Principal": {
 "Service": "iotfleetwise.amazonaws.com"
 },
 "Action": "sts:AssumeRole",
 "Condition": {
 "StringEquals": {
 "aws:SourceArn": [
 "arn:aws:iotfleetwise:region:account-id:campaign/campaign-name"
],
 "aws:SourceAccount": [
 "account-id"
]
 }
 }
 }
]
}
```

Create a permissions policy to give AWS IoT FleetWise permissions to write data into Timestream. To create a permissions policy, run the following command.

### To create a permissions policy

- Replace *AWSIoT FleetwiseAccessTimestreamPermissionsPolicy* with the name of the policy you're creating.
- Replace *permissions-policy* with the name of the JSON file that contains the permissions policy.

```
aws iam create-policy --policy-name AWSIoT FleetwiseAccessTimestreamPermissionsPolicy --
policy-document file://permissions-policy.json
```

```
{
 "Version": "2012-10-17",
 "Statement": [
```



```
{
 "Sid": "timestreamIngestion",
 "Effect": "Allow",
 "Action": [
 "timestream:WriteRecords",
 "timestream:Select",
 "timestream:DescribeTable"
],
 "Resource": "table-arn"
},
{
 "Sid": "timestreamDescribeEndpoint",
 "Effect": "Allow",
 "Action": [
 "timestream:DescribeEndpoints"
],
 "Resource": "*"
}
]
```

## To attach the permissions policy to your IAM role

1. From the output, copy the Amazon Resource Name (ARN) of the permissions policy.
2. To attach the IAM permissions policy to your IAM role, run the following command.
  - Replace *permissions-policy-arn* with the ARN that you copied in the previous step.
  - Replace *TimestreamExecutionRole* with the name of the IAM role that you created.

```
aws iam attach-role-policy --policy-arn permissions-policy-arn --role-
name TimestreamExecutionRole
```

For more information, see [Access management for AWS resources](#) in the *IAM User Guide*.

## Identity and Access Management for AWS IoT FleetWise

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in)

and *authorized* (have permissions) to use AWS IoT FleetWise resources. IAM is an AWS service that you can use with no additional charge.

## Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How AWS IoT FleetWise works with IAM](#)
- [Identity-based policy examples for AWS IoT FleetWise](#)
- [Troubleshooting AWS IoT FleetWise identity and access](#)

## Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS IoT FleetWise.

**Service user** – If you use the AWS IoT FleetWise service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS IoT FleetWise features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS IoT FleetWise, see [Troubleshooting AWS IoT FleetWise identity and access](#).

**Service administrator** – If you're in charge of AWS IoT FleetWise resources at your company, you probably have full access to AWS IoT FleetWise. It's your job to determine which AWS IoT FleetWise features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS IoT FleetWise, see [How AWS IoT FleetWise works with IAM](#).

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS IoT FleetWise. To view example AWS IoT FleetWise identity-based policies that you can use in IAM, see [Identity-based policy examples for AWS IoT FleetWise](#).

## Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see [Signing AWS API requests](#) in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Multi-factor authentication](#) in the *AWS IAM Identity Center User Guide* and [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

### AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

## Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

## IAM users and groups

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see [Rotate access keys regularly for use cases that require long-term credentials](#) in the *IAM User Guide*.

An [IAM group](#) is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

## IAM roles

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or

AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see [Creating a role for a third-party Identity Provider](#) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see [Permission sets](#) in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
- **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).
- **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

## Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

## Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

## Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

## Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

## Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

## Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

## How AWS IoT FleetWise works with IAM

Before you use IAM to manage access to AWS IoT FleetWise, learn what IAM features are available to use with AWS IoT FleetWise.



## IAM features you can use with AWS IoT FleetWise

| IAM feature                             | AWS IoT FleetWise support |
|-----------------------------------------|---------------------------|
| <a href="#">Identity-based policies</a> | Yes                       |
| <a href="#">Resource-based policies</a> | No                        |
| <a href="#">Policy actions</a>          | Yes                       |
| <a href="#">Policy resources</a>        | Yes                       |
| <a href="#">Policy condition keys</a>   | Yes                       |
| <a href="#">ACLs</a>                    | No                        |
| <a href="#">ABAC (tags in policies)</a> | Partial                   |
| <a href="#">Temporary credentials</a>   | Yes                       |
| <a href="#">Principal permissions</a>   | Yes                       |
| <a href="#">Service roles</a>           | No                        |
| <a href="#">Service-linked roles</a>    | No                        |

To get a high-level view of how AWS IoT FleetWise and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

### Identity-based policies for AWS IoT FleetWise

|                                  |     |
|----------------------------------|-----|
| Supports identity-based policies | Yes |
|----------------------------------|-----|

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

## Identity-based policy examples for AWS IoT FleetWise

To view examples of AWS IoT FleetWise identity-based policies, see [Identity-based policy examples for AWS IoT FleetWise](#).

## Resource-based policies within AWS IoT FleetWise

|                                  |    |
|----------------------------------|----|
| Supports resource-based policies | No |
|----------------------------------|----|

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are *IAM role trust policies* and *Amazon S3 bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

## Policy actions for AWS IoT FleetWise

|                         |     |
|-------------------------|-----|
| Supports policy actions | Yes |
|-------------------------|-----|

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Action element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AWS IoT FleetWise actions, see [Actions Defined by AWS IoT FleetWise](#) in the *Service Authorization Reference*.

Policy actions in AWS IoT FleetWise use the following prefix before the action:

```
iotfleetwise
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
 "iotfleetwise:action1",
 "iotfleetwise:action2"
]
```

You can specify multiple actions using wildcards (\*). For example, to specify all actions that begin with the word List, include the following action:

```
"Action": "iotfleetwise:List*"
```

To view examples of AWS IoT FleetWise identity-based policies, see [Identity-based policy examples for AWS IoT FleetWise](#).

## Policy resources for AWS IoT FleetWise

|                           |     |
|---------------------------|-----|
| Supports policy resources | Yes |
|---------------------------|-----|

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (\*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AWS IoT FleetWise resource types and their ARNs, see [Resources Defined by AWS IoT FleetWise](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by AWS IoT FleetWise](#).

To view examples of AWS IoT FleetWise identity-based policies, see [Identity-based policy examples for AWS IoT FleetWise](#).

## Policy condition keys for AWS IoT FleetWise

|                                                 |     |
|-------------------------------------------------|-----|
| Supports service-specific policy condition keys | Yes |
|-------------------------------------------------|-----|

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Condition element (or Condition *block*) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

To see a list of AWS IoT FleetWise condition keys, see [Condition Keys for AWS IoT FleetWise](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions Defined by AWS IoT FleetWise](#).

To view examples of AWS IoT FleetWise identity-based policies, see [Identity-based policy examples for AWS IoT FleetWise](#).

## Access control lists (ACLs) in AWS IoT FleetWise

Supports ACLs

No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

## Attribute-based access control (ABAC) with AWS IoT FleetWise

Supports ABAC (tags in policies)

Partial


Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [What is ABAC?](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

 **Note**

AWS IoT FleetWise only supports `iam:PassRole`, which is required for the `CreateCampaign` API operation.

## Using Temporary credentials with AWS IoT FleetWise

|                                |     |
|--------------------------------|-----|
| Supports temporary credentials | Yes |
|--------------------------------|-----|

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

## Cross-service principal permissions for AWS IoT FleetWise

|                                        |     |
|----------------------------------------|-----|
| Supports forward access sessions (FAS) | Yes |
|----------------------------------------|-----|

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see [Forward access sessions](#).

## Service roles for AWS IoT FleetWise

|                        |    |
|------------------------|----|
| Supports service roles | No |
|------------------------|----|

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

### Warning

Changing the permissions for a service role might break AWS IoT FleetWise functionality. Edit service roles only when AWS IoT FleetWise provides guidance to do so.

## Service-linked roles for AWS IoT FleetWise

|                               |    |
|-------------------------------|----|
| Supports service-linked roles | No |
|-------------------------------|----|

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see [AWS services that work with IAM](#). Find a service in the table that includes a Yes in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

## Using service-linked roles for AWS IoT FleetWise

AWS IoT FleetWise uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to AWS IoT FleetWise. Service-linked roles are predefined by AWS IoT FleetWise and include the permissions that AWS IoT FleetWise needs to send metrics to Amazon CloudWatch. For more information, see [Monitoring AWS IoT FleetWise with Amazon CloudWatch](#).

A service-linked role makes setting up AWS IoT FleetWise quicker because you don't have to manually add the necessary permissions. AWS IoT FleetWise defines the permissions of its service-linked roles, and unless defined otherwise, only AWS IoT FleetWise can assume its roles. The defined permissions include the trust policy and the permissions policy. This permissions policy can't be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your AWS IoT FleetWise resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#), and look for the services that have **Yes** in the **Service-linked roles** column. To view the service-linked role documentation for that service, choose a **Yes** with a link.

### Service-linked role permissions for AWS IoT FleetWise

AWS IoT FleetWise uses the service-linked role named **AWSServiceRoleForIoT FleetWise** – An AWS managed policy that is used for all out-of-the-box permissions for AWS IoT FleetWise.

The **AWSServiceRoleForIoT FleetWise** service-linked role trusts the following services to assume the role:

- `IoT FleetWise`

The role permissions policy named `AWSIoT FleetWiseServiceRolePolicy` allows AWS IoT FleetWise to complete the following actions on the specified resources:

- Action: `cloudwatch:PutMetricData` on resource: \*

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-linked role permissions](#) in the *IAM User Guide*.



## Creating a service-linked role for AWS IoT FleetWise

You don't need to manually create a service-linked role. When you register an account in the AWS IoT FleetWise console, the AWS CLI, or the AWS API, AWS IoT FleetWise creates the service-linked role for you. For more information, see [Configuring settings](#).

### Creating a service-linked role in AWS IoT FleetWise (console)

You don't need to manually create a service-linked role. When you register an account in the AWS IoT FleetWise console, the AWS CLI, or the AWS API, AWS IoT FleetWise creates the service-linked role for you.

### Editing a service-linked role for AWS IoT FleetWise

You can't edit the `AWSServiceRoleForIoT FleetWise` service-linked role in AWS IoT FleetWise. Because various entities might reference any service-linked role you create, you can't change the name of the role. However, you can edit the description of the role by using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

### Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

#### Note

If AWS IoT FleetWise is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again. To learn how to delete the service-linked-role through the console, AWS CLI, or AWS API, see [Using service-linked roles](#) in the *IAM User Guide*.

If you delete this service-linked role, and then need to create it again, you can register an account with AWS IoT FleetWise. AWS IoT FleetWise then creates the service-linked role for you again.

## Identity-based policy examples for AWS IoT FleetWise

By default, users and roles don't have permission to create or modify AWS IoT FleetWise resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line

Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Creating IAM policies](#) in the *IAM User Guide*.

For details about actions and resource types defined by AWS IoT FleetWise, including the format of the ARNs for each of the resource types, see [Actions, Resources, and Condition Keys for AWS IoT FleetWise](#) in the *Service Authorization Reference*.

## Topics

- [Policy best practices](#)
- [Using the AWS IoT FleetWise console](#)
- [Allow users to view their own permissions](#)
- [Access resources in Amazon Timestream](#)

## Policy best practices

Identity-based policies determine whether someone can create, access, or delete AWS IoT FleetWise resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to

specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Configuring MFA-protected API access](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

## Using the AWS IoT FleetWise console

To access the AWS IoT FleetWise console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AWS IoT FleetWise resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AWS IoT FleetWise console, also attach the AWS IoT FleetWise ConsoleAccess or ReadOnly AWS managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

## Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```

{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "ViewOwnUserInfo",
 "Effect": "Allow",
 "Action": [
 "iam:GetUserPolicy",
 "iam:ListGroupsWithUser",
 "iam:ListAttachedUserPolicies",
 "iam:ListUserPolicies",
 "iam:GetUser"
],
 "Resource": ["arn:aws:iam::*:user/${aws:username}"]
 },
 {
 "Sid": "NavigateInConsole",
 "Effect": "Allow",
 "Action": [
 "iam:GetGroupPolicy",
 "iam:GetPolicyVersion",
 "iam:GetPolicy",
 "iam:ListAttachedGroupPolicies",
 "iam:ListGroupPolicies",
 "iam:ListPolicyVersions",
 "iam:ListPolicies",
 "iam:ListUsers"
],
 "Resource": "*"
 }
]
}

```

## Access resources in Amazon Timestream

Before using AWS IoT FleetWise, you must register your AWS account, IAM, and Amazon Timestream resources to grant AWS IoT FleetWise permission to send vehicle data to AWS Cloud on your behalf. To register, you need:

- An Amazon Timestream database.
- A table created in the specified Amazon Timestream database.
- An IAM role that allows AWS IoT FleetWise to send data to Amazon Timestream.

For more information, including procedures and example policies, see [Configure Settings](#).

## Troubleshooting AWS IoT FleetWise identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS IoT FleetWise and IAM.

### Topics

- [I am not authorized to perform an action in AWS IoT FleetWise](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my AWS IoT FleetWise resources](#)

### I am not authorized to perform an action in AWS IoT FleetWise

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about a fictional *myVehicle* resource but does not have the `iotfleetwise:GetVehicleStatus` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
iotfleetwise:GetVehicleStatus on resource: myVehicle
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *myVehicle* resource using the `iotfleetwise:GetVehicleStatus` action.

### I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS IoT FleetWise.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS IoT FleetWise. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

## I want to allow people outside of my AWS account to access my AWS IoT FleetWise resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS IoT FleetWise supports these features, see [How AWS IoT FleetWise works with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

# Compliance Validation for AWS IoT FleetWise

## Note

AWS IoT FleetWise isn't in scope of any AWS compliance programs.

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](#) and choose the compliance program that you are interested in. For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- [Architecting for HIPAA Security and Compliance on Amazon Web Services](#) – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

## Note

Not all AWS services are HIPAA eligible. For more information, see the [HIPAA Eligible Services Reference](#).

- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [AWS Customer Compliance Guides](#) – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).

- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see [Security Hub controls reference](#).
- [Amazon GuardDuty](#) – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.
- [AWS Audit Manager](#) – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

## Resilience in AWS IoT FleetWise

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

### Note

Data processed by AWS IoT FleetWise is stored in an Amazon Timestream database. Timestream supports backups to other AWS Availability Zones or Regions. However, you can write your own application using the Timestream SDK to query data and save it to the destination of your choice.

For more information about Amazon Timestream, see the [in the Amazon Timestream Developer Guide](#).



# Infrastructure security in AWS IoT FleetWise

As a managed service, AWS IoT FleetWise is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see [AWS Cloud Security](#). To design your AWS environment using the best practices for infrastructure security, see [Infrastructure Protection](#) in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access AWS IoT FleetWise through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

You can call these API operations from any network location, but AWS IoT FleetWise does support resource-based access policies, which can include restrictions based on the source IP address. You can also use AWS IoT FleetWise policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given AWS IoT FleetWise resource from only the specific VPC within the AWS network.

## Topics

- [Connecting to AWS IoT FleetWise through an interface VPC endpoint](#)

## Connecting to AWS IoT FleetWise through an interface VPC endpoint

You can connect directly to AWS IoT FleetWise by using an [interface VPC endpoint \(AWS PrivateLink\)](#) in your Virtual Private Cloud (VPC), instead of connecting over the internet. When you use an interface VPC endpoint, communication between your VPC and AWS IoT FleetWise is conducted entirely within the AWS network. Each VPC endpoint is represented by one or more [Elastic network interfaces](#) (ENIs) with private IP addresses in your VPC subnets.

The interface VPC endpoint connects your VPC directly to AWS IoT FleetWise without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the AWS IoT FleetWise API.

To use AWS IoT FleetWise through your VPC, you must connect from an instance that is inside the VPC or connect your private network to your VPC by using an AWS Virtual Private Network (VPN) or AWS Direct Connect. For information about Amazon VPN, see [VPN connections](#) in the *Amazon Virtual Private Cloud User Guide*. For information about AWS Direct Connect, see [Creating a connection](#) in the *AWS Direct Connect User Guide*.

You can create an interface VPC endpoint to connect to AWS IoT FleetWise by using the AWS console or AWS Command Line Interface (AWS CLI) commands. For more information, see [Creating an interface endpoint](#).

After you create an interface VPC endpoint, if you enable private DNS hostnames for the endpoint, the default AWS IoT FleetWise endpoint resolves to your VPC endpoint. The default service name endpoint for AWS IoT FleetWise is in the following format.

```
iotfleetwise.Region.amazonaws.com
```

If you don't enable private DNS hostnames, Amazon VPC provides a DNS endpoint name that you can use in the following format.

```
VPCE_ID.iotfleetwise.Region.vpce.amazonaws.com
```

For more information, see [Interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.

AWS IoT FleetWise supports making calls to all of its [API actions](#) inside your VPC.

You can attach VPC endpoint policies to a VPC endpoint to control access for IAM principals. You can also associate security groups with a VPC endpoint to control inbound and outbound access based on the origin and destination of network traffic, such as a range of IP addresses. For more information, see [Controlling access to services with VPC endpoints](#).

## Creating a VPC endpoint policy for AWS IoT FleetWise

You can create a policy for Amazon VPC endpoints for AWS IoT FleetWise to specify the following:

- The principal that can or can't perform actions
- The actions that can or can't be performed

For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

### Example – VPC endpoint policy to deny all access from a specified AWS account

The following VPC endpoint policy denies AWS account *123456789012* all API calls using the endpoint.

```
{
 "Statement": [
 {
 "Action": "*",
 "Effect": "Allow",
 "Resource": "*",
 "Principal": "*"
 },
 {
 "Action": "*",
 "Effect": "Deny",
 "Resource": "*",
 "Principal": {
 "AWS": [
 "123456789012"
]
 }
 }
]
}
```

### Example – VPC endpoint policy to allow VPC access only to a specified IAM principal (user)

The following VPC endpoint policy allows full access only to the a user *lijuan* in AWS account *123456789012*. It denies all other IAM principals access to the endpoint.

```
{
 "Statement": [
 {
 "Action": "*",
```

```
 "Effect": "Allow",
 "Resource": "*",
 "Principal": {
 "AWS": [
 "arn:aws:iam::123456789012:user/lijuan"
]
 }
]
}
```

### Example – VPC endpoint policy for AWS IoT FleetWise actions

The following is an example of an endpoint policy for AWS IoT FleetWise. When attached to an endpoint, this policy grants access to the listed AWS IoT FleetWise actions for the IAM user *fleetWise* in the AWS account *123456789012*.

```
{
 "Statement": [
 {
 "Principal": {
 "AWS": [
 "arn:aws:iam::123456789012:user/fleetWise"
],
 },
 "Resource": "*",
 "Effect": "Allow",
 "Action": [
 "iotfleetwise:ListFleets",
 "iotfleetwise:ListCampaigns",
 "iotfleetwise:CreateVehicle",
]
 }
]
}
```

## Configuration and vulnerability analysis in AWS IoT FleetWise

IoT environments can consist of large numbers of devices that have diverse capabilities, are long-lived, and are geographically distributed. These characteristics make device setup complex and error-prone. Also, because devices are often constrained in computational power, memory, and storage capabilities, the use of encryption and other forms of security on the devices is limited. Devices often use software with known vulnerabilities. These factors make IoT devices, including

vehicles collecting data for AWS IoT FleetWise, an attractive target for hackers and make it difficult to secure them on an ongoing basis.

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS [shared responsibility model](#).

## Security best practices for AWS IoT FleetWise

AWS IoT FleetWise provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

To learn about security in AWS IoT see [Security best practices in AWS IoT Core](#) in the *AWS IoT Developer Guide*

### Grant minimum possible permissions

Follow the principle of least privilege by using the minimum set of permissions in IAM roles. Limit the use of the \* wildcard for the Action and Resource properties in your IAM policies. Instead, declare a finite set of actions and resources when possible. For more information about least privilege and other policy best practices, see [the section called "Policy best practices"](#).

### Don't log sensitive information

You should prevent the logging of credentials and other personally identifiable information (PII). We recommend that you implement the following safeguards:

- Don't use sensitive information in device names.
- Don't use sensitive information in the names and IDs of AWS IoT FleetWise resources, for example in the names of campaigns, decoder manifests, vehicle models, and signal catalogs, or the IDs of vehicles and fleets.

### Use AWS CloudTrail to view API call history

You can view a history of AWS IoT FleetWise API calls made on your account for security analysis and operational troubleshooting purposes. To receive a history of AWS IoT FleetWise API calls

made on your account, simply turn on CloudTrail in the AWS Management Console. For more information, see [the section called “CloudTrail logs”](#).

## Keep your device clock in sync

It's important to have an accurate time on your device. X.509 certificates have an expiry date and time. The clock on your device is used to verify that a server certificate is still valid. Device clocks can drift over time or batteries can get discharged.

For more information, see the [Keep your device's clock in sync](#) best practice in the *AWS IoT Core Developer Guide*.

# Monitoring AWS IoT FleetWise

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS IoT FleetWise and your other AWS solutions. AWS provides the following monitoring tools to watch AWS IoT FleetWise, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the [Amazon CloudWatch User Guide](#).
- *Amazon CloudWatch Logs* can be used to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account. Then, it delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the [AWS CloudTrail User Guide](#).

## Monitoring AWS IoT FleetWise with Amazon CloudWatch

Amazon CloudWatch metrics are a way to monitor your AWS resources and how they're performing. AWS IoT FleetWise sends metrics to CloudWatch. You can use the AWS Management Console, the AWS CLI, or an API to list the metrics that AWS IoT FleetWise sends to CloudWatch. For more information, see the [Amazon CloudWatch User Guide](#).

### Important

You must configure settings so that AWS IoT FleetWise can send metrics to CloudWatch. For more information, see [Configuring settings](#).

The `AWS/IoTFleetWise` namespace includes the following metrics.

## Signal metrics

| Metric                 | Description                                                                                                                                                                                                                                                        |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IllegalMessageFromEdge | <p>A message sent from the vehicle and received by AWS IoT FleetWise didn't match the required format.</p> <p>Units: Count</p> <p>Dimensions: VehicleName</p> <p>Valid statistics: Sum</p>                                                                         |
| MessageThrottled       | <p>A message sent from the vehicle to AWS IoT FleetWise was throttled. This is because you exceeded the <a href="#">service limits</a> for this account in the current Region.</p> <p>Units: Count</p> <p>Dimensions: VehicleName</p> <p>Valid statistics: Sum</p> |
| ModelingError          | <p>A message sent from the vehicle and received by AWS IoT FleetWise contains signals that fail to validate against the vehicle model.</p> <p>Units: Count</p> <p>Dimensions: ModelManifestName</p> <p>Valid statistics: Sum</p>                                   |
| DecodingError          | <p>A message sent from the vehicle and received by AWS IoT FleetWise contains signals that fail to decoder against the vehicle's decoder manifest.</p> <p>Units: Count</p>                                                                                         |



| Metric | Description                                          |
|--------|------------------------------------------------------|
|        | Dimensions: DecoderName<br><br>Valid statistics: Sum |

## Campaign metrics

| Metric           | Description                                                                                                                                                                         |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VehicleNotFound  | A message sent from the vehicle and received by AWS IoT FleetWise, where the vehicle is unknown.<br><br>Units: Count<br><br>Dimensions: VehicleName<br><br>Valid statistics: Sum    |
| CampaignInvalid  | A message sent from the vehicle and received by AWS IoT FleetWise, where the campaign isn't valid.<br><br>Units: Count<br><br>Dimensions: CampaignName<br><br>Valid statistics: Sum |
| CampaignNotFound | A message sent from the vehicle and received by AWS IoT FleetWise, where the campaign is unknown.<br><br>Units: Count<br><br>Dimensions: CampaignName<br><br>Valid statistics: Sum  |

## Campaign data destination metrics

| Metric               | Description                                                                                                                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TimestreamWriteError | <p>AWS IoT FleetWise couldn't write a message from the vehicle to the Amazon Timestream table.</p> <p>Units: Count</p> <p>Dimensions: DatabaseName, TableName</p> <p>Valid statistics: Sum</p>                |
| S3WriteError         | <p>AWS IoT FleetWise couldn't write a message from the vehicle to the Amazon Simple Storage Service (Amazon S3) bucket.</p> <p>Units: Count</p> <p>Dimensions: BucketName</p> <p>Valid statistics: Sum</p>    |
| S3ReadError          | <p>AWS IoT FleetWise couldn't read an object key from the vehicle in the Amazon Simple Storage Service (Amazon S3) bucket.</p> <p>Units: Count</p> <p>Dimensions: BucketName</p> <p>Valid statistics: Sum</p> |

## Customer managed AWS KMS key metrics

| Metric             | Description                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------|
| KMSKeyAccessDenied | <p>AWS IoT FleetWise couldn't write a message from the vehicle to the Timestream table or</p> |

| Metric | Description                                                         |
|--------|---------------------------------------------------------------------|
|        | the Amazon S3 bucket because of an AWS KMS key access denied error. |
|        | Units: Count                                                        |
|        | Dimensions: KMSKeyId                                                |
|        | Valid statistics: Sum                                               |

## Monitoring AWS IoT FleetWise with Amazon CloudWatch Logs

Amazon CloudWatch Logs monitors the events that occur in your resources and alerts you if there are any issues. If you receive an alert, you can access the log files to get information about the specific event. For more information, see the [Amazon CloudWatch Logs User Guide](#).

### Viewing AWS IoT FleetWise logs in the CloudWatch console

#### Important

Before you can see the AWS IoT FleetWise log group in the CloudWatch console, make sure that the following is true:

- You've enabled logging in AWS IoT FleetWise. For more information about logging, see [Configure AWS IoT FleetWise logging](#).
- There are already log entries written by AWS IoT operations.

#### To view your AWS IoT FleetWise logs in the CloudWatch console

1. Open the [CloudWatch console](#).
2. On the navigation pane, choose **Logs, Log groups**.
3. Choose the log group.
4. Choose **Search log group**. You'll see a complete list of the log events generated for your account.

5. Choose the expand icon to look at an individual stream and find all logs that have a log level of ERROR.

You can also enter a query in the **Filter events** search box. For example, you can try the following query:

```
{ $.logLevel = "ERROR" }
```

For more information about creating filter expressions, see [Filter and pattern syntax](#) in the *Amazon CloudWatch Logs User Guide*.

### Example log entry

```
{
 "accountId": "123456789012",
 "vehicleName": "test-vehicle",
 "message": "Unrecognized signal ID",
 "eventType": "MODELING_ERROR",
 "logLevel": "ERROR",
 "timestamp": 1685743214239,
 "campaignName": "test-campaign",
 "signalCatalogName": "test-catalog",
 "signalId": 10242
}
```

### Signal event types

| Event type                | Description                                                                                                                                                                                                                                                                                    |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MODELING_ERROR            | <p>A message sent from the vehicle and received by AWS IoT FleetWise contains signals that fail to validate against the vehicle model.</p> <p>Attributes: vehicleName, campaignName, signalCatalogName, signalId, signalValue, signalValueRangeMin, signalValueRangeMax, modelManifestName</p> |
| ILLEGAL_MESSAGE_FROM_EDGE | <p>A message sent from the vehicle and received by AWS IoT FleetWise didn't match the required format.</p>                                                                                                                                                                                     |

| Event type     | Description                                                                                                                                                                                                                                                             |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | Attributes: vehicleName, campaignName, signalCatalogName                                                                                                                                                                                                                |
| DECODING_ERROR | <p>A message sent from the vehicle and received by AWS IoT FleetWise contains signals that fail to decoder against the vehicle's decoder manifest.</p> <p>Attributes: campaignName, signalCatalogName, decoderManifestName, (optional) signalName, (optional) s3URI</p> |

### Campaign event types

| Event type         | Description                                                                                                                                                         |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VEHICLE_NOT_FOUND  | <p>A message sent from the vehicle and received by AWS IoT FleetWise, where the vehicle was unknown.</p> <p>Attributes: vehicleName, campaignName</p>               |
| CAMPAIGN_NOT_FOUND | <p>A message sent from the vehicle and received by AWS IoT FleetWise, where the campaign was unknown.</p> <p>Attributes: vehicleName (optional), campaignName</p>   |
| CAMPAIGN_INVALID   | <p>A message sent from the vehicle and received by AWS IoT FleetWise, where the campaign was not valid.</p> <p>Attributes: vehicleName (optional), campaignName</p> |

## Campaign data destination event types

| Event type             | Description                                                                                                                                                                                  |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TIMESTREAM_WRITE_ERROR | <p>AWS IoT FleetWise couldn't write a message from the vehicle to the Amazon Timestream table.</p> <p>Attributes: vehicleName, campaignName, timestreamDatabaseName, timestreamTableName</p> |
| S3_WRITE_ERROR         | <p>AWS IoT FleetWise couldn't write a message from the vehicle to the Amazon Simple Storage Service (Amazon S3) bucket.</p> <p>Attributes: campaignName, destinationName</p>                 |
| S3_READ_ERROR          | <p>AWS IoT FleetWise couldn't read an object key from the vehicle in the Amazon Simple Storage Service (Amazon S3) bucket.</p> <p>Attributes: campaignName, destinationName</p>              |

## Customer managed AWS KMS key event types

| Event type            | Description                                                                                                                                                       |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| KMS_KEY_ACCESS_DENIED | <p>AWS IoT FleetWise couldn't write a message from the vehicle to the Timestream table or the Amazon S3 bucket because of an AWS KMS key access denied error.</p> |

## Attributes

All CloudWatch Logs entries include these attributes:

**accountId**

Your AWS account ID.

**eventType**

The event type for which the log was generated. The value of the event type depends on the event that generated the log entry. Each log entry description includes the value of eventType for that log entry.

**logLevel**

The log level that is being used. For more information, see [Log levels](#) in the *AWS IoT Core Developer Guide*.

**message**

Contains specific details about the log.

**timestamp**

The epoch millisecond timestamp of when AWS IoT FleetWise processed the log.

**Optional attributes**

CloudWatch Logs entries optionally include these attributes, depending on the eventType:

**decoderManifestName**

The name of the decoder manifest that contains the signal.

**destinationName**

The name of the destination for vehicle data. For example, the Amazon S3 bucket name.

**campaignName**

The name of the campaign.

**signalCatalogName**

The name of the signal catalog that contains the signal.

**signalId**

The ID of the error signal.

**signalIds**

A list of error signal IDs.

**signalName**

The name of the signal.

**signalTimestampEpochMs**

The timestamp of the error signal.

**signalValue**

The value of the error signal.

**signalValueRangeMax**

The maximum range of the error signal.

**signalValueRangeMin**

The minimum range of the error signal.

**s3URI**

The Amazon S3 unique identifier of an Amazon Ion file from a vehicle message.

**timestreamDatabaseName**

The name of the Timestream database.

**timestreamTableName**

The name of the Timestream table.

**vehicleName**

The name of the vehicle.

## Configure AWS IoT FleetWise logging

You can send your AWS IoT FleetWise log data to a CloudWatch log group. CloudWatch Logs give visibility in case AWS IoT FleetWise fails to process messages from vehicles. For example, this can happen because of a faulty configuration or other client errors. You're notified of any errors so you can identify and mitigate issues.

Before you can send logs to CloudWatch, you must create a CloudWatch log group. Configure the log group with the same account and in the same Region that you used with AWS IoT FleetWise.



When you enable logging in AWS IoT FleetWise, provide the log group name. After logging is enabled, AWS IoT FleetWise delivers logs to the CloudWatch log group in log streams.

You can view log data sent from AWS IoT FleetWise in the CloudWatch console. For more information about configuring a CloudWatch log group and viewing log data, see [Working with Log Groups](#).

## Permissions to publish logs to CloudWatch

Configuring logging for a CloudWatch log group requires the permissions settings described in this section. For information about managing permissions, see [Access management for AWS resources](#) in the *IAM User Guide*.

With these permissions, you can change the logging configuration, configure log delivery for CloudWatch, and retrieve information about your log group.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Action": [
 "iotfleetwise:PutLoggingOptions",
 "iotfleetwise:GetLoggingOptions"
],
 "Resource": [
 "*"
],
 "Effect": "Allow",
 "Sid": "IoTFleetwiseLoggingOptionsAPI"
 }
],
 {
 "Sid": "IoTFleetwiseLoggingCWL",
 "Action": [
 "logs:CreateLogDelivery",
 "logs:GetLogDelivery",
 "logs:UpdateLogDelivery",
 "logs>DeleteLogDelivery",
 "logs:ListLogDeliveries",
 "logs:PutResourcePolicy",
 "logs:DescribeResourcePolicies",
 "logs:DescribeLogGroups"
]
 }
}
```

```
],
 "Resource": [
 "*"
],
 "Effect": "Allow"
}
]
```

When actions are permitted on all AWS resources, it's indicated in the policy with a "Resource" setting of "\*". This means that the actions are permitted on all AWS resources *that each action supports*.

## Configure logging in AWS IoT FleetWise (console)

This section describes how to use the AWS IoT FleetWise console to configure logging.

### To use the AWS IoT FleetWise console to configure logging

1. Open the [AWS IoT FleetWise console](#).
2. In the left pane, choose **Settings**.
3. In the **Logging** section of the **Settings** page, choose **Edit**.
4. In the **CloudWatch logging** section, enter the **Log group**.
5. To save your changes, choose **Submit**.

After you enable logging, you can view your log data in the [CloudWatch console](#).

## Configure default logging in AWS IoT FleetWise (CLI)

This section describes how to configure logging for AWS IoT FleetWise by using the CLI.

You can also perform this procedure with the API by using the methods in the AWS API that correspond to the CLI commands shown here. You can use the [GetLoggingOptions](#) API operation to fetch the current configuration and the [PutLoggingOptions](#) API operation to modify the configuration.

### To use the CLI to configure logging for AWS IoT FleetWise

1. To get the logging options for your account, use the **get-logging-options** command.

```
aws iotfleetwise get-logging-options
```

2. To enable logging, use the **put-logging-options** command.

```
aws iotfleetwise put-logging-options --cloud-watch-log-delivery
logType=ERROR,logGroupName=MyLogGroup
```

where:

### **logType**

The type of log to send data to CloudWatch Logs. To disable logging, change the value to OFF.

### **logGroupName**

The CloudWatch Logs group the operation sends data to. Make sure you create the log group name before you enable logging for AWS IoT FleetWise.

After you enable logging, see [Search log entries using the AWS CLI](#).

## **Logging AWS IoT FleetWise API calls using AWS CloudTrail**

AWS IoT FleetWise is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS IoT FleetWise. CloudTrail captures all API calls for AWS IoT FleetWise as events. The calls captured include calls from the AWS IoT FleetWise console and code calls to the AWS IoT FleetWise API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS IoT FleetWise. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS IoT FleetWise, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

### **AWS IoT FleetWise information in CloudTrail**

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS IoT FleetWise, that activity is recorded in a CloudTrail event along with other AWS service

events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for AWS IoT FleetWise, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for creating a trail](#)
- [CloudTrail supported services and integrations](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple Regions](#)
- [Receiving CloudTrail log files from multiple accounts](#)

All AWS IoT FleetWise actions are logged by CloudTrail and are documented in the [AWS IoT FleetWise API Reference](#). For example, calls to the `CreateCampaign`, `AssociateVehicleFleet`, and `GetModelManifest` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity element](#).

## Understanding AWS IoT FleetWise log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the *AssociateVehicleFleet* operation.

```
{
 "eventVersion": "1.05",
 "userIdentity": {
 "type": "AssumedRole",
 "principalId": "AIDACKCEVSQ6C2EXAMPLE",
 "arn": "arn:aws:iam::111122223333:assumed-role/NikkiWolf",
 "accountId": "111122223333",
 "accessKeyId": "access-key-id",
 "userName": "NikkiWolf"
 },
 "eventTime": "2021-11-30T09:56:35Z",
 "eventSource": "iotfleetwise.amazonaws.com",
 "eventName": "AssociateVehicleFleet",
 "awsRegion": "us-east-1",
 "sourceIPAddress": "192.0.2.21",
 "userAgent": "aws-cli/2.3.2 Python/3.8.8 Darwin/18.7.0 botocore/2.0.0",
 "requestParameters": {
 "fleetId": "f1234567890",
 "vehicleId": "v0213456789"
 },
 "responseElements": {
 },
 "requestID": "9f861429-11e3-11e8-9eea-0781b5c0ac21",
 "eventID": "17385819-4927-41ee-a6a5-29ml0br812v4",
 "eventType": "AwsApiCall",
 "recipientAccountId": "111122223333"
}
```

# Document history for the AWS IoT FleetWise Developer Guide

The following table describes the documentation releases for AWS IoT FleetWise.

| Change                                        | Description                                                                                                                                                                                                                                                                                                                                                    | Date              |
|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| <a href="#">Vision system data preview</a>    | You can use the preview of vision system data from AWS IoT FleetWise to collect and organize data from vehicle vision systems, including from cameras, radars, and lidars. It keeps both structured and unstructured vision system data, metadata (event ID, campaign, vehicle), and standard sensor (telemetry data) automatically synchronized in the cloud. | November 26, 2023 |
| <a href="#">AWS KMS customer managed keys</a> | AWS IoT FleetWise now supports AWS KMS customer managed keys. You can use KMS key to encrypt server-side data related to AWS IoT FleetWise resources (signal catalog, vehicle model, decoder manifest, vehicles, and data collection campaign configurations) stored in AWS Cloud.                                                                             | October 16, 2023  |
| <a href="#">Object storage in Amazon S3</a>   | AWS IoT FleetWise now supports storing data using Amazon Simple Storage                                                                                                                                                                                                                                                                                        | June 1, 2023      |

Service (Amazon S3). You can store data collected during campaigns in Amazon S3, in addition to Amazon Timestream.

### [General availability](#)

This is the public release of AWS IoT FleetWise.

September 27, 2022

### [Initial release](#)

This is the preview release of the AWS IoT FleetWise Developer Guide.

November 30, 2021