AWS Whitepaper

Designing Next Generation Vehicle Communication with AWS IoT Core and MQTT

Designing Next Generation Vehicle Communication with AWS IoT Core and MQTT: AWS Whitepaper

Table of Contents

Designing Next Generation Vehicle Communication with AWS IoT Core and MQTT

Utilizing MQTT and AWS IoT Core to Implement a Connected Vehicle Architecture

Publication date: **January 12, 2024** (*Document history*)

This whitepaper outlines the best practices for implementing an extensible, scalable and resilient communication architecture for the next generation of vehicles on AWS. Utilizing AWS IoT Core with its managed MQTT broker as the centralized communication platform for vehicle telemetry, provides OEMs this global platform to build their connected vehicle platforms upon and enables differentiated customer experiences and brand-new mobility use cases the industry has begun to demand. This paper reviews why MQTT and publish/subscribe pattern work best for connected vehicle platforms and reviews, in detail, the building blocks the AWS IoT Core can provide to enable OEMs and other connected mobility provides to build a managed platform with those tools.

This whitepaper is intended for vehicle manufacturer cloud architects and engineers or decision makers determining if AWS IoT Core is the proper solution for next-generation vehicle workloads to the cloud.

Introduction

The automotive industry is seeing a transformational change in the way consumers interact with their vehicles. This change is driving the size of the connected car global market, which is projected to reach $225 billion by 2027. We are seeing shifts in available use cases as automotive manufacturers (OEMs) and tier 1 suppliers begin to adopt these technologies in the vehicle, and these are enabling the following vehicle capabilities:

- The consumer's connected experience, in and out of the vehicle
- Car-sharing and ride sharing services, enabling a new mobility vertical
- Autonomous enhancements
- Fleet management

In the industry today, we see OEMs pushing both for newer revenue streams and the aggregation and monetization of vehicular data by connecting all vehicles to these platforms. Primarily, OEMs are looking to capture signal information off the vehicle allowing for more efficient identification

of fleet-wide issues, predictive maintenance and reduction of warranty claims, one of the larger expenses for OEMs. In addition, streamlining the operationalization of the vehicle's lifecycle, OEMs are introducing more customer-centric use cases via a connected vehicle functions which allow for convenient monitoring of the vehicle from your smart phone, remote commands, roadside assistance and emergency calling.

On most legacy connected vehicle platforms, many of these connected services were built using technologies not designed for the connected vehicle use cases, but understandably were the more prominent technologies available to implement. Additionally, when designing these platforms, cloud technologies were still in their infancy and OEMs that did switch their connected vehicle platforms to the cloud, used a lift and shift mechanism for many of their workloads, not optimizing the platform for cloud native architectures. This led to higher operational overheads, inefficiencies with capacity planning and, in turn, much higher variable costs to manage and run these platforms.

We are now starting to see a shift away from these architectures and OEMs have begun a movement to managed platforms and cloud native implementations, letting companies like AWS manage the security, extensibility and scalability of their connected vehicle platforms.

The next generation of vehicles will demand a better user experience, on a scalable, durable, extensible platform. This document will cover the implementation of AWS IoT Core as the next generation connected vehicle communication platform.

Are you Well-Architected?

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems in the cloud. The six pillars of the Framework allow you to learn architectural best practices for designing and operating reliable, secure, efficient, cost-effective, and sustainable systems. Using the AWS Well-Architected Tool, available at no charge in the AWS Management Console, you can review your workloads against these best practices by answering a set of questions for each pillar.

In the IoT Lens and IoT Lens Checklist, we focus on best practices for architecting your IoT applications on AWS.

In the Connected Mobility Lens, we focus on best practices for integrating technology into transportation systems and enhancing the overall mobility experience.

For more expert guidance and best practices for your cloud architecture—reference architecture deployments, diagrams, and whitepapers—refer to the AWS Architecture Center.

Challenges with connected vehicle platforms

The customer experience is a key differentiator for OEMs and when it comes to digital technologies, customers expect a fast and responsive user experience. There are some challenges within the automotive landscape to ensure this desired experience is met, including:

Scalability – Massive scalability is required for millions of devices and billions of messages streamed from the vehicles during peak usage. For vehicles, peak demand is morning and evenings, with large periods of limited activity late at night. The platform should be elastic, allowing for expansion and contraction as demand varies, ensuring cost and performance efficiency.

Durability – For fleet operators, connectivity to the cloud is vital portion of their connected vehicle platform. OEMs and fleet operators demand some of the stringent requirements for platform availability and attempt to minimize downtime disruptions from unplanned peaks in demand to ensure revenue and customer experiences are not impacted by these outages.

Unreliable networks – As the vehicles, connected to the cloud via a cellular network, move along in their journey, intermittent connectivity and drops from the network need to be managed by the platform to ensure messages are delivered as required. The communications protocol needs to support use cases around intermittent connectivity.

Instantaneous bidirectional communication – Ensuring the vehicle can receive and execute remote commands in near-real time are vital to the user's experience of interacting with the vehicle's systems, such as unlocking the door or starting the vehicle if a key has been misplaced.

Data residency and global availability – As OEMs continue to expand globally, navigating regulatory requirements like GDPR, CCPA, and UNR 155/156 and ensuring data privacy for their customers is a key concern. Additionally, ensuring the connected vehicle platform can operate at a global scale with local availability of cloud computing as close to the vehicle as possible is vital to the success of a connected vehicle platform.

Security – Securing vehicles requires collaboration across the enterprise. Securing vehicles goes beyond just the vehicle as connectivity proliferates. Security includes all applications and services the vehicle connects to, back-end resources that interact with vehicles, and any other services the vehicle relies on for connectivity. Customers need to consider identity across device, endpoint, and human entities interacting with the vehicle, network and application security of connected applications, detecting and responding to connected vehicle alerts, protecting vehicle data, and compliance.

Differentiators for automotive platforms on AWS

As the overall trends in the automotive industry tend to be moving away from on-premises infrastructure, we are also seeing a trend away from provisioned infrastructure running an open-source software handling ingest to adopting more cloud managed services.

Data ingestion and processing of vehicle telemetry data at scale requires a very heavy lift for the OEMs, especially handling peak processing for a short period of time. For the newer connected vehicles, 100 data sensors in the vehicle can generate 25 gigabytes of data per hour, with only a fraction of that being published to the connected car platform. If an OEM produces one million vehicles a year, and the peak usage is 60% of these vehicles on the road at the same time, the OEM will need to scale their platform to ingest a significant amount of data per vehicle, and that will grow as the use cases increase. AWS IoT Core, delivered as a distributed and managed platform, scales automatically as your vehicle fleet grows, and will scale as the vehicle data ingest increases as well.

There are other additional benefits to the automotive use case the managed services of AWS IoT Core:

- **Scalability** – Proper resource allocation specific to the OEMs workloads along with elastic infrastructure allows the OEM to not worry about a capacity planning effort as their fleet or customer use cases change over time - the operator can scale from zero to millions of vehicles automatically.
- **Global Availability** – AWS IoT Core is a global service, available in 21 regions throughout the world, allowing for global architectures and enabling the operator to comply with local data storage and privacy requirements providing customers the flexibility to choose which region their content is stored based on their requirements. In addition to regulatory compliance, having global endpoints allows the broker to be as close as possible to the vehicle, decreasing latency and increasing end user satisfaction.
- **Cost savings** – Using AWS IoT as the message broker within a connected vehicle platform reduces operational overhead as the OEM does not need to worry about provisioning infrastructure, cluster management, right-sizing compute, or administrative functions when building on a fully managed message broker
- **Reliability** – With millions of daily connected devices, and trillions of messages processed monthly and 99.9% uptime service-level agreement (SLA), end customers have discovered the reliability of AWS IoT Core for other workloads across industries.

- **Observability** – With many integrated services like CloudWatch, FleetHub and AWS IoT Device Management AWS IoT Core provides the ability to monitor your fleet with unified service metrics and dashboards across your fleet of vehicles and provides the ability to automate the detection and mitigation of problems.

In addition to the managed services of AWS IoT Core, there are several other advantages for automotive OEMs to select AWS IoT Core as its managed message broker for its connected vehicle platform.

Security of the AWS Cloud: Cloud security at AWS is the highest priority. The OEM benefits from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. In the cloud, you don't have to manage physical servers or storage devices. Instead, you use software-based security tools to monitor and protect the flow of information into and out of your cloud resources. Security is a shared responsibility between AWS and the customer. 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. Customers should carefully consider the services they choose as their responsibilities vary depending on the services used, the integration of those services into their IT environment, and applicable laws and regulations. Each connected device or client must establish trust by authenticating using principals such as X.509 certificates, which should not be shared between devices. All traffic to and from AWS IoT is sent securely most commonly using Mutual Transport Layer Security (mTLS) or other authentication mechanisms. Data moving between services is authenticated and authorized by Identity and Access Management. In addition, AWS IoT provides security services like AWS IoT Device Defender which allows you to automatically detect, generate & rotate expiring certificates on your devices

Securing connectivity platforms using AWS services and features:

Provisioning vehicle identities using AWS IoT Core: AWS provides several different ways to provision your vehicle's connectivity devices to IoT Core and to enable the device manufacturer to install unique X.509 certificates on the device. This flexibility allows OEMs to pick the best method for their specific use cases, whether the certificate is installed on the device before they are delivered or installing the certificate later on in the manufacturing process such as the first time the device tries to connect. This topic will be covered in more detail later in the document.

OTA and the edge: The AWS IoT Jobs service allows the OEM to securely update the vehicle software over the air (OTA). The binaries are signed in the cloud using AWS Signer which is to ensure integrity in transit to the vehicle such that the device agent can verify the signature against

a known code-signing certificate. The agent on the vehicle connects to the Jobs service via MQTT or HTTPs each boot to check if there is a software update scheduled. The job execution state is updated by the device agent to ensure the installation is successful.

MQTT for connected vehicle platforms

Over the years, MQTT has been adopted as the industry standard communication protocol for connected vehicle platforms. MQTT allows for a persistent, always-on connection between the vehicle and the cloud. With intermittent connectivity to the cloud, MQTT effortlessly handles buffering queuing and synchronizing when connectivity is re-established. Residing on top of the TCP/IP network stack, MQTT is a lightweight publish/subscribe messaging protocol designed for low-bandwidth, high latency and unreliable networks. These features make it the de facto standard in the industry to send high volumes of vehicle sensor data to the cloud.

Unlike a traditional client-server model where the client communicates directly with a specified endpoint, MQTT clients fall into two separate categories: publishers and subscribers. The publishers in most connected vehicle use cases will be the vehicles, and the subscribers will be the processors of that telemetry data by downstream cloud services. The publisher and subscriber are isolated from each other and are completely decoupled and never communicate directly with each other, providing extensibility to writing applications at the edge or in the cloud. Between the publisher and subscriber, the MQTT message broker handles routing of the messages from the publisher to any endpoint subscribed to a specific topic.

The message broker handles the management of topics (how publishers and subscribers communicate), the distribution of messages to subscribers and many other functions to ensure acceptance and delivery of the messages. AWS IoT Core offers this functionality as a managed broker, so no code, setup, or provisioning is necessary by the customers to begin working with MQTT on AWS IoT Core.

Recently, the MQTT specification was updated to from version 3.1.1 to MQTT version 5. AWS IoT Core has adopted this specification with many new connected vehicle specific features that we will discuss over the next section. With the latest announcement of support for MQTT5, and the features that align to that version of the specification, AWS IoT Core is an industry leading managed message broker for connected vehicle workloads.

The capability to provide separation of concerns between the publisher and subscribers, the bi-directional communication, the ability to define quality of service for the messages, the lightweight code footprint at the edge and the advanced message retention policies around unreliable networks makes. This makes MQTT an easy choice for delivering connected vehicle workloads to the cloud.

MQTTv5 support on AWS IoT Core

Recently, AWS IoT Core announced the General Availability of an upgraded message broker service that now includes support for the MQTT version 5 protocol (MQTTv5). With this release, customers can connect their devices to AWS IoT Core over MQTTv5, or leverage a mix of MQTTv3 or MQTTv5 connected vehicles interacting with one another to support heterogeneous deployments.

Working backward from automotive customers, our teams heard feedback on many features necessary to run and support vehicle workloads on AWS, and more specifically on AWS IoT Core. Many automotive customers adhere closely to standards and avoid having custom code from vendors in-vehicle. This provides the flexibility to be multi-cloud and switch brokers in case of ongoing issues. With IoT Core now adhering to the latest MQTTv5 standard, this allows automotive customers flexibility to align to an industry standard like MQTT with their in-vehicle architectures and code, but provides the flexibility to pivot to any number of different cloud message brokers depending on the implementation and customer requirements. Now that AWS IoT Core has adopted MQTTv5, OEMs see the long-term value in IoT Core as a managed broker.

Automotive customers who already have MQTTv3.1.1 deployments can make use of the new MQTTv5 features as AWS IoT Core provides seamless integration between both versions and supports deployments during the migration process. In the next few sections, we will cover some MQTTv5 features with connected vehicle use cases to show how you can design more flexible and efficient IoT design patterns. We also show how MQTTv5 brings new possibilities for your existing vehicle fleet running AWS IoT Core.

User properties

One of the new features of MQTTv5, user properties introduced basic key-value pairs that developers can append to most MQTT packets in the header, providing a mechanism to add meta-data that can be used in downstream processing of the packet. A customer can use a near unlimited number of user properties to add metadata to MQTT messages and pass information between publisher, broker, and subscriber.

Utilizing an MQTT broker as a routing and transporting component in large scale data processing and streaming implementation is a well-documented use case. These types of deployments, especially in a connected vehicle vertical, frequently contain several different variations of either devices, firmware or both and so, it is not out of the ordinary for the requirement to be able to

handle payloads from different devices or software versions different ways. For example, to ensure backwards compatibility, utilizing the software version to determine which processor to use or to identify a specific payload to be used with real-time streaming and other identification mechanism to route for more analytical storage of the same data. In this case, implementing MQTTv5 user properties can function as a routing mechanism for encoded or compressed payloads without the need for visibility into the message to ensure it is routed appropriately. The MQTT broker can then, with very little overhead, by just inspecting the header of the packet, route certain messages to a specific set of subscribers, based on the value of the fields in user properties. This allows for even more flexible application-level logic for providing message relevance to the proper processors and end state of the telemetry.

Session expiry

When connecting to the broker, in the CONNECT packet, a connecting client can now set a *session expiry interval* in seconds. This interval defines the period of time that the MQTT broker stores the client's session information and allows the customer to define fixed intervals. For example, if we wanted to set the session expiry to 10 seconds and the client is disconnected, the connected vehicle's session in the AWS IoT Core MQTT broker will be removed along with queued messages. Even if the message expiry intervals allow queuing messages, they won't be received by the connected car since the session is removed after 10 seconds. This allows for more granular control over client connection mechanisms when combined with cloud-side device management of these parameters, more holistic control over the entire fleet.

In a similar context to the session expiry interval, the primary motivation to add the message expiry interval to the MQTT protocol standard was to implement the capability of automatic deletion of retained messages and additionally allowing more granular control over the payloads. Many connected vehicles are designed to be in a disconnected state for prolonged periods without an internet connection. For connected vehicles, retained messages are used to deliver critical payloads to the vehicle when it comes back online.

In the disconnected state, messages are retained on the cloud broker and then delivered when the vehicle comes back online and subscribes to the topic. With larger implementations, providing the customer control over the types of messages and how long they are retained on the broker is key to ensuring the message is delivered (or not delivered). For example, a message about a local weather alert or a vehicle accident along the route is only relevant for a limited time. However, ensuring the vehicle receives a message about a pending OTA firmware updates must not expire until the next OTA software update is available to consume.

Allowing customers to set an appropriate message expiry interval for payloads no longer need to be delivered within a certain amount of time and leaving critically-relevant messages without an expiry ensures that efficiencies, cost and the implementation is delivered appropriately.

Topic aliases

The topic aliases feature allows MQTT clients to assign numeric aliases to topics and then refer to the alias when publishing further messages. This allows reduction in the transmitted MQTT packet size by referencing the topic with a single number instead of the topic itself.

Most vehicles are running cellular devices in the telematics control unit (TCU) and use mobile networks to communicate with their back-end services. These TCUs are designed to operate on the lowest possible bandwidth because of their metered data services. Reducing the size of these transmission is key to sustaining limited power consumption, but more importantly saving on data egress costs on the MNO's network towards the cloud.

Request/Response

The request/response messaging pattern is a method to track responses to client requests in an asynchronous way. It's a mechanism implemented in MQTTv5 to allow the publisher to specify a topic for the response to be sent for a particular message. Therefore, when the subscriber receives the request, it also receives the topic to send the response. It also supports the correlation data field that allows tracking of packets, for example, request or device identification parameters. This will be reviewed in depth when remote commands are discussed later on in the document.

Broker modernization on AWS IoT Core

The broker modernization reference architecture provides some high-level guidance for table-stakes use cases, some of which might be already in use by the automaker, and some might be on the roadmap. Not all of these use cases have to be implemented, but we wanted to demonstrate the power of MQTTv5 on AWS IoT Core and repeatable design patterns with downstream AWS services. With the modernization approach, we make some basic assumptions in that the vehicle is (or can be) provisioned to AWS IoT Core using mTLS, the MQTT and cryptography libraries properly support the requirements to connect to AWS IoT Core. From there, all existing publish and subscribe mechanisms using MQTT will work with IoT Core, just the processing logic to handle the payloads will need to be setup to subscribe and publish to these topics from the rest of the cloud infrastructure.

At AWS re:Invent 2022, Mercedes Benz presented their broker modernization approach and how they migrated millions of vehicles to AWS IoT Core, to reduce the overall complexity of the message broker, a top-down initiative to move towards managed services, serverless components and overall cost reduction. For Mercedes, the publish/subscribe architecture brings better observability on a per-vehicle basis for troubleshooting, debug and trace techniques. With a streaming architecture using a broker, they can separate telemetry vs commands, quickly iterate on prototyping on production workloads and integrate more seamlessly with other downstream AWS services such as Amazon Kinesis.

With the broker modernization approach, vehicle manufacturers can begin their AWS IoT Core journey with a few simple steps, and have an immediate impact on their operationalization, observability, security, and scalability of their connected vehicle platforms. Eventually, the power of the AWS IoT Core ecosystem and the differentiation it brings will drive OEMs to extend their connected vehicle offering to implement other AWS managed services AWS IoT Core such as AWS IoT Device Management and AWS IoT Device Defender.

Broker replacement

The first step to the broker modernization approach is to evaluate the current message broker implementation and determine if this approach will work for the existing vehicles in the field. The preference for this approach is to leave the edge implementation static and repointing the edge end from the old MQTT broker in the current production implementation to the AWS IoT Core endpoint. If MQTT 3.1.1 specification is followed in the existing vehicles, then this is the recommended approach to take, minimizing (or even eliminating) any code updates at the edge.

To provision vehicles to IoT Core or to send telemetry with the service, an endpoint is necessary. Using IoT Core custom domain configurations, users can build configurable endpoints for AWS IoT Core. Enabling our customers to take their existing fully qualified domain names (FQDN) and create a traffic-based policy in Amazon Route 53 to resolve the location using geolocation or latency-based routing. If needed, a just-in-time Provisioning workflow (explained in detail later in the document) will allow auto-registration of your vehicles once they connect to AWS IoT Core or the devices could be provisioned in AWS IoT Core prior to the migration.

Replacing the broker in your connected vehicle architecture reduces licensing costs, operationalization support and resources and allows AWS to handle the undifferentiated heavy lift of managing your MQTT broker at scale. Recently, we introduced several MQTTv5 features that can help with specific automotive use cases, those are discussed below.

Shared subscriptions

Using shared subscriptions, incoming vehicle telemetry can be distributed to message processors in a more efficient manner. When higher than normal demand for telemetry processing exists, AWS IoT Core will randomly distribute the published payloads to a single subscriber spreading the message processing load across a larger set of subscribers enabling more of a load-balanced approach to ensure a single instance of a processor is not overwhelmed. If shared subscriptions is not enabled, each subscriber would get messages published to that topic, which creates redundant processing for the cloud.

This allows for custom processing of telemetry ingest by having, perhaps, a micro-service container-based layer to handle the topic subscriptions and process the message as they are received. We could potentially ramp up the number of processors to handle peak-demand. Each micro-service can read from the shared topic on a shared subscription. This will help in distributing the processing load amongst the application servers within a group. Without the shared subscription feature, any messages that gets post on the MQTT topic will be published to all the clients subscribing to the topic. With the shared subscription feature only one of the groups clients will receive the message on this topic.

Within the connected vehicle space, it is highly expedient for a peak-demand scenario where millions of vehicles publish messages to a common topic. Those millions of messages need to be processed and sent to backend systems for consumption which may be companion mobile applications that shows the driver trip data or perform pre-processing prior to storage in data lake, or a machine learning pipeline for predictive maintenance and many others.

Building connected vehicle platforms on AWS IoT Core

A strong basis for a connected vehicle platform that is built on AWS IoT Core is providing security of vehicle to cloud connectivity and securing the platform and the data sent to the platform to risks the connected vehicle platform. In this section, we will discuss several security best practices and dive deep into vehicle identity lifecycle.

Topics

Connected vehicle security

Vehicle to cloud network connectivity

Customers have several choices when connecting their vehicles to AWS. Each option presents trade-offs in areas like scalability and cost. Below we will illustrate the options with diagrams. In each diagram, we will assume traffic is flowing through a Mobile Network Operator (MNO).

Connectivity over the internet

With this option, each vehicle's Electric Control Unit (ECU) uses AWS IoT Core's public endpoint for communication. IoT Core's public endpoint will resolve to an AWS owned public IP address. With this deployment model, the bandwidth from each ECU is limited by the internet performance. Additionally, we recommend using an application-level encryption mechanism like TLS to encrypt your data in transit. You can also encrypt highly sensitive data client side, which we will illustrate below.

Connectivity via VPN over the internet

With this option, the ECU of each vehicle must establish a VPN connection into AWS. The VPN connection will allow applications running on ECUs to consume resources within Amazon Virtual Private Clouds (VPCs).

The ECUs can then communicate with IoT Core over VPC Endpoints. The VPC Endpoints will create elastic network interfaces within the VPCs.

With this deployment model, the throughput from each ECU is limited by internet performance, and the maximum available bandwidth for each IPSec tunnel.

There are additional charges for accessing data over interface endpoints.

Private connectivity over Direct Connect

This deployment model leverages both VPC endpoints and Direct Connect for providing access to AWS IoT Core. Traffic from all ECUs is first redirected to an on-premises data center.

Direct Connect is used to transfer data from on-premises data-centers into AWS. You can either use a private virtual interface (VIF) or a Transit VIF. With the Private VIF option, you can connect up to a maximum of 500 VPCs, and with Transit VIF you can connect to up to 5,000 Regional VPCs. However, with the Transit VIF option, you'll incur Transit Gateway data processing charges by the GB. Since Private VIF option doesn't use a Transit Gateway, there are no Transit Gateway data processing charges with the Private VIF option.

Since the AWS IoT Core endpoints will be deployed in only a few VPCs, we recommend using the Private VIF option as it's more cost effective.

There are additional charges for accessing data over interface endpoints.

Identity best practices on AWS IoT Core

AWS IoT allows client authentication with different types of device credentials. In most use cases, X.509 certificates are the recommended method of authenticating your ECU. Another method, custom authentication, should only be used in migration scenarios where none of the above options are available. You should **only** use IAM-user; credentials during research and development. If you need to make direct AWS API calls from the device, it is recommended you use the IoT Core Credential Provider. The credential provider authenticates a caller using an X.509 certificate and issues a temporary, limited permissions security token. The token can be used to sign and authenticate any AWS request.

Each device should have a unique X.509 certificate, and identities should not be shared across ECUs. ECUs must use TLS version 1.3 when connecting to AWS IoT Core for enhanced security and performance. Customers can use AWS IoT Device Defender Audit Checks for a list of comprehensive device security posture checks.

A vehicle has several connected ECUs, and an ECU may have more than one identity depending on back-end interaction requirements. It is important to tie these identities together using an asset store mapped to different ECUs using something like an ECU ID. You can use AWS IoT Device Management which provides a built-in registry for Things and their registered X.509 certificates. The registry allows you to define attributes (three per thing) which are name-value pairs you can use to store information about the thing, such as ECU ID. Each certificate registered in AWS IoT Core can be associated with an AWS IoT Core Policy (either directly or via Thing Groups) that authorizes actions that the ECU can perform on the AWS IoT Core service such as allowing connections, publishing or subscribing to certain MQTT topics. IoT Core also allows you to register certificates the first time an ECU connects using Just-in-time-Registration by registering the Certificate Authority (CA) that issued the certificate.

Least privilege is the practice of only granting access that identities, in this case ECUs need to perform the intended function. It is important to discuss some anti-patterns and best practice guidance for granting least privilege to ECUs in AWS IoT Core. Common anti-patterns include:

- Granting broad permission by assigning "*" to actions or resources. A "*" on action will allow the device any data plane operation. A "*" on resources will authorize any resource to conduct the policy action.
- Avoid using hardcoded values like client ID, and instead use characteristics of things such as `ThingName`, `ThingNameType`, `Thing Attributes`, or certificate attributes such as `Subject`, `Issuer`, and `Subject Alternate Name`.

Continuous monitoring of your policies is an important mechanism to ensure that overly permissive policies are addressed. It is recommended to review your AWS IoT Device Defender Audit checks related to overly permissive and misconfigured ECUs, and create a response/remediation strategy when these ECUs are detected. You can send your AWS IoT Device Defender Audit findings to AWS Security Hub, which is a cloud security posture management service that performs security best practice checks, aggregates alerts, and enables automated remediation.

For more information, see the Identity checklist in the AWS Well-Architected IoT Lens.

Identity onboarding and lifecycle management

A secure connection is required between vehicles and the cloud. The recommendation from AWS is to leverage X.509 certificates for the mutual TLS connection. We will utilize this section to dive into the process of provisioning as well as the lifecycle management of certificates. Also, we will extend

on several options to get insights into the security posture of your devices and best practices for connectivity as well as data transmission.

In order to describe a possible flow for onboarding as well as lifecycle management of the certificates, we will base the further discussion in this sub section on a few assumptions:

- The customer has the offline root certificate authority (CA) on premises: With most customers in the automotive space when migrating their connected vehicle platform to AWS or building it from scratch, vehicle PKI is established with an offline Root CA on-premises. As it is pre-existing and already fully being managed, the preference often is to keep it as is. If the option is available to build the root CA on AWS and want to use a managed service for implementation, AWS provides AWS Private CA for that purpose.
- The customer is hosting a certificate broker API, which is leveraged for interfacing with the Subordinate CA to issue certificates and transmit those over to the device. We often see customers having existing infrastructure on-premises that they don't want to or can't move to the cloud for several reasons. This certificate broker can also be hosted on AWS using a combination of services like Amazon API Gateway and AWS Lambda.
- The customer already has an attestation (or bootstrap) certificate on the vehicle gateway. The flow demonstrated will only cover how to transmit the operational certificate used for the mTLS connection to AWS IoT Core.

Onboarding and provisioning

The following figure demonstrates a flow through the typical setup leveraging a variety of AWS services and features.

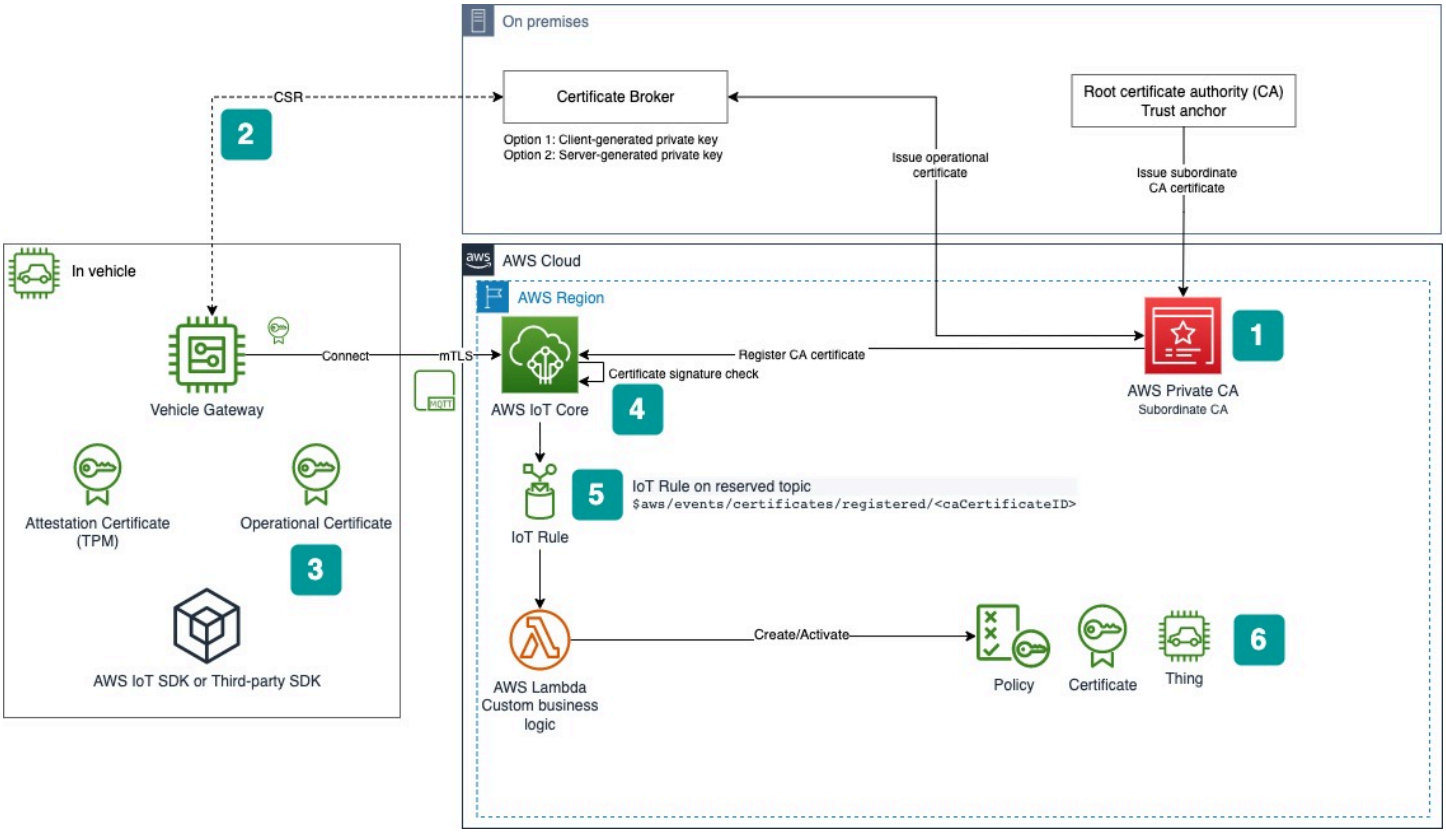

Figure – Provisioning mTLS certificates for the vehicle gateway

1/ Initial activity as preparation for the flow

AWS Private CA will be used for your Subordinate CA, which will issue all the Operational Certificates for your fleet. A required one-time activity is to issue the Subordinate CA certificate from your root CA, which you are hosting on-premises.

> **Note**
>
> AWS Private CA is not required in this instance, it's provided as an example implementation where AWS provides a service to assist this process. The customer can provide their own private CA infrastructure running on AWS compute or even a CA external to their AWS account.

AWS IoT Core will use the JITR flow for registering certificates and creating the needed resources in AWS IoT Core. The issuing CA needs to be registered with AWS IoT Core in order as a prerequisite. Also auto registration must be enabled for the CA certificate which will allow the service to register

operational certificates with the status of `PENDING_ACTIVATION` on first connect of the vehicle gateway if those have been signed by the CA.

2/ Issuing the certificate and transmitting it to the vehicle gateway

We see two options for issuing certificates and transmitting them onto the device.

Option 1 Client-generated private key: There is a secure connection in place between the private on-premises certificate authority and the vehicle gateway. The vehicle gateway generates a CSR, which it sends to the certificate broker securely via an mTLS connection using the attestation certificate identity. The certificate broker then calls the IssueCertificate API from the Subordinate CA to issue a client certificate and sends the operational certificate back to the device over the secure channel.

Option 2 Server-generated private key: This approach is used if the device is not generating the certificate request. A non-automated process might be directly working with the certificate broker to request several operational certificates. The certificate broker uses the IssueCertificate API from the Subordinate CA and communicates the operational certificates back to the certificate broker. There is typically a well-defined process to store the certificate and private key into a secure element within the vehicle gateway SOC.

3/ Storing the operational certificate on the vehicle gateway

Once the vehicle gateway receives the operational certificate it should store the private key and other secrets in a specialized protected module like a software or hardware based HSM. The operational certificate should be kept safe on the device, and in the next section we will discuss additional certificate lifecycle concepts like certificate rotation.

4/ Connecting to AWS IoT Core for the first time

From here on out we will discuss the approach of JITR for provisioning the certificates. The first step consists of using the operational certificate the device received during or after manufacturing during the first connect request. The first connect request to AWS IoT Core will not succeed, as the certificate isn't a registered and active certificate with the service. Since auto registration is enabled with the service and if the presented certificate was signed by the Subordinate CA, the certificate will be registered pending activation. You need to have automatic re-connection programmed into the device.

5/ AWS IoT Core publishes to reserved topic and subsequently invokes a rule

Every time AWS IoT Core registers a certificate with the status of `PENDING_ACTIVATION` it will publish a message to the reserved events topic `$aws/events/certificates/registered/caCertificateId`, where *`caCertificateId`* is the ID of the CA Certificate that signed the operational certificate. The flow now requires an AWS IoT rule to subscribe to exactly that topic and act on any message published. You should configure that rule to invoke an AWS Lambda function with the registration event to perform custom logic prior to granting access and creating the necessary resources.

6/ Perform business logic and create resources in the AWS Lambda function

The AWS Lambda function (registration Lambda) allows the customer to perform required custom business logic to check for the validity of the request. This is an important step to ensure that only trusted devices get access to AWS IoT Core resources. Customers often have an existing database with information about all manufactured devices that are expected to communicate with the cloud at some point, which they leverage as an additional security mechanism in the function. Also, you can do your Online Certificate Status Protocol (OCSP) checks here before you activate any certificate. After the validity of the request is confirmed, registration Lambda would create an IoT Thing, an IoT Policy (the best practice is to use templated policies with fine granular access to a selected list of topics and resources) and activate the certificate.

This concludes the flow to provision a vehicle gateway in AWS IoT Core. In the next section we will elaborate on rotating operational certificates.

Lifecycle management

In the automotive space, often attestation as well as operational certificates are issued for the lifetime of a vehicle. This would map to one certificate from the time a vehicle was manufactured to when it is decommissioned. But there are multiple use cases that require an option to rotate a certificate. The following use cases can be seen as a few examples:

- **Vehicle owner changes**: Within the latest enterprise-grade connected vehicle platforms gather significant amounts of data that is aligned to the vehicle owner as well as the certificate leveraged for connection and communication. A vehicle typically changes its owner every 6 years. The best practice to provide true separation between the previous owner and the new one is to also reprovision the vehicle with a new certificate.
- **Certificate compromised**: The certificate should be securely stored on the vehicle. In spite of all taken security measures, we recommend customers to plan for worst case scenario. So, for

the case that a certificate is compromised the customer should build in the option to revoke any existing permissions given to a vehicle based on the certificate and also have a process that allows you to issue a new certificate to the device.

- **Vehicle decommissioning**: At the end of the vehicle's lifetime, not only the hardware, but also the communication channel to AWS IoT Core should be decommissioned. For AWS IoT Core the communication is based on a valid operational certificate, which you should deactivate to avoid any malicious behavior or inaccurate data reported by a non-functioning vehicle.

The guidance here would be to issue operational certificates for a limited scope for expiration and have a rotation process in place that is based on a trusted connection to the device, the possibility to rotate the certificate as well as the private key on the hardware with a process that is invoked and tracked from the cloud.

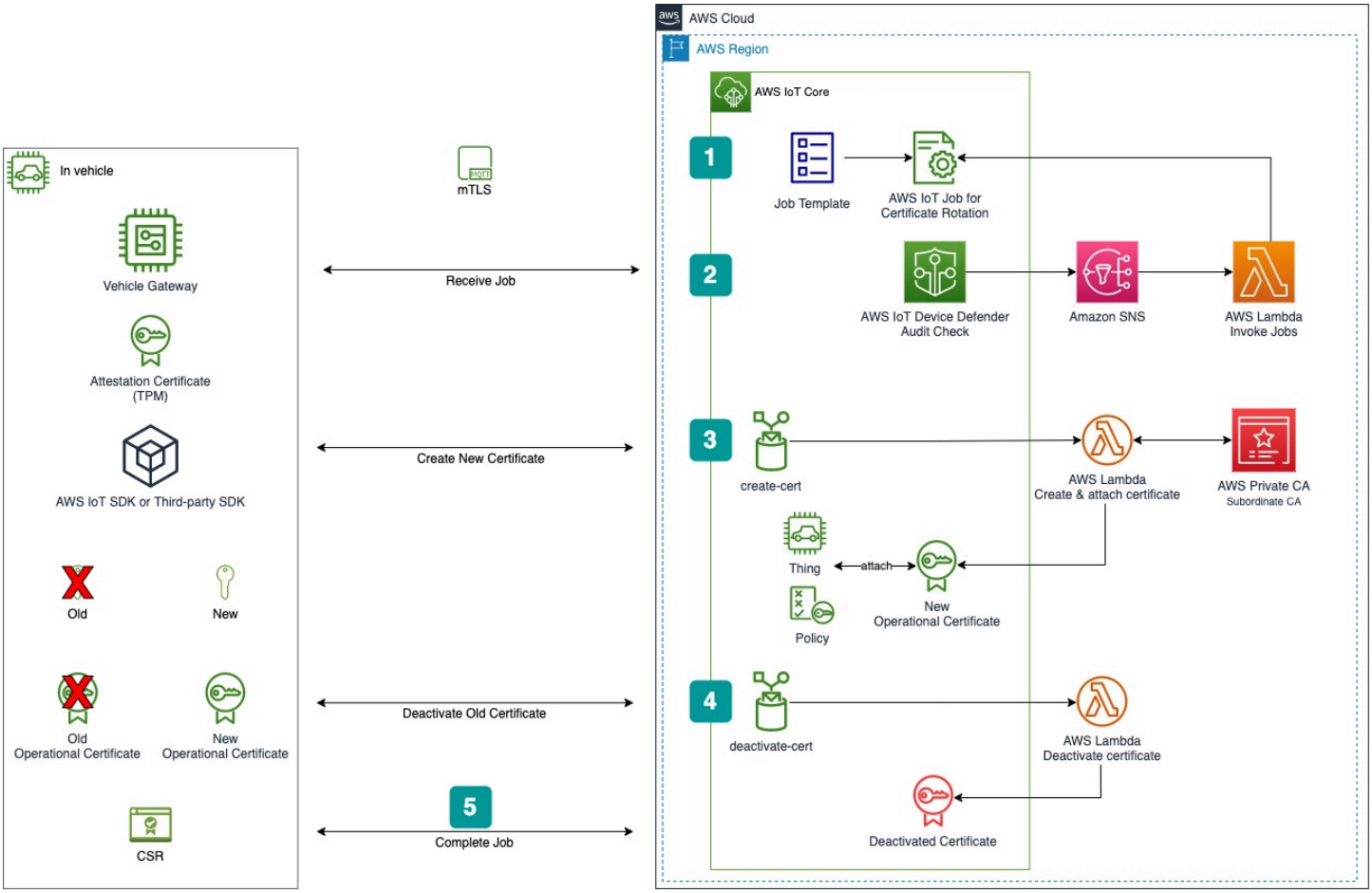

Figure - Rotate operational certificates on the vehicle

Set up AWS IoT Jobs for invoking and tracking certificate rotation

In the above, we are leveraging AWS IoT Jobs as a foundation for certificate rotation as it provides you with a defined set of remote operations that you can send to and run on one or more devices. For jobs with standard configurations, AWS IoT job templates are recommended as they allow you to deploy a preconfigured job to multiple sets of target devices. In this case, the procedure on device side should be standardized for every certificate rotation process and the configuration like aborting criteria, timeouts and retries will be configured once for the whole fleet.

This paper is not going to attempt to cover the device side as this is different implementation dependent on the silicon, device manufacturer or software provider. The device needs to follow the job workflow.

AWS IoT Device Defender Audit Checks for invoking rotation

To allow for automatic detection of the necessity to rotate certificates, the recommendation is to add the features of Audit Checks of AWS IoT Device Defender. For this use case we leverage the two Audit Checks `REVOKED_DEVICE_CERTIFICATE_STILL_ACTIVE_CHECK` and `DEVICE_CERTIFICATE_EXPIRING_CHECK`. The first one will check and return any certificates that are on its CA's certificate revocation list (CRL) and the second one will return those that are already expiring or will expiry within 30 days. AWS IoT Device Defender can then be configured to raise an alarm, which is coupled with an Amazon SNS notification. The customer can utilize the notification to invoke an AWS Lambda function, that then starts the AWS IoT Jobs workflow.

Create the new operational certificate

After the device receives the job following the job workflow, the device will send over a MQTT message to AWS IoT Core using its existing secured connection with the command to create a new certificate. The AWS IoT rules engine should be configured to invoke an AWS Lambda function, which then interacts with the Subordinate CA to issue a new operational certificate. The newly issued certificate will be delivered back to the device using MQTT messages. The Lambda function should also attach the new certificate to the existing IoT Thing and attach the existing IoT policy to it.

Deactivating the expiring certificate

After the device receives the certificate, it needs to also rotate the private key. If you don't have the option to rotate the private key on the device, you can't do certificate rotation.

Next step is to send the message to deactivate the expiring certificate to the cloud. Again, the AWS IoT rules engine needs to invoke a Lambda function on that message which will then deactivate the certificate in AWS IoT Core. Deactivating a certificate also leads to a loss of permissions of the policy attached to the certificate.

Complete the job

After all the previous steps are completed, a device should also complete (update the Job Execution with SUCCEEDED) the job by sending the according message to the broker.

Compliance

Next generation vehicles require compliance that spans both onboard and offboard the vehicle. Customers need to consider relevant global and regional standards and regulations that span across the vehicle ecosystem. There are several automotive standards and regulations such as ISO 21434 for cybersecurity, ISO 26262 for functional safety, ASPICE for software development lifecycle process, ISO 24089 for OTA, and regulations like UNR 155/156 for developing a cyber security management system and software update management system in automotive. As mentioned above, consider and align to general standards when building connected mobility applications like ISO 27001, NIST Cybersecurity Framework, and privacy laws like GDPR. This helps to ensure that the systems in scope meet the highest levels of safety, security, and privacy. Our Compliance Center is a central location to research cloud-related regulatory requirements and how they impact your industry.

The UNR 155 regulation mentioned above has provisions that requires the ability to detect, prevent, monitor vehicle vulnerabilities, attacks, and threats. These requirements have influence solutions like a vehicle security operations center (VSOC) and vulnerability management systems for automotive. In the monitoring section, we will cover some ways that AWS IoT, other services and partners can help customers address some of these requirements.

Global implementation for connected vehicles

When building a connected vehicle architecture on AWS, customers need to be aware that most of the AWS services are regional services. When it comes to high availability (HA) and durability, many services already are, or can provide, support for building highly available solutions within a Region (like AWS IoT Core, or Amazon S3).

We see customers with the need to deploy workloads in multiple regions as the vehicles will also be distributed globally to ensure regional latency is minimized between the vehicle and the cloud.

This scope of this whitepaper is not to provide a generalized solution to align for all business use cases, but will dive into a few considerations and explain the patterns we see repeated between OEMs.

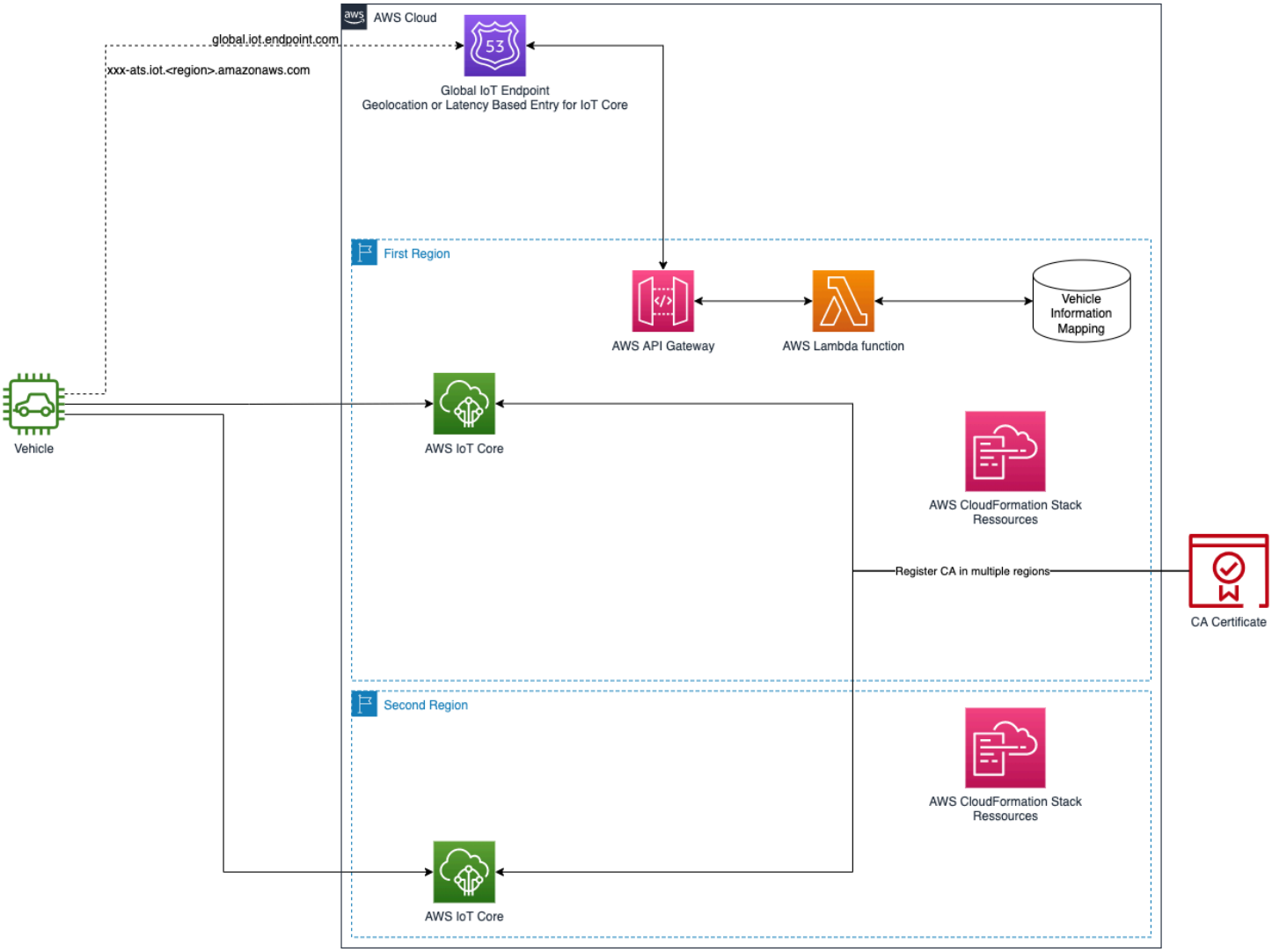

Figure - Global implementation for connected vehicles

Global endpoints

Each regional AWS IoT Core endpoint is reachable either by the individual service endpoints defined by AWS or the customer is also able to create configurable endpoints to connect vehicle gateways.

> **Note**
>
> Using a configurable endpoint over the default AWS IoT Core endpoint is highly recommended to be stay flexible throughout the vehicle lifetime.

Either way, for the vehicle devices the customer will want to minimize a static configuration on the vehicle to stay flexible throughout the lifetime of the vehicle; this flexibility should also cover the endpoint that the vehicle connects to for communicating with the backend systems. With this reason in mind the customer should leverage Amazon Route 53, the globally highly available DNS web service provided by AWS, or any other DNS Server as the global endpoint that will manage the routing logic for the vehicles.

Amazon Route 53 offers geolocation or latency based routing logic to direct the vehicle to the right AWS IoT Core endpoint. How this can be configured is described in detail in this blog post.

Additionally, to the endpoint and routing logic, customers implement a vehicle registry table with the vehicle information which contains the existing regional endpoint for the vehicle gateway. The vehicle registry table is normally initially populated with the device information during manufacturing time prior to connecting to AWS IoT Core. This table can be coupled with the Amazon Route 53 setup by adding an Amazon API gateway as interface and then additionally use Amazon Lambda for the compute logic which interacts with the table.

Depending to your business requirements, the customer might want to add logic to the device itself that allows it to reconnect to AWS Route 53 on location change (which also might be needed for some countries where specific or all data is not allowed to be stored or transmitted depending on country requirements). This approach is normally limited in scope to smaller OEMs and mobility providers.

Static regional endpoint

Slightly larger OEMs, with limited regional opportunities for sales markets, are choosing the market at the end of the production line, knowing that vehicle gateway will be pointed to the single region for the lifetime of the vehicle gateway. This simplifies the process for the OEMs to have a single workflow in the device-VIN pairing process to burn into the vehicle gateway the IoT Core endpoint.

> **Note**
>
> This endpoint can still be redirected to other AWS IoT Core endpoints or other cloud provider if necessary, as the DNS lookup in Amazon Route 53 or the DNS server of your choosing can be modified, but would be limited to a single endpoint change across the fleet, rather than reconfiguring a single vehicle to point to a new endpoint.

Regional vehicle provisioning process using AWS IoT Core JITR

Since the customer might not know which Region a device will connect to throughout its lifetime, the implementation will need to leverage a provisioning flow that allows the connection for every region the connected vehicle platform will support.

From a best practices perspective, the AWS team recommends the just-in-time registration (JITR) flow as the best practice in the connected vehicle space providing the most flexibility and extensibility for regional vehicle provisioning (this is discussed in the previous section of Onboarding and Provisioning). For JITR, the customer will first need to import the certificate of the Subordinate/signing CA (either managed on-premises or by AWS Private CA) to AWS IoT Core. There are two modes to register CA certificates: DEFAULT and SNI_ONLY. If the certificate is registered in different regions, those can all be in DEFAULT mode. Within a region (and then multiple accounts) only one CA can be registered in DEFAULT mode and the others need to be registered in SNI_ONLY mode. The CA will need to be registered in every account and region, in which the vehicles could eventually connect to, namely where your connected vehicle platforms exist.

The implementation/platform will deploy all other resources required for JITR (Lambdas, Private CA, etc.) using AWS CloudFormation (or the preferred infrastructure-as-code solution). This allows the customer to also make this part of your version control solution in place, track any changes (also allow for roll backs) and provide the exact same functionality across all required regions.

Replication of data between Regions

Another consideration for a global location is the reconfiguration of endpoints during the lifetime of a vehicle. An example would be that a person moves from Europe to the United States and also moves the vehicle that they own.

The simplest solution is to determine the minimum data and configuration necessary to mirror over to new region. The vehicle needs to reconnect to the global endpoint on Region change

(which needs to be implemented on vehicle side) and the AWS Lambda function can then check the vehicle registry table to incorporate the regional reconfiguration of locations and clean up and replicate anything in the old region to the new region if necessary.

Customers often switch to this approach, which is the simplest solution, after they first propose to keep all regions in sync. This would firstly be very cost intensive and not only impact AWS IoT Core, its data and configuration but also any downstream environments (examples are, but not limited to, ETL solutions, Machine Learning models, Analytics and Databases). A few of the AWS services allow for multi-region data synchronization like Amazon Dynamo DB or Amazon S3. You also need to keep the data end user in mind and our recommendation is to only take the hard requirements into account for replicating not only storage but also transfer and transformation. Also be aware that some data might not be allowed to leave a region and be synchronized to a different one based on country laws. For setting up this logic of replication Figure 3 mentions AWS CloudFormation stack resources which you deploy into the regions needed and facilitate the logic needed on cloud side.

Remote commands for companion applications

For connected vehicle platforms, sending telemetry is one of the more important use cases, but secondary to that, especially as a customer differentiator is the ability to manage the vehicle remotely. Enabling these remote functions will unlock several new features for customers and fleet operators. These use cases are being demanded by the end users, but OEMs have encountered several challenges in meeting those expectations from their customers.

One of the challenges OEMs encounter is the end-to-end latency around downstream communication and commands with the vehicle. The ability to remote start, utilize a shared digital key, and a find-your-car feature are differentiated value adds to the customer experience in the next generation of vehicles. This is a difficult use case to accomplish for developers when there is limited, spotty or, a majority of the time, no connectivity (because the vehicle is not running) to the cloud platform from the vehicle to send a command.

Currently, to implement remote commands when the vehicle is disconnected, to the vehicle most OEMs employ a shoulder tap architecture, where sending an SMS message to the device wakes it up and have it check the command queue for processing. The mobile terminated (MT-SMS) that initiates the internet protocol (IP) data session and is supported by current LTE modems that is a low-power or sleep mode which reduces the drain on the battery, but comes at a cost for reliability and speed.

SMS is generally a reliable mechanism for exchanging messages, but can be troublesome if the cellular connectivity is reduced and without the ability to ACK the receipt of the SMS, produces a poor customer experience, especially in the unlock door use case. Customers have reported anywhere from 10-35 seconds for a remote command initiated by SMS to complete the end-to-end flow from companion application, to the cloud, to the vehicle, and to report success back to the customer.

With the evolution of cellular technologies this problem becomes less ubiquitous and the OEM can modify the protocols used to receive the remote commands. With LTE Cat M1 or Cat 1 and extended discontinuous reception (eDRX) the device gains the ability in the vehicle to consume up to 10x less battery, a 50% reduction in module cost as compared to Cat 4 devices, and a lower MHz channel that allows for better penetration into parking garages. This requires widespread utilization of CAT M1 within the vehicle networks and has not been widely adopted by OEMs.

The most common approach to solving the latency problem with remote commands is keep the MQTT connection to the broker open after the vehicle has been turned off, allowing for remote commands to be delivered to the vehicle immediately. In previous model years, keeping the broker connection open when the ignition is off, the battery drain would impact the ability to start the vehicle and so the customer experience suffered. with the evolution of the electric vehicle space and much higher capacity of vehicle batteries,

OEMs are now enabled create a longer broker connection time upon ignition off. Many OEMs are switching to a longer Keep Alive, sometimes extending it up to seven days after ignition off. The Keep Alive is configured at the start of the MQTT session and requires a PINGREQ from the client and a PINGRESP from the broker ensuring the connection is still open. As these pings keep the connection open, remote commands are received from the native application in near real time and improves the overall customer experience drastically.

MQTTv5 Request/Response

Using some of the latest features of IoT Core and MQTTv5 can help enhance some of the more crucial customer experiences. Below, in Figure 4, we have a reference architecture that provides high-level guidance around developing a companion application for remote commands on AWS IoT. This implementation provides some basic patterns as described above to implement an architecture that could help provide a much better customer experience when attempting to execute a remote command. The guidance provides several scenarios for managing vehicles remotely, whether the vehicles are connected to the broker or disconnected. AWS IoT Lifecycle

events help provide the connection status to the AWS AppSync middleware layer with various subscriptions and resolvers to the persistency state.

After determining if the vehicle is connected, the backend can use the new request/response pattern (defined in MQTTv5 section) and send commands to the vehicle in the request topic which is paired with a response topic. When a command is executed (successfully or not) the vehicle will then post the result on the response topic which will then be sent back to the front-end application via AWS AppSync custom resolver. When the device is not connected, we provide a mechanism to send the SMS-shoulder tap via Amazon Simple Notification Service (SNS).

Consumer identity

You can build and operate your own identity provider or use a cloud SaaS identity provider. Amazon Cognito is a fully managed identity provider that allows you to add user sign-up and sign-in features and control access to your web and mobile applications. Amazon Cognito user pools provides an identity store that scales to millions of users, supports social and enterprise identity federation, and offers advanced security features to protect your consumers and business. You can implement Device Authorization grant using Cognito and AWS Lambda. AWS services such as Application Load Balancer (ALB), API Gateway, AWS AppSync, and AWS IoT Core can authenticate and authorize requests using tokens issued by Cognito User Pools.

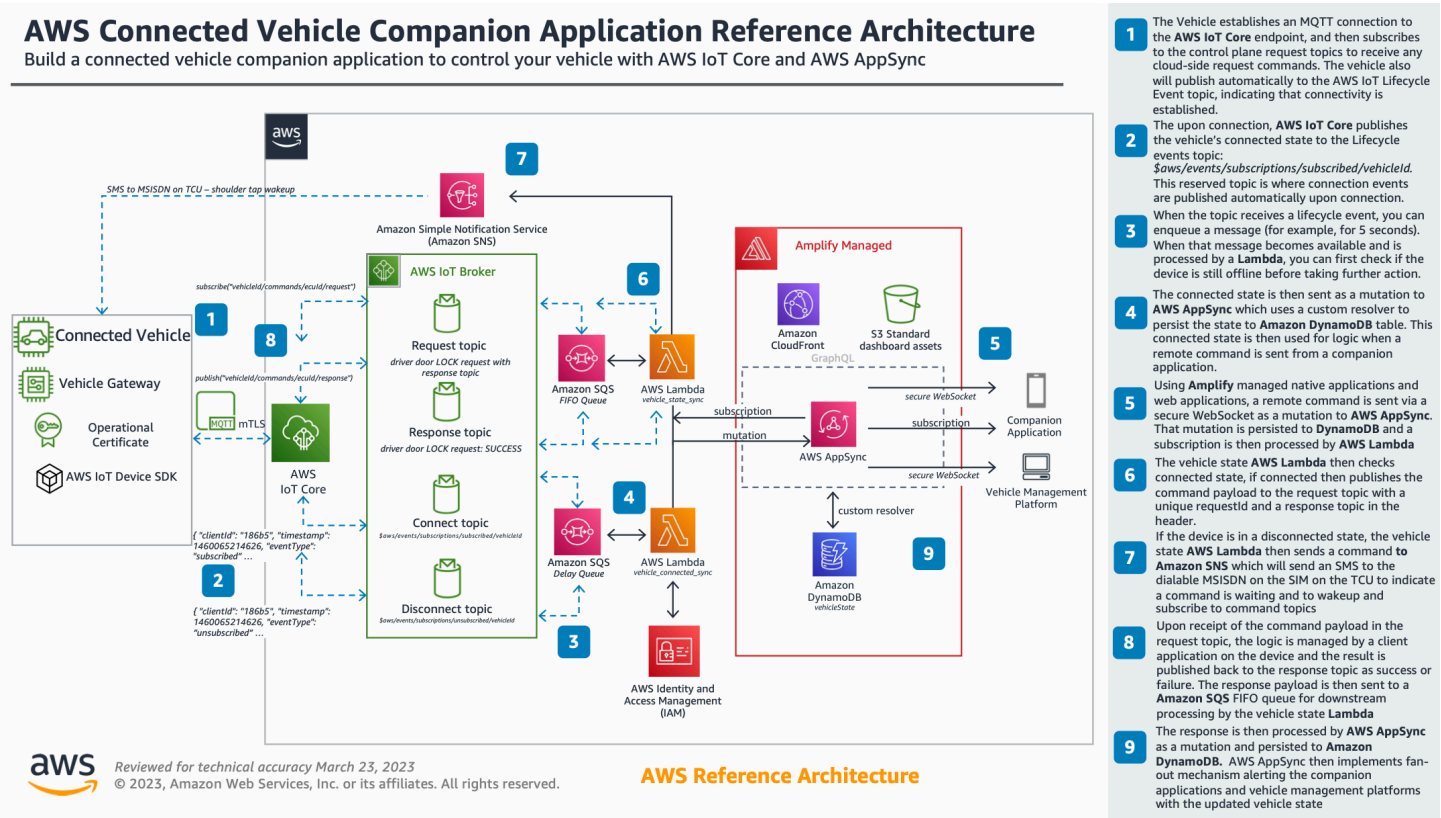

Figure : AWS connected vehicle companion application reference architecture

AWS IoT Device Shadow service

The AWS IoT Device Shadow service is a key feature set for AWS IoT remote command and control. The device shadow makes it possible for the developer to decouple the companion application and the vehicle architecture, passing messages between the two systems using JSON documents. A vehicle's shadow document is stored in AWS Cloud to persist the current state for the vehicle.

With the device shadow, applications can interact with the vehicles even when they are not connected. This provides the following features:

- Cloud representation of vehicle state
- Last known state query for offline vehicles
- Real-time vehicle state changes
- Command and control via a change in state

The state of the vehicle is persisted in the cloud and completely customizable to fit the OEM's trackable attributes. For example, if the window's state (up/down) can be remotely controlled via the companion application for a specific trim of a vehicle, but not for other trims, the windows attribute can be removed from the JSON shadow definition document for the trims that do not support that feature. This allows for complete flexibility as the OEM builds its state mechanism, but at the same time, Vehicle Shadow adds more cost to the overall solution and might not meet the flexibility required by the OEM.

Remote command approach

When building a connected vehicle platform, many decisions will need to be made on aligning to a specification or aligning to cloud services. For remote commands, AWS offers a solution to both approaches. With the request/response feature of MQTTv5, builders can implement remote command functionality, regardless of the underlying cloud provider. This approach will require more upfront work, and more operational overhead as connectivity state will need to be monitored and stored along with the existing state of the parameter being changed.

With AWS IoT Device Shadow services, the implementation requires much less up-front work, but will have impacts on cost and aligns the platform closer to AWS IoT Core specific features which will not translate to other cloud providers. AWS works backwards from our customers to

provide these flexible building blocks to ensure platform requirements are met, either reduction in operational overhead or being cloud agnostic.

Intelligent data collection with AWS IoT FleetWise

For most automotive companies, the primary motivation for a connected vehicle platform is to provide a mechanism to retrieve data off the vehicle and monetize the data for downstream services. Therefore, a common pattern exists where this data from the vehicle needs to be collected, normalized and aggregated to the cloud. With AWS IoT FleetWise, the undifferentiated heavy lift of building a data management platform of your connected vehicle is removed and another building block to the overall connected vehicle platform on AWS is delivered.

Most vehicle manufacturers have, in some form or another, collected telemetry data from vehicles over the past several years to help diagnose potential issues, identify preventative maintenance assistance and potential recalls. These automotive companies are beginning to shift towards building vehicles with more advanced sensors that generate orders of magnitude of larger data volume —with LIDAR and camera data, this can mean up to 2 terabytes of data every hour.

With these vast amounts of data now being generated by the vehicle, automotive companies need access to this data in the cloud to derive insights that can help improve vehicle quality, safety, and autonomy. As mentioned, transferring this data off the vehicle to the cloud can be complicated and expensive. Additionally, with the multitude of added rich data sensors in vehicles that generate data in different proprietary formats create a complex array of data across vehicles. Collecting this data in an efficient and cost-effective manner requires a custom-built in-vehicle data-collection system, which, for automotive companies, can be a difficult task. As a result, these automotive companies over index on building data management capabilities, rather than focusing on their own differentiators to allow their data scientists deliver insights and create new experiences for users in a highly performant manner.

With AWS IoT FleetWise the undifferentiated heavy lift of building this data collection platform is removed. These challenges of collecting vehicle data are now performed by a fully managed service that customers can use to collect, transform, and transfer vehicle data to the cloud in near real time. With AWS IoT FleetWise, automotive companies can now collect and organize data from vehicles with differing protocols and proprietary data formats. AWS IoT FleetWise helps to transform CAN and OBD telemetry binary frames into human-readable data and then standardizes that data into a vehicle model in the cloud for data analyses. Vehicle manufacturers can then define different data collection campaigns to remotely determine which vehicle data to collect and how frequently to transfer that data to the cloud.

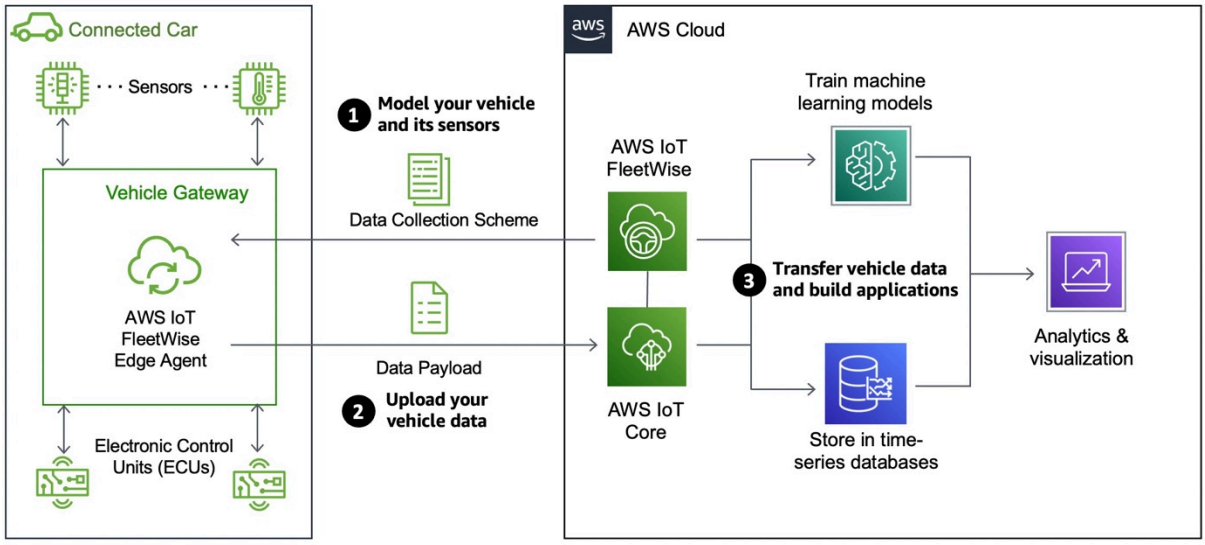

Figure: AWS IoT FleetWise high-level architecture

Figure : AWS IoT FleetWise - high-level architecture

Data modeling

AWS IoT FleetWise provides a vehicle model orchestrator that automotive companies can use to build digital twins of their vehicles in the cloud. Vehicle signals, signal catalogs, vehicle models, and decoder manifests are the core components that help deliver data from the vehicle to the cloud efficiently and effectively.

Signals

Signals are fundamental structures that customer utilize to define to contain vehicle data and its metadata. A signal can be an attribute, a branch, a sensor, or an actuator. For example, an automotive OEM can create a signal to receive in-vehicle temperature values, and to store its metadata, including a sensor name, a data type, and a unit.

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.

Vehicle model

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.

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.

Data collection

Once the vehicle has been modeled, and the signal catalog has been created, the customers are now able to create data collection campaigns using signals created within the model.

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.

All campaigns are created in the cloud. After the campaigns have been marked as approved by team members, then AWS IoT FleetWise automatically deploys them to vehicles. Automotive teams can choose to deploy a campaign to a specific vehicle or a fleet of vehicles. The Edge Agent software will not start collecting data of the vehicle network until a running campaign is deployed to the vehicle.

Data protection considerations

Next generation vehicle communication requires robust encryption mechanisms. There are different internal and external requirements based on threat models that inform your data protection decisions. For encryption in transit, AWS IoT Core supports TLS 1.2 and 1.3. AWS IoT Core also provides security policy options that include several different ciphers. AWS IoT Core allows you to select TLS security policies. You can choose a predefined policy that supports the TLS protocols and ciphers that meet your requirements.

Customers might want to encrypt sensitive data client-side before sending it to the cloud, or before sending data to the vehicle. AWS Key Management Service (AWS KMS) lets you create,

manage, and control cryptographic keys across your applications and AWS services. On AWS, you can use the AWS KMS to securely manage your encryption keys for envelope encryption. You can use the AWS Encryption SDK to implement envelope encryption with data key caching on the ECU and backend servers to improve performance, help reduce cost, and stay within the AWS KMS service quotas as your application scales. Your ECUs can obtain temporary credentials to invoke AWS KMS API calls by using AWS IoT Core credential provider.

OEMs collect a significant amount of data from the vehicle. This can include consumer data such as driving behavior, insurance carriers, PII (for example, name and email address), VIN or ECU IDs, and navigation services. AWS IoT Core and AWS IoT FleetWise provide the ability to store data centrally in Amazon S3, as described later in this whitepaper.

One challenge OEMs face is identifying sensitive data coming from the vehicle to determine the types of sensitive data stored in the backend. You can use Amazon Macie to discover and help protect your sensitive data. Macie uses a combination of criteria and techniques, including machine learning (ML) and pattern matching, to detect sensitive data. Macie can detect a large and growing list of sensitive data types for many countries and regions, including multiple types of credentials data, financial data, personal health information (PHI), and personally identifiable information (PII). VINs can be detected using a managed data identifier. You also can build custom data identifiers using regular expressions (regex) to match vehicle-specific identifiers such as ECU IDs, and ECU serial numbers.

Data analytics

Once the campaigns have been executed in the vehicle, the destination of your data is determined by the campaign setup. For near-real time analytics and visualization dashboards of your data, Amazon Timestream would be the selected destination for telemetry data. When looking to create a performant data lake, centralized data storage and data processing pipelines, AWS IoT FleetWise offers storage in Amazon S3 with Apache Parquet or JSON data formats.

With flexible data storage options using AWS IoT FleetWise, automotive companies can customize their usage of AWS in their connected vehicle platform to collect the data as they see fit; some data needs to be pulled real time for vehicle tracking use cases, other data can be batched and stored for further processing to fulfill predicative maintenance use cases where data can be loaded into machine learning (ML) models to help predict issues within the fleet before they happen.

Using IoT Device Management and IoT Device Defender in automotive workloads

One key function of a connected vehicle platform is to ensure the operationalization of the fleet of vehicles. Whether the ask is to organize and group vehicle fleets into flexible hierarchies to streamline maintenance by make/model/year or to ensure the in-vehicle firmware is up-to-date, AWS IoT Core Device Management offers capabilities to ensure long-standing operations of a fleet of vehicles is handled with ease. From securing and monitoring device fleet health status and enablers to analyze trends, observability and push updates at scale.

When a vehicle is onboarded to AWS IoT Core, all other services within the AWS IoT ecosystem are available to implement if the customer can derive value from those services. These services are not required to be used as part of a connected vehicle platform, but add differentiators to the long-term operationalization of the fleet of vehicles.

AWS IoT Jobs

One key aspect of the software defined vehicle, is the capability of perform device software updates, over-the-air (OTA). With AWS IoT Jobs, it provides a framework to send a set of remote operations over MQTT to be run on remote devices connected to AWS IoT. For example, a customer can define an IoT Job that instructs a set of ECUs to download and install an application, run a firmware update, reboot the ECU or TCU, rotate certificates and perform operational steps such as remote troubleshooting session. AWS IoT Jobs is a simple framework to enable job documents and will still require an edge implementation to listen to job topics for remote commands and job documents.

Lifecycle events

Ensuring the current status of each vehicle is key for workflows such as remote commands. With AWS IoT Core, lifecycle events help enable storing the presence of the vehicle, either connected to AWS IoT Core or in a disconnected state. Each time a device connects or disconnects a payload is sent to a topic (`$aws/events/presence/connected/clientId`) which can then be used to create workflows aligned to these events. An example payload is as follows:

```
{
    "clientId": "186b5",
```

```
    "timestamp": 1573002230757,
    "eventType": "connected",
    "sessionIdentifier": "a4666d2a7d844ae4ac5d7b38c9cb7967",
    "principalIdentifier": "12345678901234567890123456789012",
    "ipAddress": "192.0.2.0",
    "versionNumber": 0
}
```

Most design patterns around lifecycle events store the connection status in DynamoDB or some even use the AWS IoT Device Shadow service to persist the current connection state.

Vehicle security monitoring and response

Monitoring and responding to vehicle events

Regulations such as the UNECE Regulation 155 and standards such as ISO 21434 require that vehicles are monitored throughout the lifecycle. Monitoring vehicle threats goes beyond technology and requires organizational strategy, people, and processes to do so successfully. AWS can help provide services that provide insights from data collected from the vehicle and vehicle ecosystem (for example, charging stations, and companion applications).

Customers can use AWS IoT Core to analyze relevant security data in the cloud. Customers can send data to services that can analyze data for security purposes using AWS IoT Rules. For example, you can send telemetry data, CAN data, or other types of data via an AWS IoT Rule to Amazon OpenSearch Service. From OpenSearch Service you can configure rules that you want to alert on for anomalous behavior coming from your ECUs. This can be telemetry data that is anomalous like successive door openings, or ECU related logs that may indicate an ECU has an issue.

AWS IoT Device Defender is a downstream service of AWS IoT Core which provides additional security services that allows the customer to audit the configuration of vehicles, monitor connected vehicles to detect abnormal behavior, and mitigate security risks. This can be fed to AWS Security Hub and to your vehicle security operation center which can provide detection, runbooks, and remediation mechanisms across the fleet using AWS services. For more information on a detect and response architecture and example, see the connected vehicle security reference architecture.

OEMs need to configure their ECUs for least privilege behaviors. Not every ECU will not the same permissions to the same resources. Without a policy engine, it is difficult to implement least privilege. AWS IoT Device Defender addresses these challenges by providing tools to identify security issues and deviations from best practices. AWS IoT Device Defender can audit device

fleets to ensure they adhere to security best practices such as overly permissive devices and detect abnormal behavior on devices.

Conclusion

With AWS IoT Core and its supporting ecosystem, OEMs can begin to envision the next generation's vehicle communication platform. This whitepaper has discussed some best practices for ways OEMs can implement some existing and future use cases utilizing the technologies that AWS IoT provides. With AWS IoT Core running as a managed service, automatically scaling to meet the demand, allows the OEMs to focus on their value add, rather than undifferentiated heavy lifting of infrastructure management. In addition to a serverless construct, this paper discussed the MQTT specification and how to best apply it to a vehicle architecture by using its feature set to match the use cases as necessary. With the publish/subscribe design pattern, the ability to easy exchange commands with the vehicle and ingest telemetry data allows for a platform that can be extended to use cases in and out of the vehicle.

Working alongside AWS IoT Core, there are several additional IoT specific services that provide added value for the OEM to manage ongoing operational objectives with their vehicles. AWS provides a comprehensive set of tools and services for the automotive industry. IoT Jobs provides an integrated OTA support, IoT Device Defender device security and certificate management, and Fleet Hub allow the OEM to manage all vehicles seamlessly through the console. In addition to those tools and services, the true value add for OEMs is the downstream services that seamlessly connect with AWS IoT Core, such as Amazon DynamoDB, Amazon Kinesis Data Streams and Amazon S3.

As this paper has demonstrated, using AWS IoT Core for some (or all) of the vehicle-to-cloud communication framework would provide the OEM with a secure, extensible and reliable platform that can support the use cases customers are demanding in today’s vehicles.

Contributors

Contributors to this document include:

- Andrew Givens, Senior IoT Specialist, SA, Automotive, Amazon Web Services
- Katja-Maja Kroedel, Senior EMEA IoT Specialist, SA, Automotive, Amazon Web Services
- Omar Zoma, Senior Security Solutions Architect Automotive, Amazon Web Services
- Maitreya Ranganath, Principal Security SA, Automotive, Amazon Web Services
- Lowry Snow, Principal GTM Specialist, Automotive, Amazon Web Services

Further reading

For additional information, see:

- AWS Connected Vehicle Reference Architecture
- Building and Modernizing Connected Vehicle platforms with AWS IoT
- Securing modern Connected Vehicle platforms with AWS IoT
- AWS IoT FleetWise object storage in Amazon S3
- Transforming fleet telematics into predictive analytics with Capgemini's Trusted Vehicle and AWS IoT FleetWise
- Connected Mobility on AWS
- How Reply Built a Connected Vehicle Platform with AWS IoT and Amazon Alexa
- People Tech Group Enables Digital Twin for Infotainment by Leveraging AWS IoT

Document history

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Minor update	Corrected the omission of data protection considera tions.	February 7, 2024
Major update	Updated to reflect new MQTT v5 support. Security and provisioning sections added, plus numerous updates throughout.	January 12, 2024
Initial publication	Whitepaper first published.	September 22, 2021

Note

To subscribe to RSS updates, you must have an RSS plug-in enabled for the browser that you are using.

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.

Notices

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