AWS Well-Architected Framework

Internet of Things (IoT) Lens

Internet of Things (IoT) Lens: AWS Well-Architected Framework

Table of Contents

Internet of Things (IoT) Lens

This whitepaper describes the AWS IoT Lens for the AWS Well-Architected Framework, which you can use to review and improve your cloud-based architectures and better understand the business impact of your design decisions. This document describes general design principles, as well as specific best practices and guidance for five of the six pillars of the Well-Architected Framework.

Publication date: **July 2, 2025** (*Document revisions*)

Introduction

The AWS Well-Architected Framework helps you understand the pros and cons of the decisions you make when building systems on AWS. Using the Framework allows you to learn architectural best practices for designing and operating reliable, secure, efficient, and cost-effective systems in the cloud. The Framework provides a way for you to consistently measure your architectures against best practices and identify areas for improvement. We believe that having well-architected systems greatly increases the likelihood of business success.

In this Lens, we focus on how to design, deploy, and architect your Internet of Things (IoT) workloads at the edge and in the AWS Cloud. The guidance provided includes both IoT and industrial IoT (IIoT) workloads and the document calls out specific guidance for segments such as consumer, commercial and industrial when relevant. To implement a well-architected IoT application, follow the well-architected principles, starting from the procurement of connected physical assets (things), operating the asset to the eventual decommissioning of those same assets in a secure, reliable, scalable, sustainable and automated fashion. In addition to AWS Cloud best practices, this document also articulates the impact, considerations, and recommendations for connecting physical assets to the internet.

This document only covers IoT specific workload details from the Well-Architected Framework. We recommend that you read the AWS Well-Architected Framework whitepaper and consider the best practices and questions for other lenses.

This document is intended for those in technology roles, such as chief technology officers (CTOs), architects, developers, embedded engineers, security and operations team members. After reading this document, you will understand AWS best practices and strategies for IoT and IIoT applications.

Lens availability

The IoT Lens is available as an AWS-official lens in the Lens Catalog of the AWS Well-Architected Tool.

To get started, follow the steps in Adding a lens to a workload and select the **IoT Lens**.

Definitions

The AWS Well-Architected Framework is based on six pillars — operational excellence, security, reliability, performance efficiency, cost optimization and sustainability. When architecting technology solutions, you must make informed trade-offs between pillars based upon your business context. For IoT workloads, AWS provides multiple services that allow you to design robust architectures for your applications. Internet of Things (IoT) applications are composed of many devices (or things) that securely connect and interact with complementary edge-based and cloud-based components to deliver business value. IoT applications gather, process, analyze, and act on data generated by connected devices. Industrial Internet of Things (IIoT) are systems that connect and integrates industrial control systems with enterprise systems and the internet, business processes and analytics and is a key enabler for Smart Manufacturing and Industry 4.0. The AWS IoT Lens can be used across all IoT use cases including industrial, consumer (OEM), or and any other workload that has many devices connecting at scale for telemetry and command and control.

This section presents an overview of the AWS components that are used throughout this document to architect IoT workloads. There are seven distinct logical layers to consider when building an IoT workload:

- Design and manufacturing layer
- Edge layer
- Fleet provisioning layer
- Communication layer
- Ingestion layer
- Analytics layer
- Application layer

Design and manufacturing layer

The design and manufacturing layer consists of product conceptualization, business and technical requirements gathering, prototyping, product layout and design, component sourcing, manufacturing and distribution. Decisions made in each layer affects the next logical layers of the IoT workload.

For example, some IoT device creators prefer to have a common firmware image installed and tested by the contract manufacturer. The decisions made at the design and manufacturing layer will partly determine what steps are required during the provisioning layer.

You may go a step further and provision and install an X.509 certificate and its private key to each device during manufacturing, or include a hardware security module with credentials already pre-provisioned. This decision can affect the provisioning and communications layers, since the type of credential can influence the subsequent selection of network protocols. If the credential is long-lived, it can simplify communications and provisioning layers but could increase security risks from unintended exposure of these credentials.

Edge layer

The edge layer of your IoT workload consists of the physical hardware of your devices, the embedded operating system that manages the processes on your device, and the device firmware, which is the software and instructions programmed onto your IoT devices. The edge is responsible for sensing and acting on other peripheral devices. Common use cases are reading sensors connected to an edge device, or changing the state of a peripheral based on a user action, such as turning on a light when a motion sensor is activated.

While the AWS IoT Lens is applicable to all IoT systems, industrial IoT deployments often have additional safety, resiliency and compliance requirements in addition to the standard well-architected guidance.

Industrial IoT deployments consist of a combination of plant-local Operational Technology (OT), plant-local Information Technology (IT) resources, and remote IT resources which may be in the public cloud or an enterprise datacenter. The benefit of splitting workloads between local and remote processing is to balance the timeliness and high bandwidth of local resources with the scale and elasticity of remote resources.

The edge deployments are heavily influenced by what AWS calls the three laws of distributed computing:

- **Law of physics**, which constrain the latency, throughput and availability of network connectivity.
- **Law of economics** which determine the cost-effectiveness of transferring ever-increasing volumes of data.

Law of the land which regulate how data is handled and where it can be stored.

AWS offers the following software and services for the edge layer:

AWS IoT device SDKs include open-source libraries, developer guides with samples, and porting guides so that you can build innovative IoT products or solutions with AWS IoT on your choice of hardware systems.

FreeRTOS is a real time operating system for microcontrollers that lets you program small, low-power, edge devices while leveraging memory-efficient, secure, embedded libraries.

AWS IoT Greengrass is an IoT edge runtime and cloud service that helps customers build, deploy, and manage intelligent IoT device software. It provides developers with pre-built components for common capabilities, such as local/cloud MQTT messaging, support for local edge processing including Machine Learning (ML) inference, logging, monitoring, out-of-the-box integration with AWS services, and local data aggregation, filtering, and transmission to cloud targets. Once development is complete, customers can seamlessly deploy and remotely manage device software on millions of devices.

AWS IoT SiteWise Edge provides software that runs on premises at industrial sites and makes it straightforward to collect, process, and monitor equipment data locally before sending the data to AWS Cloud destinations. AWS IoT SiteWise Edge software can be installed on local hardware such as third-party industrial gateways and computers, or on AWS Outposts and AWS Snow Family compute devices. It uses AWS IoT Greengrass, an edge runtime that helps build, deploy, and manage applications.

AWS IoT FleetWise Edge is the edge software component for AWS IoT FleetWise. AWS IoT FleetWise Edge allows connected vehicles to collect data and upload it to the AWS IoT FleetWise service. 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 schemes to control what data to collect in vehicles and when to transfer it to the cloud.

AWS IoT ExpressLink is connectivity software that powers a range of hardware modules developed and offered by AWS partners, such as Espressif, Telit, Realtek and u-blox. Integrating these wireless modules into the hardware design of your device makes it faster and easier to build Internet of Things (IoT) products that connect securely with AWS services. These modules provide cloud-connectivity and implement AWS-mandated security requirements.

Fleet provisioning layer

The provisioning layer of your IoT workloads consists of mechanisms used to create device identities and the application workflow that provides configuration data to the device. In many cases, it consists of a Public Key Infrastructure (PKI). The provisioning layer is also involved with ongoing maintenance and eventual decommissioning of devices over time. IoT applications need a robust and automated provisioning layer so that devices can be added and managed by your IoT application in a frictionless way. When you provision IoT devices, you must install a unique cryptographic credential onto them and securely store these credentials.

By using **X.509 certificates,** you can implement a provisioning layer that securely creates a trusted identity for your device that can be used to authenticate and authorize against your communication layer. X.509 certificates are issued by a trusted entity called a certificate authority (CA). While X.509 certificates do consume resources on constrained devices due to memory and processing requirements, they are an ideal identity mechanism due to their operational scalability and widespread support by standard network protocols.

The **AWS IoT Device Registry** helps you manage and operate your things. A thing is a representation of a specific device or logical entity in the cloud. Things can also have custom defined static attributes that help you identify, categorize, and search for your assets once deployed.

AWS Private Certificate Authority (AWS Private CA) helps you automate the process of managing the lifecycle of private certificates for IoT devices using APIs. Private certificates, such as X.509 certificates, provide a secure way to give a device an identity that can be created during provisioning and used to identify and authorize device permissions against your IoT application.

AWS IoT Just-in-Time-Registration (JITR) enables you to programmatically register devices to be used with managed IoT systems such as AWS IoT Core. With JITR, when devices are first connected to your AWS IoT Core endpoint, you can automatically trigger a workflow that can determine the validity of the certificate identity and determine what permissions it should be granted.

Provisioning devices that do not have certificates: With AWS IoT fleet provisioning, AWS IoT can generate and securely deliver device certificates and private keys to your devices when they connect to AWS IoT for the first time. AWS IoT provides client certificates that are signed by the Amazon Root certificate authority (CA). There are two ways to use fleet provisioning:

- **Provisioning by claim**: Devices can be manufactured with a provisioning claim certificate and private key (which are special purpose credentials) embedded in them. If these certificates are registered with AWS IoT, the service can exchange them for unique device certificates that the device can use for regular operations.
- **Provisioning by trusted user**: A device connects to AWS IoT for the first time when a trusted user, such as an end user or installation technician, uses a mobile app to configure the device in its deployed location.

Communication layer

The communication layer handles the connectivity, message routing among remote devices, and routing between devices and the cloud. The communication layer lets you establish how IoT messages are sent and received by devices, and how devices represent and store their physical state in the cloud.

AWS IoT Core helps you build IoT applications by providing a managed message broker that supports the use of the MQTT protocol to publish and subscribe IoT messages between devices.

With the **AWS IoT Device Shadow service Service**, you can create a data store that contains the current state of a particular device. The Device Shadow Service maintains a virtual representation of each of your devices you connect to AWS IoT as a distinct device shadow. Each device's shadow is uniquely identified by the name of the corresponding thing.

AWS IoT Core for LoRaWAN is a fully managed LoRaWAN Network Server (LNS) that enables customers to connect wireless devices that use the LoRaWAN protocol for low-power, long-range wide area network connectivity with the AWS Cloud. This can be useful in use cases such as asset tracking, irrigation management, logistics and transportation management and smart cities.

With **Amazon API Gateway**, your IoT applications can make HTTP requests to control your IoT devices. IoT applications require API interfaces for internal systems, such as dashboards for remote technicians, and external systems, such as a home consumer mobile application. With Amazon API Gateway, you can create common API interfaces without provisioning and managing the underlying infrastructure.

Ingestion layer

A key business driver for IoT is the ability to aggregate the disparate data streams created by your devices and transmit the data to your IoT application in a secure and reliable manner. The

ingestion layer plays a key role in collecting device data while decoupling the flow of data with the communication between devices.

With **AWS IoT rules engine**, you can build IoT applications such that your devices can interact with AWS services. AWS IoT rules are analyzed and actions are performed based on the topic a message is received on.

Basic Ingest can securely send device data to the AWS services supported by AWS IoT rule actions, without incurring messaging costs AWS IoT Core pricing. Basic Ingest optimizes data flow by removing the publish or subscribe message broker from the ingestion path, making it more cost effective.

Using **AWS IoT Greengrass**, data can be ingested in S3 buckets, Firehose, AWS IoT SiteWise, AWS IoT Analytics, and with custom code to other AWS services.

AWS IoT SiteWise is a managed service that simplifies collecting, organizing, and analyzing industrial equipment data at scale to help you make better, data-driven decisions. You can use AWS IoT SiteWise to monitor operations across facilities, quickly compute common industrial performance metrics, and create applications that analyze industrial equipment data.

AWS IoT FleetWise is a managed service that makes it straightforward to collect, organize, and transfer vehicle data to the cloud so you can gain insights about your fleet(s) of vehicles. You can use data transferred to build applications that quickly detect fleet-wide quality issues, remotely diagnose individual vehicle problems in near real-time, and improve autonomous driving systems.

Amazon Kinesis and **Amazon Simple Queue Service (Amazon SQS)** could be used in your IoT application to decouple the communication layer from your application layer. Amazon Kinesis is a managed service for streaming data, enabling you to get timely insights and react quickly to new information from IoT devices. Amazon Kinesis integrates directly with the AWS IoT rules engine, creating a seamless way of bridging from a lightweight device protocol of a device using MQTT with your internal IoT applications that use other protocols. Amazon SQS enables an event-driven, scalable ingestion queue when your application needs to process IoT applications once where message order is not required.

Analytics layer

One of the benefits of implementing IoT solutions is the ability to gain deep insights from data about what is happening in the local or edge environment. A primary way of realizing contextual insights is by implementing solutions that can process and perform analytics on IoT data.

Storage services

IoT workloads are often designed to generate large quantities of data. Make sure that this discrete data is transmitted, processed, and consumed securely, while being stored durably.

Amazon S3 is object-based storage engineered to store and retrieve data from anywhere on the internet. With Amazon S3, you can build IoT applications that store large amounts of data for a variety of purposes: regulatory, business evolution, metrics, longitudinal studies, analytics, security, machine learning, and organizational enablement. Amazon S3 gives you a broad range of flexibility in the way you manage data for cost optimization, latency, access control and compliance.

Analytics and machine learning services

After your IoT data reaches a central storage location, you can begin to unlock the full value of IoT by implementing analytics and machine learning on device behavior. With analytics systems, you can begin to operationalize improvements in your device firmware, as well as your edge and cloud logic, by making data-driven decisions based on your analysis. With analytics and machine learning, IoT systems can implement proactive strategies like predictive maintenance or anomaly detection to improve the efficiencies of the system.

AWS IoT Events is a managed service that makes it straightforward to detect and respond to events from IoT sensors and applications. Events are patterns of data identifying more complicated circumstances than expected.

AWS IoT SiteWise is a managed service that simplifies collecting, organizing, and analyzing industrial equipment data at scale to help you make better, data-driven decisions. You can use AWS IoT SiteWise to monitor operations across facilities, quickly compute common industrial performance metrics such as overall equipment effectiveness (OEE), and create applications that analyze industrial equipment data.

AWS IoT SiteWise allows you to create no-code, fully managed web applications using AWS IoT SiteWise Monitor. With this feature, you can visualize and interact with operational data from devices and equipment connected to AWS IoT services.

Amazon Athena is an interactive query service that makes it straightforward to analyze data in Amazon S3 using standard SQL. Amazon Athena is serverless, so there is no infrastructure to manage, and customers pay only for the queries that they run.

Amazon SageMaker AI AI is a fully managed service that enables you to quickly build, train, and deploy machine learning models in the cloud and the edge layer. With Amazon SageMaker AI AI, IoT architectures can develop a model of historical device telemetry in order to infer future behavior. Through the integration of AWS IoT Greengrass and Amazon SageMaker AI AI, customers can automate the full ML lifecycle of collecting IoT data, ML training in the cloud, deploying ML models to the edge for local inference, and then retraining and redeploying in a cycle for continuous improvement of their ML models.

AWS IoT TwinMaker is an AWS IoT service that you can use to build operational digital twins of physical and digital systems. AWS IoT TwinMaker creates digital visualizations using measurements and analysis from a variety of real-world sensors, cameras, and enterprise applications to help you keep track of your physical factory, building, or industrial plant. You can use this real-world data to monitor operations, diagnose and correct errors, and optimize operations.

Amazon Managed Grafana is a fully managed service for open source Grafana developed in collaboration with Grafana Labs. Grafana is a popular open-source analytics solution that enables you to query, visualize, alert on and understand your metrics no matter where they are stored. Grafana has integrations with services like AWS IoT Twinmaker to make visualizations easier.

QuickSight is a cloud-scale business intelligence (BI) service that you can use to deliver straightforward insights to the people who you work with, wherever they are. QuickSight connects to your data in the cloud and combines data from many different sources from the IoT suite.

Application layer

AWS IoT provides several ways to ease the way cloud native applications consume data generated by IoT devices. These connected capabilities include features from serverless computing, fit for purpose database technologies such as time series databases to create materialized views of your IoT data, and management applications to operate, inspect, secure, and manage your IoT operations.

Management applications

The purpose of management applications is to create scalable ways to operate your devices once they are deployed in the field. Common operational tasks such as inspecting the connectivity state of a device, making sure device credentials are configured correctly, and querying devices based on their current state must be in place before launch so that your system has the required visibility to troubleshoot applications.

AWS IoT Device Defender is a fully managed service that audits your device fleets, detects abnormal device behavior, alerts you to security issues, and helps you investigate and mitigate commonly encountered IoT security issues.

AWS IoT Device Management eases the organizing, monitoring, and managing of IoT devices at scale. At scale, customers are managing fleets of devices across multiple physical locations. AWS IoT Device Management enables you to group devices for easier management. You can also enable real-time search indexing against the current state of your devices through Device Management Fleet Indexing. Both Device Groups and Fleet Indexing can be used with Over the Air Updates (OTA) when determining which target devices must be updated to target specific sub-fleets of devices when customers want to deploy remote operations (for example, remote reboots, Over-the-air (OTA) updates, configuration pushes, and resets) using Jobs. Customers can also gain privileged and synchronous access (for example, SSH) to their devices for debugging and troubleshooting with Secure Tunneling.

User applications

In addition to managed applications, other internal and external systems need different segments of your IoT data for building different applications. To support end-consumer views, business operational dashboards, and the other net-new applications you build over time, you will need several other technologies that can receive the required information from your connectivity and ingestion layer and format them to be used by other systems.

Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Apple, Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0 and OpenID Connect.

Database services

While a data lake can function as a landing zone for your unformatted IoT generated data, to support the formatted views on top of your IoT data, you need to complement your data lake with structured and semi structured data stores. For these purposes, you should leverage both NoSQL and SQL databases. These types of databases enable you to create different views of your IoT data for distinct end users of your application.

Amazon DynamoDB is a fast and flexible NoSQL database service for IoT data. With IoT applications, customers often require flexible data models with reliable performance and automatic scaling of throughput capacity.

With **Amazon Aurora** your IoT architecture can store structured data in a performant and cost-effective open-source database. When your data needs to be accessible to other IoT applications for predefined SQL queries, relational databases provide you another mechanism for decoupling the device stream of the ingestion layer from your eventual business applications, which need to act on discrete segments of your data.

Amazon Timestream is a fast, scalable, and serverless time series database service for IoT and operational applications that makes it straightforward to store and analyze trillions of events per day. Amazon Timestream's purpose-built query engine lets you access and analyze recent and historical data together, without needing to specify explicitly in the query whether the data resides in the in-memory or cost-optimized tier. Amazon Timestream has built-in time series analytics functions, helping you identify trends and patterns in your data in near real-time.

Compute services

Frequently, IoT workloads require application code to be executed when the data is generated, ingested, or consumed/realized. Regardless of when compute code needs to be executed, serverless compute is a highly cost-effective choice. Serverless compute can be leveraged from the edge to the core and from core to applications and analytics.

AWS Lambda allows you to run code without provisioning or managing servers. Due to the scale of ingestion for IoT workloads, AWS Lambda is an ideal fit for running stateless, event-driven IoT applications on a managed system.

Design principles

The Well-Architected Framework identifies the following set of design principles in order to facilitate good design in the cloud with IoT:

- **Decouple ingestion from processing**: In IoT applications, the ingestion layer must be a highly scalable system that can handle a high rate of streaming device data. By decoupling the fast rate of ingestion from the processing portion of your application through the use of queues, buffers, and messaging services, your IoT application can scale elastically as needed and make several decisions without impacting devices, such as the frequency it processes data or the type of data it is interested in.
- **Design for offline behavior**: Due to things like connectivity issues or misconfigured settings, devices may go offline for longer periods of time than anticipated. Design your edge software to handle extended periods of offline connectivity and create metrics in the cloud to track devices that are not connected or communicating on a regular timeframe.
- **Design for lean data at the edge and enrich in the cloud**: Given the constrained nature of IoT devices, the initial device schema will be optimized for storage on the physical device and efficient transmissions from the device to your IoT application. For this reason, unformatted device data will often not be enriched with static application information that can be inferred from the cloud. As data is ingested into your application, you should first enrich the data with human readable attributes, deserialize, or expand any fields that the device serialized, and then format the data in a data store that is tuned to support your applications read requirements.
- **Handle personalization**: Devices that connect to the edge or cloud through Wi-Fi must receive the SSID name and credentials as one of the first steps performed when setting up the device. This data is usually infeasible to write to the device during manufacturing since it is sensitive and site-specific or from the cloud since the device is not connected yet. These factors frequently make personalization data distinct from the device client certificate and private key, which are conceptually upstream, and from cloud-provided firmware and configuration updates, which are conceptually downstream. Supporting personalization can impact design and manufacturing, since it may mean that the device itself requires a user interface for direct data input, or the need to provide a smartphone application to connect the device to the local network.
- **Make sure that devices regularly send status checks**: Even if devices are regularly offline for extended periods of time, make sure that the device firmware contains application logic that sets a regular interval to send device status information to your IoT application. Devices must be active participants in making sure that your application has the right level of visibility. Sending

this regularly occurring IoT message make sures that your IoT application gets an updated view of the overall status of a device, and can create processes when a device does not communicate within its expected period of time.

- **Use Gateways for edge computing, network segmentation, security compliance and bridging administrative domains:** By splitting the workload between local and remote processing helps to balance the timeliness and high bandwidth of local resources with the scale and elasticity of remote resources. Edge gateways can be used to mediate data flows between a low latency local-area network (LAN) and resources on the high-latency wide-area network (WAN), protocols used in each environment and can also mediate between security and administrative domains such as in a Perdue Enterprise Network Architecture (PERA), ANSI/ISA-95 network segmentation. Edge gateways are also used in consumer IoT systems. For example, a smart home gateway which collects data from multiple smart home devices.
- **Build security into your IoT solution and apply security at all layers:** IoT implementations can have some unique considerations not present in traditional IT deployments. For example, deploying a consumer IoT device can introduce a new classification of threats that needs to be addressed and industrial IoT requires more thought around reliability, safety and compliance. Many legacy OT systems have limited security features and use industrial protocols which does not support authentication, authorization and encryption. In industrial IoT, the convergence of IT and OT systems is creating a mix of technologies that were designed to withstand risky network environments and ones that were not, which creates risk management difficulties that need to be controlled. So, building security into every part of your IoT solution is essential for minimizing risks to your data, business assets, and reputation. Apply a defense in-depth approach with multiple security controls.

Scenarios

This section addresses common scenarios related to IoT applications, with a focus on how each scenario impacts the architecture of your IoT workload. These examples are not exhaustive, but they encompass common patterns in IoT applications. We present a background on each scenario, general considerations for the design of the system, and a reference architecture of how the scenario should be implemented.

Scenarios

- Device provisioning
- Device telemetry
- Device commands
- IoT Edge computing
- Generative AI and IoT

Device provisioning

In IoT, device provisioning is composed of sequential steps. The most important outcome is that each device must be given a unique identity and authenticated by your IoT application using that identity.

The first step to provisioning a device is to install an identity. The decisions you make in device design and manufacturing determines if the device has a production-ready firmware image and unique client credential by the time it reaches the customer. Your decisions determine whether there are additional provisioning-time steps that must be performed before a production device identity can be installed.

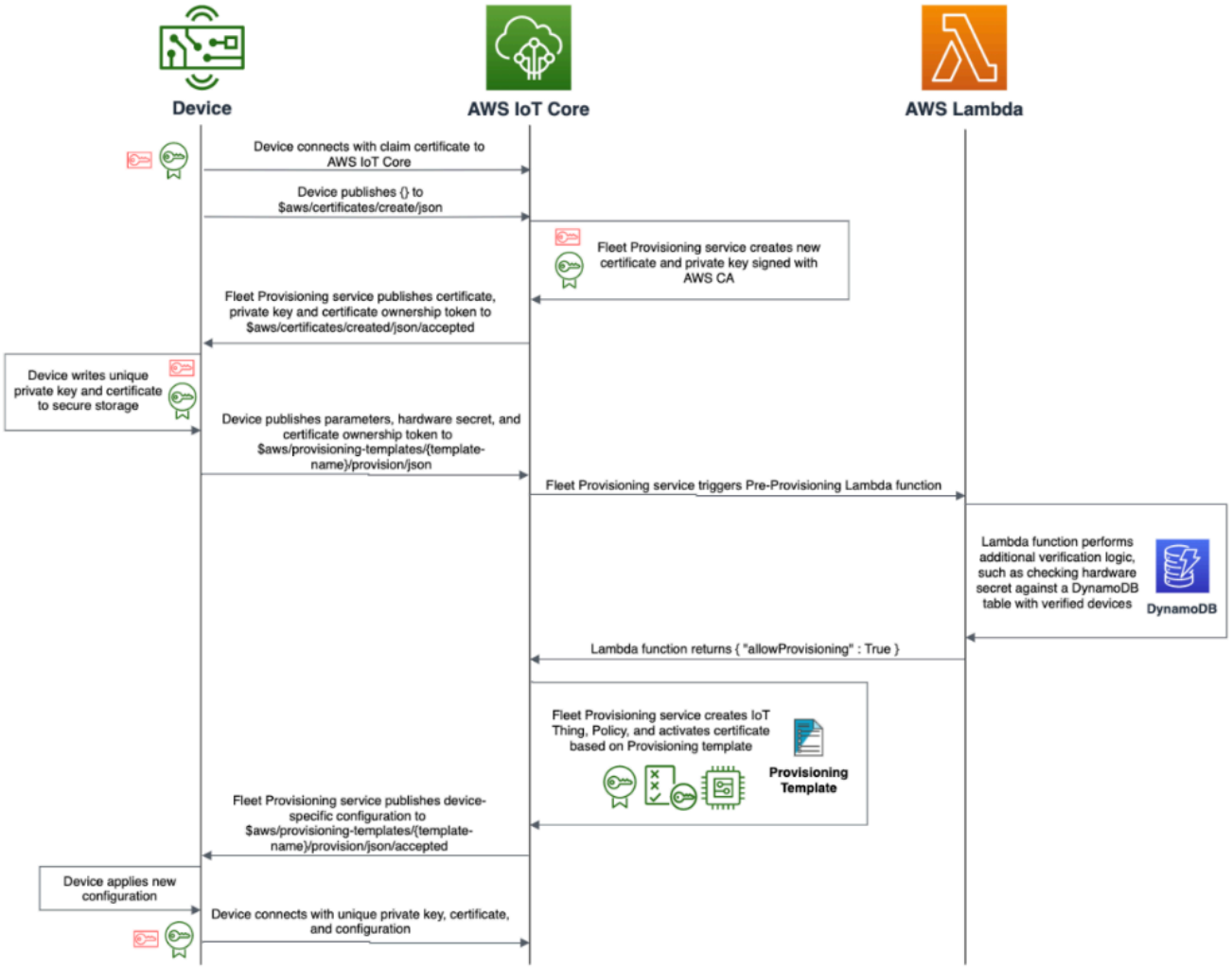
Use X.509 client certificates for your IoT devices — they are more secure and easier to manage at scale than static passwords. In AWS IoT Core, the device is registered using its certificate along with a unique thing identifier. The registered device is associated with an IoT policy. An IoT policy allows you to create fine-grained permissions per device. Fine-grained permissions make sure that only the device has permissions to interact with the right MQTT topics and messages.

The registration process makes sure that a device is recognized as an IoT asset and that the data it generates can be consumed through AWS IoT to the rest of the AWS landscape. One of the ways to

provision a device, is through Fleet Provisioning. AWS IoT can generate and securely deliver device certificates and private keys to your devices when they connect to AWS IoT for the first time. AWS IoT provides client certificates that are signed by the Amazon Root certificate authority (CA). Fleet Provisioning provides two ways to implement this: by trusted user or by claim. Let us look at the process flow for Fleet Provisioning by claim.

Some devices do not have the capability to accept credentials over a secure transport, and the manufacturing supply chain is not equipped to customize devices at manufacturing time. AWS IoT provides a path for these devices to receive a unique identity when they are deployed.

Device makers must load each device with a shared claim certificate in firmware. This claim certificate should be unique per batch of devices. The firmware containing the claim certificate is loaded by the contract manufacturer without the need to perform customization. When the device establishes a connection with AWS IoT for the first time, it exchanges the claim certificate for a unique X.509 certificate signed by the AWS certificate authority and a private key. The device should send a unique token, such as a serial number or embedded hardware secret with its provisioning request that the fleet provisioning service can use to verify against an allow list.



Registration flow

1. Device connects with claim certificate to AWS IoT Core
2. Fleet Provisioning service creates new certificate and private key assigned with AWS CA.
3. Device writes the unique private key and certificate to secure storage.
4. With the parameters published from the device, Fleet Provisioning service triggers Pre-Provisioning lambda function.
5. Lambda function performs additional verification logic such as checking the hardware secret against a DynamoDB table with verified devices.
6. Fleet provisioning service create IoT Thing, Policy, and activates certificate based on Provisioning template and publishes this to the device.

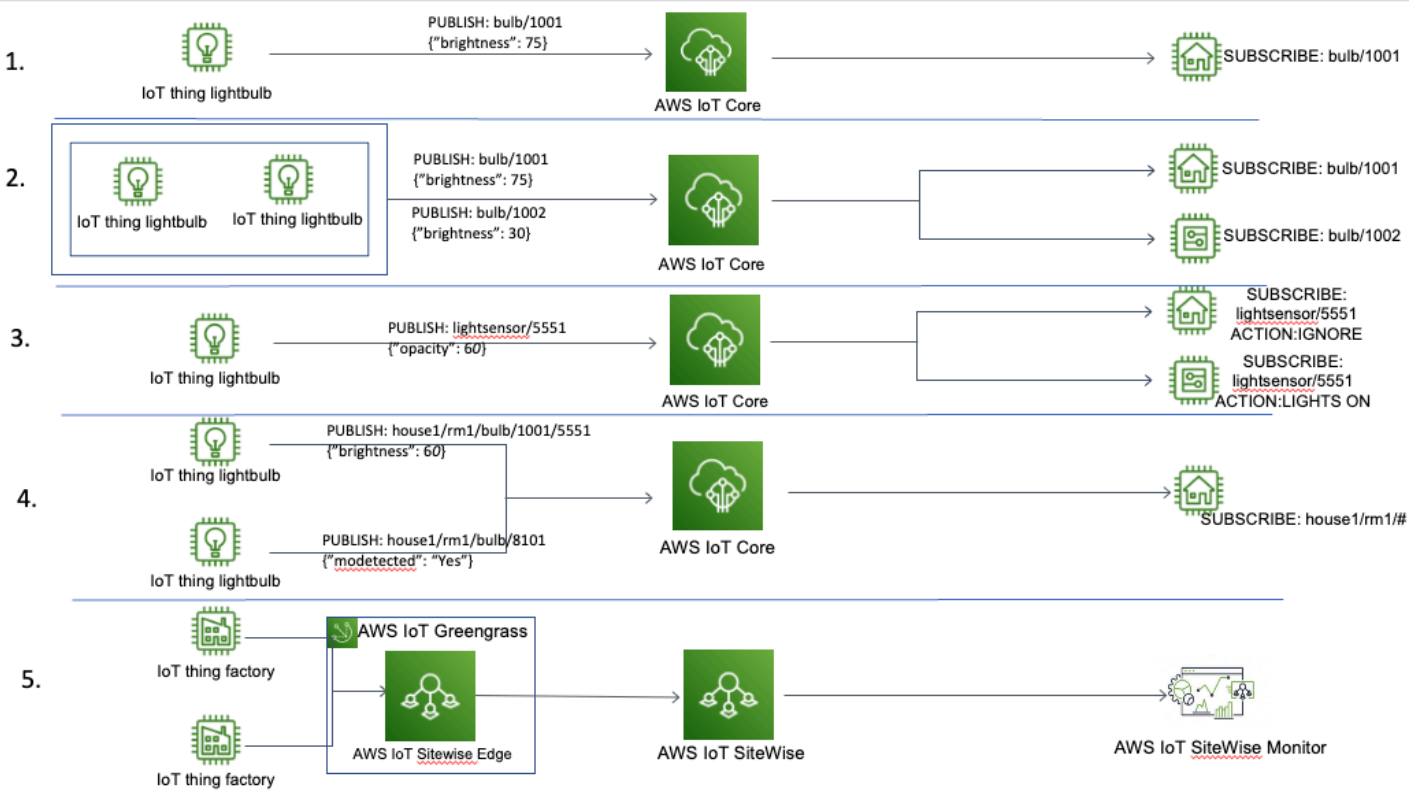
7. Device applies the new configuration and connects with the unique private key, certificates and configuration.

Device telemetry

There are many uses cases where the value for IoT is in collecting telemetry on how a machine or asset is performing. For example, this data can be used to predict costly, unforeseen equipment failures. Telemetry must be collected from the machine and sent to an IoT application. Another benefit of sending telemetry is the ability of your cloud applications to use this data for analysis and to interpret optimizations that can be made to your firmware over time.

Read-only telemetry data is collected and transmitted to the IoT application. Telemetry data and command and control should be kept on separate MQTT topic namespaces. Telemetry data topics should start with a prefix such as 'data', for example data/device/sensortype. Control plane topics should start with a prefix such as 'command'.

From a logical perspective, we have defined several scenarios for capturing and interacting with device data telemetry.



Options for capturing telemetry

1. One publishing topic and one subscriber. For example, a smart light bulb that publishes its brightness level to a single topic where only a single application can subscribe.
2. One publishing topic with variables and one subscriber. For example, a collection of smart bulbs publishing their brightness on similar but unique topics. Each subscriber can listen to a unique publish message.
3. Single publishing topic and multiple subscribers. In this case, a light sensor that publishes its values to a topic that all the light bulbs in a house subscribe to.
4. Multiple publishing topics and a single subscriber. For example, a collection of light bulbs with motion sensors. The smart home system subscribes to all of the light bulb topics, inclusive of motion sensors, and creates a composite view of brightness and motion sensor data.
5. AWS IoT Sitewise edge software running on an edge gateway is used to collect, organize, process, and monitor equipment data on-premises and sends it to AWS IoT Sitewise cloud service for data storage, organization and visualization.

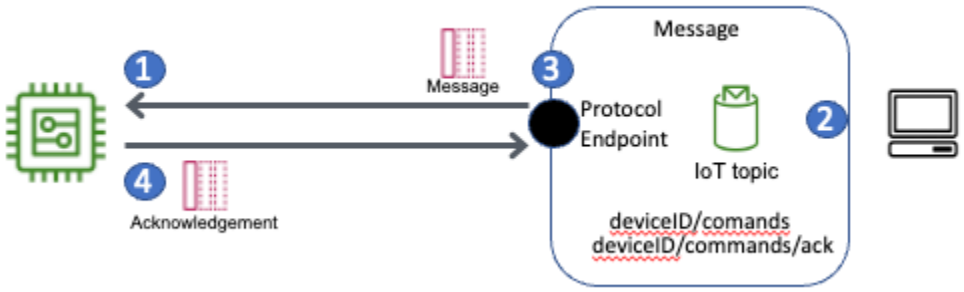
   Other AWS IoT services which can be used for device telemetry data ingestion are AWS IoT Sitewise for industrial data and AWS IoT FleetWise for vehicle data.

Device commands

When you are building an IoT application, you need the ability to interact with your device through commands remotely. For example:

- In the industrial vertical, remote commands are used to request specific data from a piece of equipment.
- In the smart home vertical, remote commands are used to schedule an alarm system remotely.

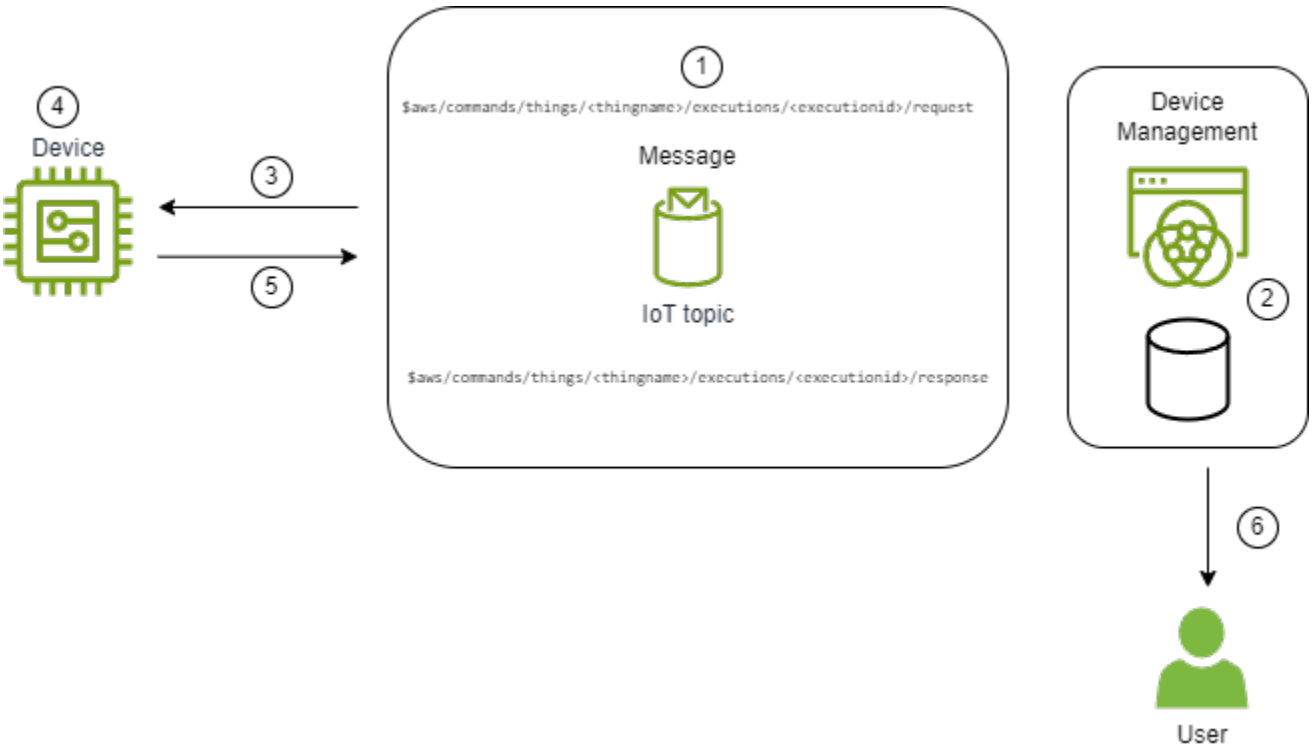
With AWS IoT Core, you can use the bi-directional MQTT protocol to implement command and control of devices. The device subscribes to a specific command MQTT topic. When the device receives a command message, it should verify that the message arrived in the correct order by implementing a sequential ID. The device should then perform the action, and publish a message to the cloud with the results of the command. This makes sure that commands are acted upon in order, and the device's current state is consistently known and maintained in the cloud.



Using a message broker to send commands to a device.

AWS IoT Device Management Commands

AWS IoT Device Management Commands feature enables customers to send remote commands to IoT devices at scale, facilitating remote monitoring, control and diagnostics. Devices subscribe to MQTT topics to receive user-defined payload from the cloud. The commands feature support delivering messages to reserved MQTT topics and for added flexibility, the target resource can be either a ThingName or a ClientID (for devices not registered in the IoT registry). Each command is a reusable AWS resource that supports granular permissions controls — you can authorize users to send specific commands targeting individual devices. If a command execution needs to be initiated within a specific time frame, you can configure them to expire after a defined timeout period. You have full visibility into command execution through status tracking, and you can optionally configure notifications to be alerted when the command state changes. Command is well suited for remotely sending specific instructions, triggering actions or modifying device configurations on-demand. Common use cases include retrieving device logs, initiating changes to device states, and allowing end users to remotely control devices through web application or a companion app, such as turning lights on and off or starting the air conditioner or fan. To send commands using Commands feature, see the following diagram:



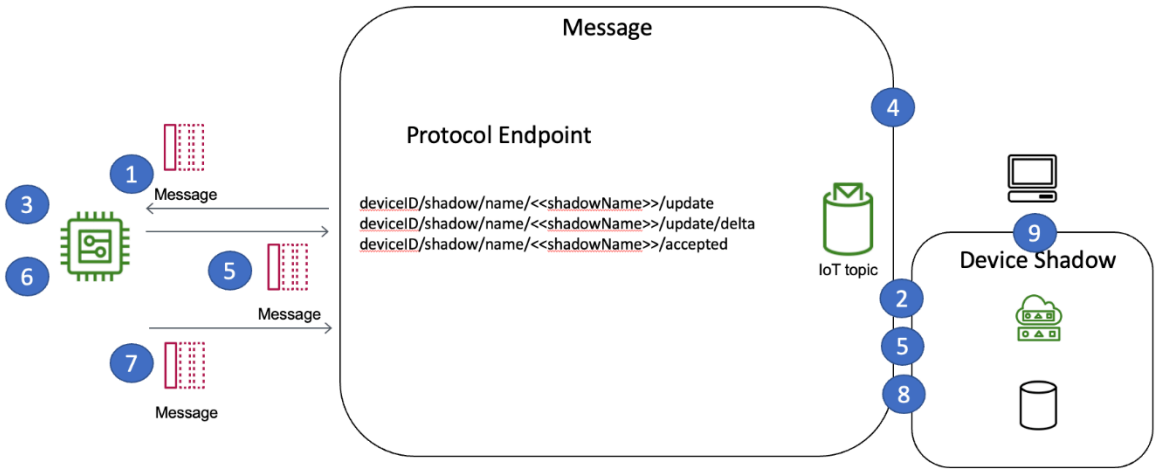
Sending commands to devices using AWS IoT Device Management Commands feature

1. A device subscribes to the commands topic `$aws/commands/things/<thingname>/executions/#/request` or `$aws/commands/clients/<clientId>/executions/#/request` upon which IoT commands payload will be delivered.
2. Create pre-defined commands and store them in AWS IoT Device Management Commands for reusability.
3. A user initiates a commands execution through user application, which publishes command payload to the request topic.
4. The device performs the actions specified by the commands execution.
5. The device publishes command execution progress and updates status through `$aws/commands/things/<thingname>/executions/<executionid>/response` or `$aws/commands/clients/<clientId>/executions/<executionid>/response` topic.
6. Commands publish update notifications through `$aws/events/commandExecutions/<CommandId>/#`. The user can configure AWS IoT rules to receive notifications optionally.

AWS IoT Device Shadow service Service

AWS provides a feature called AWS IoT Device Shadow services to implement command and control over MQTT using these best practices. The Device Shadow has several benefits over using standard MQTT topics, such as a clientToken, to track the origin of a request, version numbers for managing conflict resolution, and the ability to store commands in the cloud in the event that a device is offline and unable to receive the command when it is issued. The device's shadow is commonly used in cases where a command needs to be persisted in the cloud even if the device is currently not online. When the device is back online, the device requests the latest shadow information and executes the command.

IoT solutions that use the Device Shadow service in AWS IoT Core manage command requests in a reliable, scalable, and straightforward fashion. The Device Shadow service follows a prescriptive approach to both the management of device-related state and how the state changes are communicated. This approach describes how the Device Shadows service uses a JSON document to store a device's current state, desired future state, and the difference between current and desired states.



Using Device Shadow with devices

1. The device should check its desired state as soon as it comes online by subscribing to the $aws/things/<<thingName>>/shadow/name/<<shadowName>>/get topic. A device reports initial device state by publishing that state as a message to the update topic $aws/things/<<thingName>>/shadow/name/<<shadowName>>/update.

2. The Device Shadow reads the message from the topic and records the device state in a persistent data store.
3. A device subscribes to the delta messaging topic $aws/things/<<thingName>>/shadow/name/<<shadowName>>/update/delta upon which device-related state change messages will arrive.
4. A component of the solution publishes a desired state message to the topic $aws/things/<<thingName>>/shadow/name/<<shadowName>>/update and the Device Shadow tracking this device records the desired device state in a persistent data store.
5. The Device Shadow publishes a delta message to the topic $aws/things/<<thingName>>/shadow/name/<<shadowName>>/update/delta, and the Message Broker sends the message to the device.
6. A device receives the delta message and performs the desired state changes.
7. A device publishes an acknowledgment message reflecting the new state to the update topic $aws/things/<<thingName>>/shadow/name/<<shadowName>>/update and the Device Shadow tracking this device records the new state in a persistent data store.
8. The Device Shadow publishes a message to the $aws/things/<<thingName>>/shadow/name/<<shadowName>>/update/accepted topic.
9. A component of the solution can now request the updated state from the Device Shadow.

AWS IoT Device Management Jobs for device commands

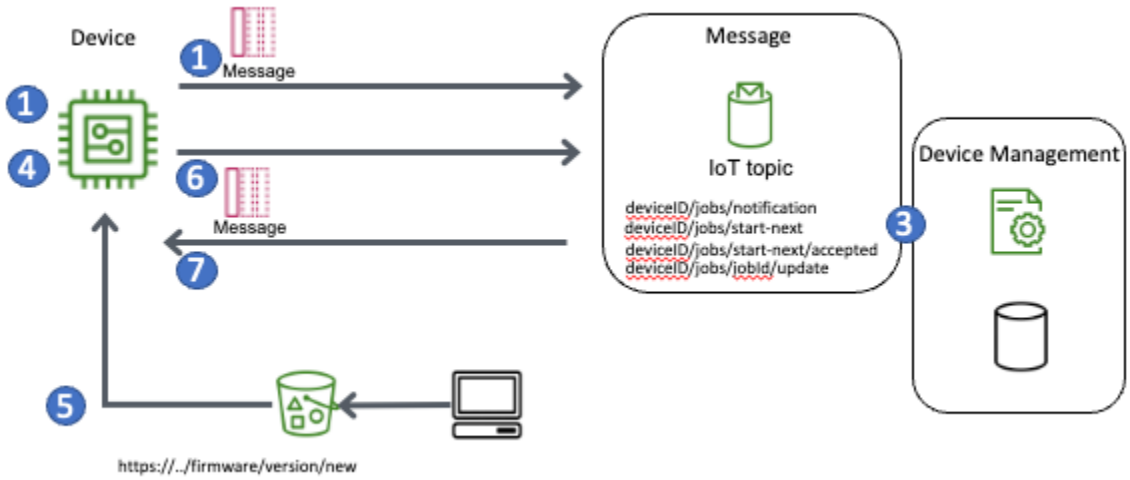
In addition to the features described above for device commands, customers can also use AWS IoT Jobs to create a command pipeline, where the device infers the command from the payload of the MQTT message, as opposed to the topic. This enables customers to perform new kinds of remote operations with minimal device-side code changes. You can control the rate of roll-outs using Jobs, and provide abort / retry / timeout criteria to further customize the behavior of the job. AWS IoT Jobs integrates with Fleet Indexing and Thing Groups, which allows you to search your fleet and target devices in your fleet that meet specific criteria. With Job Templates, you can pre-define device-commands and create a library of reusable commands with just a few clicks on the target of your choice.

Firmware updates

Supporting firmware upgrades without human intervention is critical for security, scalability, and delivering new capabilities.

AWS IoT Device Management provides a secure and straightforward way for you to manage IoT deployments including executing and tracking the status of firmware updates. AWS IoT Device Management uses the MQTT protocol with AWS IoT message broker and AWS IoT Jobs to send firmware update commands to devices, as well as to receive the status of those firmware updates over time. AWS IoT Jobs also integrates with AWS Code Signer to provide additional security to help prevent unauthorized firmware updates and man in the middle attacks. Firmware images can be signed with a private key in the cloud using the code signing feature, and the device verifies the integrity of that firmware image with the corresponding public key.

To implement firmware updates using AWS IoT Device Management and AWS IoT Jobs, see the following diagram.



Updating firmware on devices

1. A device subscribes to the IoT job notification topic $aws/things/<<thingName>>/jobs/notify-next upon which IoT job notification messages will arrive.
2. A device publishes a message to $aws/things/<<thingName>>/jobs/start-next to start the next job and get the next job, its job document, and other details including states saved in statusDetails.
3. The AWS IoT Jobs service retrieves the next job document for the specific device and sends this document on the subscribed topic $aws/things/<<thingName>>/jobs/start-next/accepted.
4. A device performs the actions specified by the job document using the $aws/things/<<thingName>>/jobs/jobId/update MQTT topic to report on the progress of the job.

5. During the upgrade process, a device downloads firmware using a pre-signed URL for Amazon S3. Use code-signing to sign the firmware when uploading to Amazon S3. By code-signing your firmware the end-device can verify the authenticity of the firmware before installing. FreeRTOS devices can download the firmware image directly over MQTT to alleviate the need for a separate HTTPS connection.
6. The device publishes an update status message to the job topic $aws/things/<<thingName>>/jobs/jobId/update reporting success or failure.
7. Because this job's execution status has changed to final state, the next IoT job available for execution (if any) will change.

IoT Edge computing

IoT Edge computing involves hardware and software technologies that enable storage, computing, processing, and networking close to the devices that generates or consumes data within a smart home, connected vehicle, factory or other industrial environments. IoT Edge computing moves processing and analysis closer to endpoints where data is generated, delivering real-time responsiveness and reducing costs associated with transferring large amounts of data to cloud services.

With AWS IoT Greengrass, you can enable devices to take actions, aggregate data, and filter it locally on the device. Some of the use cases for AWS IoT Greengrass include:

- Smart homes where an AWS IoT Greengrass gateway is used as a hub for home automation
- Smart factories where AWS IoT Greengrass can facilitate ingestion and local processing of data from the shop floor
- In autonomous vehicles where AWS IoT Greengrass is used for sensor data collection and securely sending it to AWS

AWS IoT Greengrass can act as a secure, authenticated, MQTT connection endpoint for other edge devices (also known as *client devices*), which otherwise would typically connect directly to AWS IoT Core. This capability is useful when client devices do not have direct network access to the AWS IoT Core endpoint.

In this IoT edge computing scenario, we describe using AWS IoT Greengrass with client devices for the following use cases:

- For client devices to send data to AWS IoT Greengrass

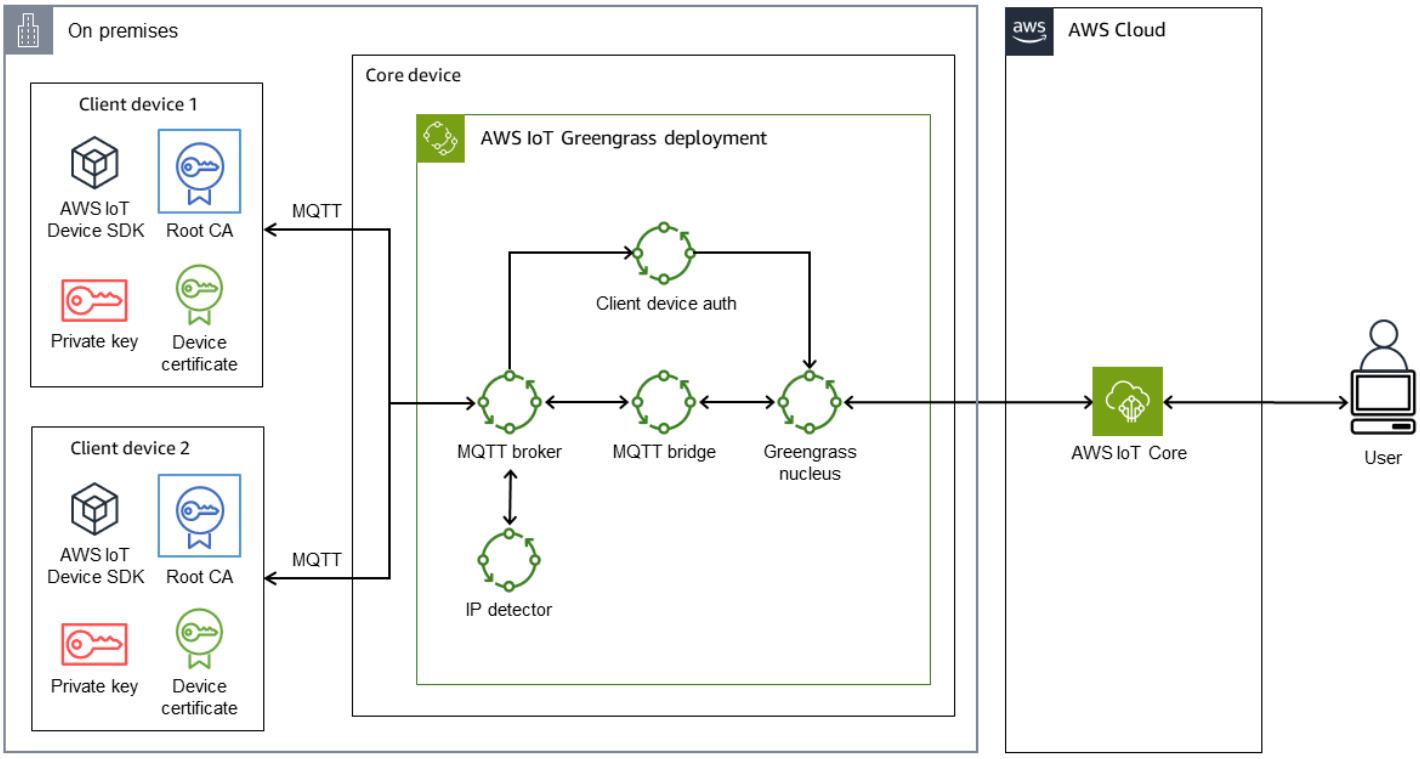
- For AWS IoT Greengrass to forward data to AWS IoT Core
- To take advantage of advanced AWS IoT Core rules engine features

These capabilities require installing and configuring the following components on the AWS IoT Greengrass device:

- MQTT broker
- MQTT bridge
- Client device authentication
- IP detector

Note: The published messages from client devices must be in JSON format or [Protocol Buffers (protobuf)Protocol Buffers] format.

This architecture pattern describes how to set up AWS IoT Greengrass for IoT edge computing.



IoT edge computing using AWS IoT Greengrass

The architecture includes:

- Two client devices. Each device contains a private key, a device certificate, and a root certificate authority (CA) certificate. The AWS IoT Device SDK, which contains an MQTT client, is also installed on each client device.
- A core device that has AWS IoT Greengrass deployed with the following components:
  - MQTT broker
  - MQTT bridge
  - Client device authentication
  - IP detector

This architecture supports the following scenarios:

- Client devices can use their MQTT client to communicate with one another through the core device's MQTT broker.
- Client devices can also communicate with AWS IoT Core in the cloud through the core device's MQTT broker and the MQTT bridge.
- AWS IoT Core in the cloud can send messages to client devices through the MQTT test client and the core device's MQTT bridge and MQTT broker.

Best practice recommendations include:

- The payload of the messages from client devices should be in either JSON or Protobuf format in order to take advantage of the advanced features of the AWS IoT Core rules engine, such as transformation and conditional actions.
- Configure the MQTT bridge to allow bidirectional communication.
- Configure and deploy the IP detector component in AWS IoT Greengrass to make sure that the core device's IP addresses are included in the subject alternative name (SAN) field of the MQTT broker certificate. The subject alternative name (SAN) plays a critical role in the server name verification on the TLS client end. It helps the TLS client make sure that it connects to the correct server and helps avoid man-in-the-middle attacks during TLS session setup.

For more information, see Set up and troubleshoot AWS IoT Greengrass with client devices.

Generative AI and IoT

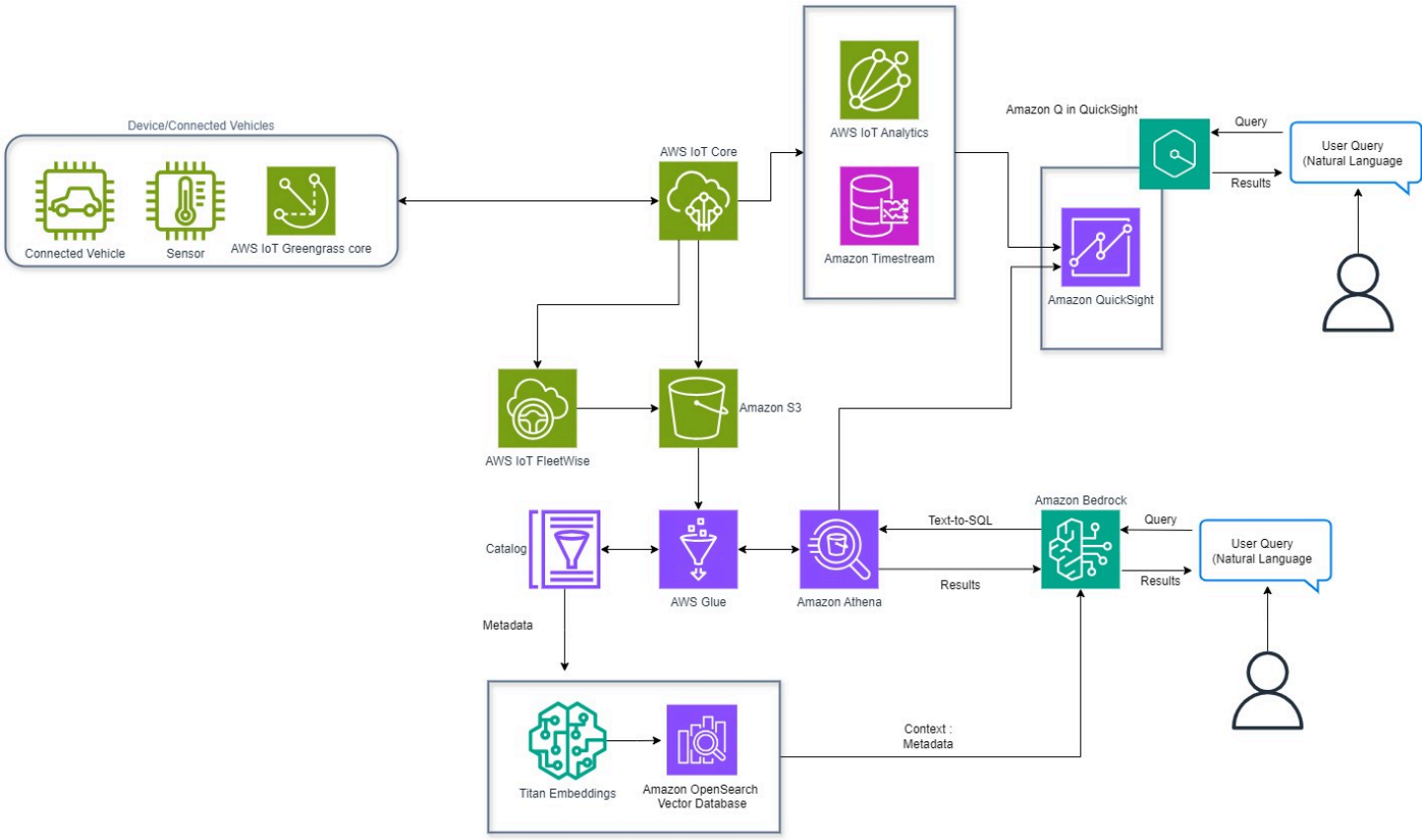
Generative artificial intelligence (generative AI) is a type of AI that can create new content and ideas, including conversations, images and videos. AI technologies attempt to mimic human intelligence in nontraditional computing tasks, such as image recognition, natural language processing (NLP), and translation. It reuses data that has been historically trained for better accuracy to solve new problems. However, the generative AI must be continuously fed with fresh, new data to move beyond its initial, predetermined knowledge and adapt to future, unseen parameters. This is where the IoT becomes pivotal in unlocking generative AI's full potential by providing multi-modality data sources such as sensor data, images and voice beyond just text. The combination of IoT and generative AI offers enterprises the unique advantage of creating meaningful impact for their business. Generative AI and IoT can be used in a range of applications such as interactive chatbots, IoT low code assistants, automated IoT data analysis and reporting, IoT synthetic data generation for model trainings and Generative AI at the edge for low latency and data privacy use cases.

In this scenario we discuss integrating IoT and generative AI using AWS IoT Core, AWS IoT Fleetwise, AWS IoT SiteWise, AWS IoT TwinMaker, AWS IoT Greengrass with AWS generative AI services such as Amazon Q or Amazon Bedrock and other AWS services such as Amazon Simple Storage Service (S3), AWS Glue, Amazon Timestream, Amazon OpenSearch Service, Amazon Kinesis, and Amazon Athena for data ingestion, storage, processing, analysis, and querying to build automated IoT data analysis and reporting applications. This allows capabilities like real-time monitoring, advanced analytics, predictive maintenance, anomaly detection, and customizations of dashboards.

Amazon Q in QuickSight improves business productivity using generative BI (enable any user to ask questions of their data using natural language) capabilities to accelerate decision making in IoT scenarios. With new dashboard authoring capabilities made possible by Amazon Q in QuickSight, IoT data analysts can use natural language prompts to build, discover, and share meaningful insights from IoT data. Amazon Q in QuickSight makes it easier for business users to understand IoT data with executive summaries, a context-aware data Q&A experience, and customizable, interactive data stories. These workflows optimize IoT system performance, troubleshoot issues, and enable real-time decision-making.

For example, in an industrial setting, you can monitor equipment, detect anomalies, provide recommendations to optimize production, reduce energy consumption, and reduce equipment failures. By combining IoT and generative AI, you will be able to gain situational awareness and understand what happened? why it happened? and what to do next? in your IoT applications.

In addition to collecting IoT data from devices, you can also send commands to devices. In the industrial asset monitoring example, if you find an abnormal situation from the asset IoT data collected, you can send a command using the Generative AI assistant to the device to collect more fine-granularity data for further troubleshooting.



IoT and generative AI for automated data analysis, control and reporting

For more information, see Emerging Architecture Patterns for Integrating IoT and generative AI on AWS.

Operational excellence

The operational excellence pillar includes operational practices and procedures used to manage production workloads. All processes and procedures of operational excellence must be documented, tested, and regularly reviewed. Best practices drive automating planned changes and responses to unexpected operational events.

Focus areas

Design principles

In addition to the overall Well-Architected Framework operational excellence design principles, there are five design principles for operational excellence for IoT:

- **Plan for device provisioning**: Design your device provisioning process to create your initial device identity in a secure location. Implement a public key infrastructure (PKI) that is responsible for distributing unique certificates to IoT devices. As described above, selection of crypto hardware with a pre-generated private key and certificate alleviates the operational cost of running a PKI. Otherwise, PKI can be done offline with a Hardware Security Module (HSM) during the manufacturing process, or during device bootstrapping. Use technologies that can manage the Certificate Authority (CA) and HSM in the cloud.
- **Implement device bootstrapping**: The design must have ability to devices to programmatically update their configuration information using a globally distributed bootstrap API. A bootstrapping design makes sure that you can programmatically send the device new configuration settings through the cloud. These changes should include settings such as which IoT endpoint to communicate with, how frequently to send an overall status for the device, and

updated security settings such as server certificates. The process of bootstrapping goes beyond initial provisioning and plays a critical role in device operations by providing a programmatic way to update device configuration through the cloud. A bootstrapping API and endpoint must be available for the entire defined life of all devices, and must be able to respond to requests for all versions of firmware that have ever been deployed on a device. Devices that support personalization by a technician in the industrial domain or user in the consumer domain can also undergo provisioning. For example, a smartphone application that interacts with the device over Bluetooth LE and with the cloud over Wi-Fi.

- **Document device communication patterns**: An operations team must formulate how the behavior of a device will scale once deployed to a fleet of devices. A cloud engineer should review the device communication patterns and extrapolate the total expected inbound and outbound traffic of device data and determine the expected infrastructure necessary in the cloud to support the entire fleet of devices. During operational planning, these patterns should be measured using device and cloud-side metrics to make sure that expected usage patterns are met in the system.
- **Implement over the air (OTA) updates**: In order to benefit from long-term investments in hardware, you must be able to continuously update the firmware on the devices with new capabilities. In the cloud, you can apply a firmware update process that allows you to target specific devices for firmware updates, roll out changes over time, track success and failures of updates, and have the ability to roll back or put a stop to firmware changes based on key performance indicators (KPIs).
- **Implement functional testing on physical assets**: IoT device hardware and firmware must undergo rigorous testing before being deployed in the field. The tests make sure that your IoT device will perform as expected when deployed. Acceptance and functional testing are critical on the path to production. The goal of functional testing is to run your hardware components, embedded firmware, and device application software through rigorous testing scenarios, such as intermittent or reduced connectivity or failure of peripheral sensors, while profiling the performance of the hardware.
- **Design and build for operations at scale**: Design and build a solution for logging, monitoring, troubleshooting, fleet management, life cycle device and application management at scale. The amount of data ingested from connected devices with low latency and high throughput rate should scale and not impact the application. For example, the data plane operations should seamlessly operate when the fleet grows.

Definitions

There are four focus areas for operational excellence:

1. Organization
2. Prepare
3. Operate
4. Evolve

In addition to what is covered by the Well-Architected Framework concerning process, runbooks, and game days, there are specific areas you should review to drive operational excellence within IoT applications.

Organization

For decades, we have seen manufacturing companies attempt to bring information technology (IT) systems into operations technology (OT) environments. Smart manufacturing increases efficiency and productivity, optimizes costs, and enhances product quality by integrating advanced technologies like robotics, AI, and IoT. Smart manufacturing provides greater flexibility and evolves through data-driven decision-making. Leaders can develop a collaborative culture where open dialogue and trust are encouraged. Clarifying roles and responsibilities and establishing accountability between teams is crucial.

While converging OT and IT brings new efficiencies, it also brings new risks. The merging of OT and IT has begun at most companies and includes the merging of systems, organizations, and policies. These components are often at different points along their journey, making it necessary to identify each one and where it is in the process to determine where additional attention is needed. It is important to understand that OT is no longer an air-gapped system that is hidden away from cyber risks, and so it now shares many of the same risks as IT.

IOTOPS01: How do you evaluate governance and compliance requirements?

A robust governance strategy across people, process and technology covering both internal and external stakeholders can help run business efficiently. From a people perspective, well-documented policies and processes, different team's role clarity with measurable goals, and

a transparent decision-making framework are essential. Process-wise, a business case-driven approach to selecting investments, proven program management methodology, financial discipline, hardware supply chain and a robust risk framework are key. And, from a technology perspective, a technology architecture blue-print for IoT and IIoT adoption, playbooks, runbooks, and drills for operational functions such as system design, maintenance, telemetry, incident response and disaster recovery with assigned ownership are crucial.

IOTOPS01-BP01 Conduct an OT and IT cybersecurity risk assessment using a common framework

When taking advantage of IT technologies in OT environments, it is important to conduct a cybersecurity risk assessment using frameworks to fully understand and proactively manage risks (for example, ISA/IEC 62443). Companies with maturing OT and IT convergence display common patterns.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTOPS01-BP01-01: *Conduct a cyber-security risk assessment so that the risks, gaps and vulnerabilities are fully understood and can be proactively managed. Create and maintain an up-to-date threat model.*

- Proactively managing risks, gaps, and vulnerabilities between OT and IT
- Up to date threat modeling capabilities for both OT and IT
- Define the system being assessed
- Identify threats, vulnerabilities and consequences of unintended access or behavior
- Rank the discovered risks
- Develop a risk mitigation strategy

IOTOPS01-BP02 Evaluate if OT and IT teams use separate policies and controls to manage cybersecurity risks or if they use the same policy

The ongoing maturity and adoption of cloud within IT and now within OT creates a more common environment. A comprehensive enterprise and OT security policy will encompass risks across the entirety of the business. This allows for OT risks such as safety to be recognized and addressed within IT. Conversely, this allows for IT risks such as bots and ransomware to be addressed within OT. While policies might converge, mitigation strategies will still differ in many cases.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTOPS01-BP02-01

- OT and IT maintaining separate risk policies.
- The degree of isolation for process control and safety networks.
- Interconnectedness of OT and IT systems and networks.
- Security risks that were applicable to either IT or OT might now apply to both.
- Singular security control policy that governs both OT and IT.
- Different mitigation strategies as appropriate for OT and for IT. For example, the speed of patching is often different between OT and IT by design.
- The use of holistic approaches to manage OT and IT risk.

Resources

- How to approach threat modeling
- AWS Threat Modeling workshop
- Assessing OT and IIoT cybersecurity risk
- Guidance on using ISA/IEC 62443 for IIoT projects

IOTOPS02: Is there a central cloud center of excellence (CCoE) with equivalent represent ation from OT and IT in industrial organizations?

Given the historical nature of the separation of IT and OT, organizations might still operate in silos. An indication of converging maturity is how well those teams are working across divisions or have even removed silos altogether. Meaningful OT/IT convergence requires focused and organized effort, which a CCoE can facilitate. A CCoE is a multi-disciplinary team of passionate OT and IT subject matter experts (SMEs) who act as change agents to accelerate IIoT adoption by standardizing and evangelizing best practices, developing repeatable patterns to scale implementation, driving governance, and providing thought leadership. The CCoE can start small with 3-5 members, cross-trained in both IT and OT aspects and can scale as needed. For a CCoE to be successful, it requires executive sponsorship and ability to act autonomously. The CCoE can focus on making incremental improvements instead of a big-bang approach. A prioritization

framework is used to identify pilot use cases starting with low-risk, high value, and low effort use cases with measurable success metrics. After the pilot use cases are deployed and business value demonstrated, this activity continues cyclically to implement the pipeline of prioritized use cases.

IOTOPS02-BP01 Consolidate resources into centers of excellence to bring focus to new or transforming enterprises

You can consider creating CCoEs around security to consolidate experts from around the company. Such focused areas are a central point of technical authority and accelerates decision making.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTOPS02-BP01-01

- Consolidating resources into centers of excellence.
- Security experts consolidated from around the company into a singular organization.
- Defining security focus areas based on risk priorities.
- Having a central point of security authority.
- OT and IT teams operating uniformly.
- Well understood and applied incident response decision rights in OT.
- Availability is vital in OT. Systems must run in order to produce and manufacture product. Downtime equals lost revenue.
- Security defenses are often designed to wrap around OT zones while restricting the conduits between them.

Resources

- AWS Blogs: 7 Pitfalls to Avoid When Building a CCOE
- AWS Blogs: Using a CCOE to Transform the Entire Enterprise
- Managing Organizational Transformation for successful OT/IT convergence

Prepare

For IoT applications, the need to procure, provision, test, and deploy hardware in various environments means that the preparation for operational excellence must be expanded to cover

aspects of your deployment that will primarily run-on physical devices and will not run in the cloud. Operational metrics must be defined to measure and improve business outcomes and then determine if devices should generate and send any of those metrics to your IoT application.

It is essential that you review how to make sure that your IoT workloads are resilient to failures, how devices can self-recover from issues without human intervention, and how your cloud-based IoT application will scale to meet the needs of an ever-increasing load of connected devices.

When using an IoT system, you have the opportunity to use additional components/tools for handling IoT operations. These tools include services that allow you to monitor and inspect device behavior, capture connectivity metrics, provision devices using unique identities, and perform long-term analysis on top of device data.

IOTOPS03: Do you organize the fleet to quickly identify devices?

The ability to quickly identify and interact with specific devices gives you the agility to troubleshoot and potentially isolate devices in case you encounter operational challenges. When operating large-scale device fleets, you need to deploy ways to organize, index, and categorize them. This is useful when targeting new device software with updates and when you need to identify why some devices in your fleet behave differently than others.

IOTOPS03-BP01 Use static and dynamic device hierarchies to support fleet operations

Using a software registry, devices can be categorized into static groups based on their fixed attributes (such as version or manufacturer) and into dynamic groups based on their changing attributes (such as battery percentage or firmware version). Operationalizing devices in groups can help you manage, control, and search for devices more efficiently.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTOPS03-BP01-01 *Manage several devices at once by categorizing them into static groups and hierarchy of groups.*

- Build a hierarchy of static groups for efficient categorization and indexing of your devices.
- Use provisioning templates to assign devices to static groups as they are provisioned for the first time.

- For example, categorize all sensors of a car under a car group and all the cars under a vehicle group. Child groups inherit policies and permissions attached to their respective parent groups.

Prescriptive guidance IOTOPS03-BP01-02 *Build a device index to efficiently search for devices, and aggregate registry data, runtime data, and device connectivity data.*

- Use a fleet indexing service from AWS IoT Core to index device and group data.
- Use a device index to search registry metadata, stateful metadata, and device connectivity status metadata.
- Use a group index to search for groups based on group name, description, attributes, and all parent group names.
- For example, if you want to send over-the-air (OTA) updates only to devices that are sufficiently charged, then define a dynamic group for devices with more than 90% battery. Devices will dynamically be added to or removed from the group as their battery percentage crosses the threshold. Send OTA updates to all things under this dynamic group

IOTOPS03-BP02 Use index and search services to enable rapid identification of target devices

A large IoT deployment can have millions of sensors sending data to the cloud. A separate indexing and search service can make it straightforward to index and organize the device data, and search for devices by attributes. Ingesting device data to a search service, for example, Amazon OpenSearch Service, makes it straightforward to use powerful search, visualization, and analytics capabilities of OpenSearch Service to organize and search for devices. You can ingest your device data and the state to OpenSearch Service seamlessly.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTOPS03-BP02-01 *Use an indexed data store to get, update, or delete device state.*

- Use messaging topics to enable applications and things to get, update, or delete the state information for a Thing (Thing Shadow).
- Ingest the shadow data to Firehose through the AWS IoT Core rules engine.
- Ingest the data from Firehose to Amazon OpenSearch Service through built-in destination options for OpenSearch Service.

- Configure search and visualizations on the data directly or through the OpenSearch Dashboards console.
- In AWS, you can create an AWS IoT thing for each physical device in the device registry of AWS IoT Core. By creating a thing in the registry, you can associate metadata to devices, group devices, and configure security permissions for devices. An AWS IoT thing should be used to store static data in the thing registry while storing dynamic device data in the thing's associated device shadow. A device's shadow is a JSON document that is used to store and retrieve state information for a device.

Resources

- AWS IoT Core - Fleet indexing serviceAWS IoT Core - Fleet indexing
- AWS IoT Core - AWS IoT Device Shadow service
- Amazon OpenSearch Service
- The Internet of Things on AWS – Official Blog: Archive AWS IoT Device Shadow services in Amazon OpenSearch Service
- Analyze device-generated data with AWS IoT and Amazon OpenSearch Service

IOTOPS04: How do you verify that newly provisioned devices have the required operational prerequisites?

Logical security for IoT and data centers is similar in that both involve predominantly machine-to-machine authentication. However, they differ in that IoT devices are frequently deployed to environments that cannot be assumed to be physically secure. IoT applications also commonly require sensitive data to traverse the internet. Due to these considerations, it is vital for you to have an architecture that determines how devices will securely gain an identity, continuously prove their identity, be seeded with the appropriate level of metadata, be organized and categorized for monitoring, and enabled with the right set of permissions.

IOTOPS04-BP01 The device management processes should be automated, data-driven, and based on previous, current, and expected device behavior

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTOPS04-BP01-01 *Defining how devices are provisioned must include how the devices are manufactured and how they are registered for both greenfield and brownfield fleet of devices.*

- In AWS IoT, you can use multiple features to provision your individual device identities signed by your CA to the cloud. This path involves provisioning devices with identities and then using just-in-time-provisioning (JITP), just-in-time-registration (JITR), fleet provisioning or Multi-Account Registration to securely register your device certificates to the cloud. Using AWS services including Route 53, Amazon API Gateway, Lambda, and DynamoDB, will create a simple API interface to extend the provisioning process with device bootstrapping.
- IoT applications must support incremental rollout and rollback strategies. By having this as part of the operational efficiency plan, you will be equipped to launch a fault-tolerant, efficient IoT application.

Resources

- Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core
- How to automate onboarding of IoT devices to AWS IoT Core at scale with Fleet Provisioning

IOTOPS05: How do you govern device fleet provisioning process?

IoT solutions can scale to millions of devices and this requires device fleets to be well planned from the perspectives of provisioning processes and metadata organization. Maintain a full chain of security controls over who or what processes can trigger device provisioning to decrease the likelihood of inviting unintended (or rogue) devices into your fleet.

IOTOPS05-BP01 Document how devices join your fleet from manufacturing to provisioning

Document the whole device provisioning process to clearly define the responsibilities of different actors at different stages. The end-to-end device provisioning process involves multiple stages owned by different actors. Documenting the plan and processes by which devices onboard and join the fleet affords the appropriate amount of review for potential gaps.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTOPS05-BP01-01 *Document each step (manual and programmatic) of all the stages for the corresponding actors of that stage and clearly define the sequence.*

- Identify the steps at each stage and the corresponding actors.
  - Device assembly by hardware manufacturer.
  - Device registration by service and solution provider.
  - Device activation by the end user of the service or solution provider.
- Clearly define and document the dependencies and specific steps for each actor from device manufacturer to the end user.
- Document whether devices can self-provision or are user-provisioned and how you can make sure that newly provisioned devices are yours.

Prescriptive guidance IOTOPS05-BP01-02 *Assign device metadata to enable straightforward grouping and classification of devices in a fleet.*

- The metadata can be used to group the devices in groups to search and force common actions and behaviors.
- For example, you can assign the following metadata at the time of manufacturing:
  - Unique ID
  - Manufacturer details
  - Model number
  - Version or generation
  - Manufacturing date
- If a particular model of a device requires a security patch, then you can easily target the patch to the devices that are part of the corresponding model number group. Similarly, you can apply the

patches to devices manufactured in a specific time frame or belonging to a particular version or generation.

- Along with creating a virtual representation of your device in the device registry, as part of the operational process, you must create thing types that encapsulate similar static attributes that define your IoT devices. A thing type is analogous to the product classification for a device. The combination of thing, thing type, and device shadow can act as your first entry point for storing important metadata that will be used for IoT operations.

IOTOPS05-BP02 Use programmatic techniques to provision devices at scale

Scaling the onboarding and provisioning of a large device fleet can be a bottleneck if there is even one manual step per device. Programmatic techniques define patterns of behavior for automating the provisioning process such that authenticated and authorized devices can onboard at any time. This practice provides a well-documented, reliable, and programmatic provisioning mechanism that is consistent across all devices devoid of human errors.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTOPS05-BP02-01 *Embed provisioning claims into the devices that are mapped to approval authorities recognized by the provisioning service.*

- Generate a provisioning claim and embed it into the device at the time of manufacturing.
- AWS IoT Core can generate and securely deliver certificates and private keys to your devices when they connect to AWS IoT for the first time, using AWS IoT Fleet Provisioning.

Prescriptive guidance IOTOPS05-BP02-2 *Use programmatic bootstrapping mechanisms if you are bringing your own certificates.*

- Determine if you will or won't have device information beforehand
- If you do not have device information beforehand, use just-in-time provisioning (JITP).
  - Enable automatic registration and associate a provisioning template with the CA certificate used to sign the device certificate.
  - For example, when a device attempts to connect to AWS IoT by using a certificate signed by a registered CA certificate, AWS IoT loads the template from the certificate and initiates the JITP workflow.

- If you have device information beforehand, use bulk registration.
  - Specify a list of single-thing provisioning template values that are stored in a file in an S3 bucket.
  - Run the start-thing-registration-task command to register things in bulk. Provide provisioning template, S3 bucket name, a key name, and a role ARN to the command.

IOTOPS05-BP03 Use device level features to enable re-provisioning

A birth or bootstrap certificate is a low-privilege unique certificate that is associated with each device during the manufacturing process. The certificate should have a policy to restrict devices to only allow connecting to specific endpoints to initiate provisioning process and fetch the final certificate. Before a device is provisioned, it should be limited in functionality to help prevent its misuse. Only after a provisioning process is invoked and approved, should the device be allowed to operate on the system as designed.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTOPS05-BP03-01 *Use a certificate bootstrapping process to establish processes for device assembly, registration, and activation.*

- For example, AWS IoT Core offers a fleet provisioning interface to devices for upgrading a birth certificate to long-lived credentials that enable normal runtime operations.

Prescriptive guidance IOTOPS05-BP03-02 *Obtain a list of allowed devices from the device manufacturer.*

- Check the allow list file to validate that the device has been fully vetted by the supplier.
- Make sure that the list is encrypted, securely stored, and can only be accessed by necessary services and users. Even if the list changes, keep the original list securely stored.
- Make sure that this list is securely transferred from the manufacturer to you, is encrypted, and is not publicly accessible.
- Make sure that any bootstrap certificate used is signed by a certificate authority (CA) you own or trust.

IOTOPS06: How do you implement observability for your IoT system?

Observability is a crucial part for your IoT application built to handle device activity at scale. As the main three pillars of observability are logging, metrics and tracing, there are more functional parts of the business goals where you actively troubleshoot and improve the application to mitigate risks.

IOTOPS06-BP01 Implement monitoring to capture logs and metrics

Monitoring is an important part of maintaining the reliability, availability, and performance of your IoT solutions. It is highly recommended to collect monitoring data from all parts of your IoT solution to make it easier to debug a multi-point failure, if one occurs. Create a monitoring plan that answers the questions such as:

- Which resources to monitor (edge, device connectivity, remote operations, or device health)?
- Which tools to use?
- Who has to be notified should an incident or event occurs?

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTOPS06-BP01-01 *Use Amazon CloudWatch to monitor your IoT fleet.*

To support operational insights to your cloud application, generate dashboards for all metrics collected from IoT Core and IoT Device Management. These metrics are available through Amazon CloudWatch Metrics. In addition, CloudWatch Logs contain information such as total successful messages inbound, messages outbound, connectivity success, and errors.

Prescriptive guidance IOTOPS06-BP01-02 *Capture the default metrics emitted by your IoT services and configure alarms for metrics that might indicate business interruption.*

For example, your business deploys a thousand IoT sensors and your operations team wants to be alerted if sensors can no longer connect to the cloud and send telemetry.

- Your IT team administering the AWS account reviews the AWS IoT Core metrics and notes the following metrics to monitor: `Connect.AuthError`, `Connect.ClientError`, `Connect.ClientIDThrottle`, `Connect.S` `Connect.Throttle`. Activity in these metrics constitutes alerting and investigation.
- Your IT team uses CloudWatch to configure new alarms on these metrics when for any period the metrics' SUM of Count is greater than zero.

- Your IT team configures an Amazon SNS topic to notify their paging tool when the new CloudWatch alarms changes status.

For more detail, see Monitor AWS IoT alarms and metrics using Amazon CloudWatch

Prescriptive guidance IOTOPS06-BP01-03 *Use unified monitor dashboard for IoT metrics.*

The unified dashboard in AWS IoT monitor allows identification of potential connectivity and operational problems, reducing the time associated with fleet troubleshooting procedures. The connectivity metrics dashboard available in the AWS IoT monitor, consolidates frequently used metrics from AWS IoT Core, such as successful connections, inbound or outbound messages published, and connection request authorization failures. A guided workflow enables AWS IoT Device Management's Fleet Indexing feature and adds widgets for connected device counts, percentage of devices disconnected, and disconnect reasons to the same page. AWS IoT provides fleet-level and device-level insights driven from the Thing Registry and Device Shadow service through search capabilities such as AWS IoT Fleet Indexing. The ability to search across your fleet eases the operational overhead of diagnosing IoT issues at the device-level or fleet-wide level.

Prescriptive guidance IOTOPS06-BP01-04 *Implementing tracing between all the resources or modules.*

- Visualizing the entire path of requests, from entry to exit helps quickly identifying where failures or performance issues occur.
- In addition to Amazon CloudWatch, it's crucial to instrument to emit trace data. This process can provide further insights into your workload's behavior and performance. Integrate X-Ray into your application to gain insights into its behavior, understand its performance, and pinpoint bottlenecks. Utilize X-Ray Insights for automatic trace analysis.

AWS Lambda Powertools is a suite of utilities that helps with implementing observability best practices without needing to write additional custom code. Powertools provides three core utilities:

- *Tracing* provides a simpler way to send traces from functions to AWS X-Ray. It provides visibility into function calls, interactions with other AWS services, or external HTTP requests. You can add attributes to traces to allow filtering based on key information.
- *Logging* provides a custom logger that outputs structured JSON. It allows you to pass in strings or more complex objects, and takes care of serializing the log output.

- *Metrics* simplify collecting custom metrics from your application, without the need to make synchronous requests to external systems.

IOTOPS06-BP02 Capture and monitor application performance at the edge

Implement tracing and observability methods that provide granular visibility into edge application health, performance, and root cause analysis. By seamlessly connecting the observability strategy for cloud-based applications with those running at the edge, organizations can gain end-to-end visibility and insights for improved application performance.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTOPS06-BP02-01 *Emit device side metrics using agents.*

- AWS IoT Device Defender Detect can collect, aggregate, and monitor metrics data generated by AWS IoT devices to identify devices that exhibit abnormal behavior. Securely deploy the AWS IoT SDK version two on your AWS IoT connected devices or device gateways to collect device-side metrics.
- You can use AWS IoT Device Client to publish metrics as it provides a single agent that covers the features present in both AWS IoT Device Defender and AWS IoT Device Management.
- Publish device-side metrics to the reserved topic in AWS IoT for AWS IoT Device Defender to collect and evaluate.

Prescriptive guidance IOTOPS06-BP02-02 *Collect application logs for tracing capabilities.*

- AWS Distro for OpenTelemetry seamlessly collects and exports metrics and traces to AWS monitoring services. Distro for OpenTelemetry Collector is an agent that runs on your application environment. When it is integrated with AWS IoT Greengrass, this combination extends your observability capabilities to both edge and cloud applications at scale, providing consistent and seamless tracing across your application infrastructure. This integrated approach delivers real-time visibility into application performance

For more information, see Monitor edge application using AWS IoT Greengrass Monitor edge application performance using AWS IoT Greengrass and AWS Distro for OpenTelemetry.

IOTOPS06-BP03 Monitor the status of your IoT devices

You need to be able to track the status of your devices. This includes operational parameters and connectivity. You need to know whether devices have disconnected intentionally or not. Monitoring the status of your device fleet is important in helping troubleshoot device software operation and connectivity disruptions.

Design a state machine for the device connectivity states, from initialization and first connection, to frequent communication (like keep-alive messages) and final state before going offline. Lifecycle events, such as connection and disconnection, can be used to observe and analyze device behavior over a period of time. Additionally, periodic keep-alive messages can track device connectivity status.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTOPS06-BP03-01 *Subscribe to lifecycle events and monitor the connections using keep-alive messages.*

- Capture messages from the IoT message broker whenever a device connects or disconnects. Being able to tell the difference between a clean and dirty disconnect is useful in many scenarios where the devices don't maintain a constant connection to the broker.
- Based on the use case and device constraints, have the device send periodic keep-alive messages to AWS IoT Core and monitor, and analyze the keep-alive messages for anomalies.
- Make sure that the frequency of sending keep-alive messages is not causing any network storms (perhaps by introducing some jitter) in the network or causing rate limits.
- Configure your devices to communicate their status periodically. Implement Last Will and Testament (LWT) messages and periodic device keep-alive messages.

Prescriptive guidance IOTOPS06-BP03-02 *Implement a well-designed device connectivity state machine*

- Make sure that the device communicates when it first comes online and just prior to going offline.
- Implement a wait state for lifecycle events. When a disconnect message is received, wait a period of time and verify that the device is still offline before taking action.

- Specify the interval with which each connection should be kept open if no messages are received. AWS IoT drops the connection after that interval unless the device sends a message or a ping.

Prescriptive guidance IOTOPS06-BP03-03 *Use device connection and disconnection status for anomaly detection.*

- Use connectivity data patterns from devices to detect anomalous behavior in their communication patterns.

Resources

- Use AWS IoT events to monitor IoT devices
- AWS IoT Now Supports WebSockets, Custom Keepalive Intervals, and Enhanced Console
- AWS Solutions Library: Real-Time IoT Device Monitoring with Managed Service for Apache Flink

IOTOPS06-BP04 Use device state management services to detect status and connectivity patterns

Edge and cloud-side management services monitor and analyze parameters, such as device connectivity status and latency, to help in providing diagnostics and predicting anomalies.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTOPS06-BP04-01 *Monitor device state and connectivity patterns.*

- Use (or develop as needed) device, gateway, edge, and cloud management tools that allow monitoring the fleet of devices
- Use logging and monitoring features at all processing points—device, gateway, edge, and cloud, to get a complete picture of device connectivity patterns.

Resources

- Monitoring your IoT fleet using CloudWatch

- Building Observability in IoT applications using AWS Lambda Powertools, AWS X-Ray, Amazon CloudWatch
- Getting Aggregate Information of Devices with AWS IoT Device Management Fleet Indexing
- AWS IoT Core - Managing thing indexing
- Security monitoring for connected devices across OT, IoT, edge, cloud (TDR222)

Operate

In IoT, you must establish the right baseline metrics of behavior for your devices, be able to aggregate and infer issues that are occurring across devices, and have a robust remediation plan that is not only executed in the cloud, but also part of your device firmware. Operational health goes beyond the operational health of the cloud application and extends to the ability to measure, monitor, troubleshoot, and remediate devices that are part of your application, but are remotely deployed in locations that may be difficult or impossible to troubleshoot locally. This requirement of remote operations must be considered at design and implementation time in order to maintain your ability to inspect, analyze, and act on metrics sent from these remote devices. You also must plan for operational excellence by creating a streamlined process of functional testing that allows you to simulate how devices may behave in their various environments.

IOTOPS07: How do you assess whether your IoT application meets your operational goals?

Evaluating your operational goals enables you to fine-tune and identify improvements throughout the lifecycle of your IoT application. Measuring and extracting operational and business value from your IoT application allows you to effectively drive high-value initiatives.

IOTOPS07-BP01 Enable appropriate responses to events

Key operational data elements are those data points that convey some notion of operational health of your application by classifying events. Detecting operational events early can uncover unforeseen risks in your application and give your operations team a head start to help prevent or reduce business interruption. By defining a minimum set of logs, metrics, and alarms, your operations team can provide a faster incident response which reduces risks of business disruption.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTOPS07-BP01-01 *Configure logging to capture and store at least error-level events.*

- Use AWS IoT service logging options to capture error events in Amazon CloudWatch Logs
- Your devices create telemetry or diagnostic messages that are not stored in the registry or the device's shadow. Instead, these messages are delivered to AWS IoT using a number of MQTT topics. To make this data actionable, use the AWS IoT rules engine to route error messages to your automated remediation process and add diagnostic information to IoT messages. The rules engine inspects the status of a message and if it is an error, it starts the Step Function workflow to remediate the device based off the error message detail payload.

Prescriptive guidance IOTOPS07-BP01-02 *Create a dashboard for your responders to use in investigations of operational events to rapidly pinpoint the period of time when errors are logged.*

- Group clusters of error events into buckets of time to quickly identify when surges of errors were captured.
- Drill down into clusters of errors to identify any patterns to signal for triage response.

Prescriptive guidance IOTOPS07-BP01-03 *Configure an automated monitoring and alerting tool to detect common symptoms and warnings of operational issues.*

- For example, configure AWS IoT Device Defender to run a daily audit on at least the high and critical checks.
- Configure an Amazon SNS topic to notify a team email list, paging tool, or operations channel when AWS IoT Device Defender reports non-compliant resources in an audit.

For more information, see AWS IoT Device Defender Audit.

IOTOPS07-BP02 Use data-driven auditing metrics to detect if any of your IoT devices might have been broadly accessed

Monitor and detect the abnormal usage patterns and possible misuse of devices and automate the quarantine steps. Programmatic methods to detect and quarantine devices from interacting with cloud resources enable teams to operate a fleet in a scalable way while minimizing a dependency on active human monitoring.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTOPS07-BP02-01 *Use monitoring and logging services to detect anomalous behavior. Once you detect the compromised device, run programmatic actions to quarantine it.*

- Disable the certificate for further investigation and revoke the certificate to help prevent the device from any future use.
- Use AWS IoT CloudWatch metrics and logs to monitor for indications of misuse. If you detect misuse, quarantine the device so it does not impact the rest of the system.
- Use AWS IoT Device Defender to identify security issues and deviations from best practices

IOTOPS08: How do you segment your device operations in your IoT application?

You need to segment your device fleet to pinpoint operational challenges and direct incident response to the appropriate responder. Device fleet segmentation enables you to identify conditions under which devices operate sub optimally and minimize response time to security events.

IOTOPS08-BP01 Use static and dynamic device attributes to identify devices with anomalous behavior

Anomalies in fleet operations might only surface when analyzing metrics that aggregate across the boundaries of your static and dynamic groups or attributes. For example, devices that are running firmware version 2.0.10 and currently have a battery level over 50%. Static and dynamic groups allow for identifying and pinpointing devices in unique ways to monitor, analyze, and take corrective actions on device behavior.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTOPS08-BP01-01 *Pinpoint devices with unusual communication patterns.*

- Use a combination of static and dynamic groups of devices to perform fleet indexing to group devices and identify behavioral patterns—connectivity status, and message transmission.
- Use lifecycle events, device connectivity, and data transmission patterns to detect anomalies and pinpoint unusual behavior using techniques such as statistical anomaly detection (for large fleet of devices).

- Once abnormal behavior has been identified, move rogue and abnormal devices into a different group so that remedial policies can be assigned and implemented on them.

Resources

- AWS IoT Core - Authorization
- AWS IoT - Device Defender

Evolve

It is essential that you regularly provide time for analysis of operations activities, analysis of failures, experimentation, and making improvements. When incidents happen, you will want to make sure that your team learns from those incidents. You should analyze disruptive incidents to identify lessons learned and plan improvements. You will want to regularly review your lessons learned with other teams to validate your insights. It is recommended to have feedback loops mechanism such as retrospective analysis within the team to index on potential improvements, new ideas, things that could be discarded which does not add value to the business.

IOTOPS09: How do you evolve your IoT application with minimum impact to downstream IoT devices?

IoT solutions frequently involve a combination of low-power devices, remote locations, low bandwidth, and intermittent network connectivity. Each of those factors poses communications challenges, including upgrading firmware and edge applications. Therefore, it's important for you to incorporate and implement an IoT update process that minimizes the impact to downstream devices and operations. In addition to reducing downstream impact, devices must be resilient to common challenges that exist in local environments, such as intermittent network connectivity and power loss.

IOTOPS09-BP01 Run ops metrics analysis across business teams, document learnings and define action items for future firmware deployments

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTOPS09-BP01-01 *Monitor the behavior of devices as they are updated in the field, and proceed only after a percentage of devices have upgraded successfully.*

During upgrades, continue to collect all of the Amazon CloudWatch Logs, telemetry, and IoT device job messages and combine that information with the KPIs used to measure overall application health and the performance of long-running canaries.

Services like AWS IoT Device Defender are used to track anomalies in overall device behavior, and to measure deviations in performance that may indicate an issue in the updated firmware.

Prescriptive guidance IOTOPS09-BP01-02 *Use AWS IoT Device Management for creating deployment groups of devices and delivering over the air updates (OTA) to specific device groups.*

Use a combination of grouping IoT devices for deployment and staggering firmware upgrades over a period of time. In AWS IoT, thing groups allow you to manage devices by category. Groups can also contain other groups — allowing you to build hierarchies. With organizational structure in your IoT application, you can quickly identify and act on related devices by device group. Leveraging the thing group allows you to automate the addition or removal of devices from groups based on your business logic and the lifecycle of your devices.

Prescriptive guidance IOTOPS09-BP01-3 *Use dynamic thing group as a target for OTA updates.*

Create a continuous job with a dynamic thing group as target allows you to automatically target devices when they meet the desired criteria. The criteria can be the connectivity state or criteria stored in registry or shadow such as software version or model. If a thing doesn't appear in the dynamic thing group, it won't receive the job document from the job.

For example, if your device fleet requires a firmware update to minimize the risk of interruption during the update process, and you only want to update the firmware on devices with a battery life greater than 80%. You can create a dynamic thing group called 80PercentBatteryLife that only includes devices with a battery life above 80% and use it as the target for your job. Only devices that meet your battery life criteria will receive the firmware update. As devices reach the 80% battery life criteria, they are automatically added to the dynamic thing group and will receive the firmware update.

IOTOPS10: How do you verify that you are ready to support the operations of devices in your IoT workload?

Operating IoT workloads at scale is different than testing and running prototypes. You need to make sure that your team is prepared and trained to operate a widely distributed IoT data collection application. IoT workloads require your teams to learn new skills and competencies to deliver edge-to-cloud outcomes. Your team needs to be able to pinpoint key operational thresholds that indicate a high level of readiness.

IOTOPS10-BP01 Train team members supporting your IoT workloads on the lifecycle of IoT applications and your business objectives

Key team members responsible for IoT workloads are trained on major IoT lifecycle events: onboarding, command and control, security, data ingestion, integration, and analytics services. Team members should be able to identify key operational metrics and know how to apply incident response measures. Training team members on the basics of IoT lifecycles and how these align with business objectives provides actionable context on failure scenarios, mitigation strategies, and defining lasting processes that effectively contribute to fewer operational events and less severe impact during events.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTOPS10-BP01-01 *Build IoT operational expertise by having team members and architects' complete reviews of common IoT architectural patterns, best practices, and educational courses.*

- Introduce new team members to IoT lifecycles with onboarding checklists that include at least one educational course.
- Introduce new team members with onboarding checklists that include a step to review, validate, and submit updates to your IoT application architecture documentation and operational monitoring plan.

Prescriptive guidance IOTOPS10-BP01-02 *Author runbooks for each component of the architecture and train team members on their use.*

Include guidance for a response procedure for remote devices that are no longer online. Apply recovery commands for troubleshooting remote devices that are faulty but still online.

Key AWS services

Several services can be used to drive operational excellence for your IoT application. The AWS Device Qualification Program helps you select hardware components that have been designed and tested for AWS IoT interoperability. Qualified hardware can get you to market faster and reduce operational friction. AWS IoT Core offers features used to manage the initial onboarding of a device. With AWS IoT Greengrass, you can run custom computes, such as AWS Lambda functions and Docker containers on the device to respond quickly to local events, interact with local resources, and process data to minimize the cost of transmitting data to the cloud and simplify remote application management. AWS IoT Device Management reduces the operational overhead of performing fleet-wide operations, such as device grouping and searching. In addition, Amazon CloudWatch is used for monitoring IoT metrics, collecting logs, generating alerts, and triggering responses. Other services and features that support the four areas of operational excellence are as follows:

- **Preparation: AWS IoT Device Tester (IDT) for FreeRTOS and AWS IoT Greengrass** is a test automation tool for connected devices to determine if your devices running FreeRTOS or AWS IoT Greengrass can interoperate with AWS IoT Services.
- **Preparation: AWS IoT Core** supports provisioning and onboarding your devices in the field, including registering the device identity using just-in-time provisioning, just-in-time registration, or Multi-Account Registration. Devices can then be associated with their metadata and device state using the device registry and the Device Shadow.
- **Operations**: **AWS IoT thing registry groups and Fleet Indexing** allow you to quickly develop an organizational structure for your devices and search across the current metadata of your devices to perform recurring device operations. Amazon CloudWatch allows you to monitor the operational health of your devices and your application.
- **Operations: AWS IoT Greengrass** provides many pre-built capabilities on the device to help you focus mostly on their business logic and offers many foundational infrastructure and operation features such as remote application management.
- **Responses**: **AWS IoT Jobs** enables you to proactively push updates to one or more devices such as firmware updates or device configuration. AWS IoT rules engine allows you to inspect IoT messages as they are received by AWS IoT Core and immediately respond to the data, at the most granular level. AWS IoT Device Defender enable you to proactively trigger notifications or remediation based on real-time analysis, real-time security and data thresholds with Device Defender.

Resources

Refer to the following resources to learn more about our best practices for operations:

- Empowering operations: A scalable remote asset health monitoring solution
- Monitor AWS IoT connections in near-real time using MQTT LWT
- How to manage IoT device certificate rotation using AWS IoT
- Configuring near real-time notification for asset-based monitoring with AWS IoT Events
- Monitor AWS IoT using CloudWatch Logs
- MQTT design best practices
- Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core

Security

The security pillar includes best practices to help you protect information, systems, and assets while delivering business value. This section provides in-depth, best-practice guidance for architecting secure IoT workloads on AWS.

Focus areas

Design principles

In addition to the overall Well-Architected Framework security design principles, there are specific design principles for IoT security:

- **Manage device security lifecycle holistically**: Device security starts at the design phase, and ends with the retirement, re-purposing, and destruction of the hardware and data. It is important to take an end-to-end approach to the security lifecycle of your IoT solution in order to maintain your competitive advantage and retain customer trust.
- **Verify least privilege permissions**: Devices should all have fine-grained access permissions. For example, while using MQTT communications, use device instance specific MQTT topics

and access permissions which limit which MQTT topics a device can use for communication. By restricting access, a compromised device will less chance of affecting other devices. This reduces the risk of your application. There can be a trade-off when a large numbers of devices are in use. In such cases, select MQTT topic spaces and access permissions to limit the the scope of affected devices if a device is compromised.

- **Secure device credentials at rest**: Devices should securely store credential information at rest using mechanisms such as a dedicated secure cryptographic element built into the device or secure flash.
- **Implement device identity lifecycle management**: Devices maintain a device identity from creation through end of life (retirement or destruction). A well-designed identity system will keep track of a device's identity, track the validity of the identity, and proactively adjust or revoke IoT permissions over time. Device identity lifecycle management will enable re-purposing or re-commissioning devices, either within the same application or account or in a separate, un-related application or account.
- **Take a holistic view of data security**: IoT deployments involving a large number of remotely deployed devices extend the risks of unintended data exposure as well as data privacy. Use a model such as the Open Trusted Technology Provider Standard to systemically review your supply chain and solution design for risk and then apply appropriate mitigations.
- **Preserve safety and reliability when used in critical OT/IIoT environments**: IIoT cybersecurity differs from the IT cybersecurity model because it is not only concerned with data protection, but also with the preservation of safety and reliability of production systems and the environment. When devices are used in OT/IIOT environments it is also necessary to adhere to environmental health and safety (EHS) requirements.
- **Implement Zero Trust Principles (ZTP) following NIST SP 800-207**: Zero trust is not limited to traditional IT. Zero Trust Principles extend to the internet of things (IoT, operational technology (OT), and industrial IoT (IIoT). A zero-trust model can significantly improve an organization's security posture by alleviating the sole reliance on network or perimeter-based protection. This does not mean getting rid of network or perimeter security altogether. Where possible, use identity and network capabilities together to protect core assets and apply zero trust principles working backwards from specific use cases with a focus on extracting business value and achieving measurable business outcomes.
- **Establish secure connection with AWS through Site-to-Site VPN or AWS Direct Connect from the industrial edge:** For IIoT workloads, AWS offers multiple ways (For more information see Network-to-Amazon VPC connectivity options) and design patterns to establish a secure connection to the AWS environment from the industrial edge. Establish a secure VPN connection

to AWS over the public internet or set up a dedicated private connection via AWS Direct Connect. Use AWS VPN with Direct Connect to encrypt traffic over Direct Connect.

- **Use VPC Endpoints whenever possible:** For IIoT workloads, once a secure connection to AWS has been established via VPN over public Internet or AWS Direct Connect, use VPC Endpoints whenever possible. VPC Endpoints enable customers to privately connect to supported regional AWS services without requiring or using the public IP address of those AWS service endpoints. Endpoints also support endpoint policies. Endpoint policies can be used to further limit access to AWS services through the network configuration for the solution.
- **Use secured and encrypted connections to AWS services over public internet paths:** Use HTTPS and MQTTS for devices to communicate with AWS services. If a VPC Endpoint for the required service is not available, establish a secure connection over the public Internet. If TLS-protected communications are not supported by the device itself, it is a best practice to route un-protected communications through a TLS proxy and a firewall. Using a proxy allows the cloud traffic to be inspected and monitored enabling threat and malware detection. This also allows the security policies to be applied at the network layer. Firewall rules can be established for HTTPS and MQTT traffic to securely connect to AWS IoT services over the public internet. **Note:** AWS IoT Core requires secure communications to API and Data endpoints. For more information, see Device communication protocols.
- **Use secure protocols:** In most environments, prefer to use secure protocols which support end-to-end encryption. When using secure protocols is not an option, tighten the trust boundaries as described in the next point. For example, secure protocols such as HTTP over TLS (HTTPS) and MQTT over TLS using mutually-authenticated TLS (mTLS).
- **Use network segmentation and tighten trust boundaries:** Follow a micro segmentation approach. Build small islands of components within a single network that communicate only with each other and control the network traffic between segments. Select the newer version of industrial protocols which offer security features and configure the highest level of encryption available when using industrial control system (ICS) protocols such as CIP Security, Modbus Secure and OPC UA. When using secure industrial protocols is not an option, tighten the trust boundary using a protocol converter to translate the insecure protocol to a secure protocol as close to the data source as possible.
  - Alternatively, separate the plant network into smaller cell or area zones by grouping ICS devices into functional areas. This separation enables the limiting of scope and area of insecure communications. Consider using unidirectional gateways and data diodes for one-way data flow where appropriate. Consider using specialized firewall and inspection products that

understand ICS protocols to inspect traffic entering and leaving cell or area zones and can detect anomalous behavior in the control network.

- **Securely manage and access edge computing resources:** Keeping computing resources at the industrial edge up to-date, securely accessing to them for configuration and management, and automatically deploying changes can be challenging. AWS provides options to securely manage edge compute resources (AWS Systems Manager), IoT resources (IoT Device management, AWS IoT Greengrass) and also provides fully managed infrastructure services (AWS Outposts, Amazon EKS Anywhere, AWS Snowball) to make it straightforward to consistently apply best practices to all resources.
- **Shared Responsibility Model:** Be aware of and follow the guidance provided in the Shared Responsibility Model. Understand your responsibilities for the security of resources and data in the cloud as well as AWS's responsibility for security of the cloud. The shared responsibility model also extends into environments where devices are deployed. In the environments where devices are deployed, often called IoT edge locations, your responsibilities are much broader than the cloud environment. Security of edge environments is your responsibility and includes securing the edge network, edge network perimeter, devices in the edge network, securely connecting to the cloud, handling software updates of edge equipment and devices, and edge network logging, monitoring, and auditing, as examples. AWS is responsible for the AWS provided edge software such as AWS IoT Greengrass and AWS IoT SiteWise Edge and AWS edge infrastructure such as AWS Outposts and AWS Snowball.

Definitions

There are nine focus areas for security. These are inspired by the AWS Cloud Adoption Framework. For more information, see Security Perspective: compliance and assurance.

- Identity and access management (IAM)
- Detective controls
- Infrastructure protection
- Data protection
- Incident response
- Application security
- Vulnerability management
- Security governance

- Security assurance

These focus areas encompass IoT device hardware, as well as the end-to-end solution. IoT implementations require expanding your security model to make sure that devices implement both hardware and software security best practices and your IoT applications follow security best practices for factors such as adequately scoped device permissions and detective controls.

The security pillar focuses on protecting information and systems. Key topics include confidentiality, integrity, and availability of data, identifying and managing who can do what with privilege management, protecting systems, and establishing controls to detect and respond to security events. Privilege management is part of authentication, authorization, administration, and auditing (AAAA).

Each of the following sections presents IoT-centric information and recommendations for each of the nine security best practice areas. In each section, a description is provided followed by a list of relevant questions to prompt assessment of an environment and solution.

Identity and access management

IoT devices frequently face elevated security risks. There are several reasons for this. Devices are often provisioned with a trusted identity. Devices may store or have access to strategic customer or business data (such as the firmware itself). Devices may be remotely accessible over the internet. Devices may be subject to to direct physical tampering. To provide protection against unauthorized access, you need to always begin with implementing security at the device level. From a hardware perspective, there are several solutions that you can implement to reduce the risk of tampering with sensitive information on the device such as:

- Hardware crypto modules
- Software-supported solutions including secure flash
- Physical function modules that cannot be cloned
- Up-to-date cryptographic libraries and standards including PKCS #11 and TLS 1.2/1.3

To secure device hardware, you should implement solutions so that private keys and sensitive device identity information are unique to a device and stored only in a secure hardware location on the device. You should implement unique device identities using public-private key pairs. An X.509 certificate containing a public key represents the identity of the device. The corresponding

private key is used by the device to prove that it corresponds to that identity. Implement hardware or software-based modules that securely store and manage access to the device's private key corresponding to its public key and X.509 certificate. FreeRTOS, AWS IoT Greengrass, and the AWS IoT Device SDKs support this through the use of PKCS#11. In addition to hardware security, IoT devices must be given a valid identity, which will be used for authentication and authorization in your IoT application.

During the lifetime of a device, you will need to be able to manage certificate request, renewal and revocation, as well as update device firmware and software. To handle these changes, you must first have the ability to update a device in the field. The ability to perform firmware updates, software updates and configuration updates on hardware is a vital underpinning to a well-architected IoT application. Through over the air (OTA) updates, you can securely rotate device certificates before expiry. OTA can also be used to update trusted certificate authorities and update firmware and software.

For example, with AWS IoT, you first provision a X.509 certificate identifying the device and then separately create the IoT permissions for connecting to AWS IoT Core, publishing and subscribing to messages, and receiving updates. This separation of identity and permissions provides flexibility in managing your device security. During the configuration of permissions, you can make sure that the device has the correct identity (authentication) as well as the right level of access control (authorization) by creating an IoT policy that restricts access to MQTT actions for each device.

Make sure that each device has its own unique X.509 certificate. Each device should have a unique public-private key pair. The private key should, if possible, be generated on the device and never leave the secure storage area where it was generated on the device. Devices should never share keys or certificates used for identification (one certificate for one device rule). In addition to using a single key or certificate per device, each device must have its own unique thing identifier in the IoT registry. In AWS IoT Core, by default, the thing name is used as the basis for the MQTT ClientID for MQTT connect. It is possible to separate Thing name and ClientID if necessary.

Every X.509 certificate has an expiration. This includes certificates representing certificate authorities, including root certificate authorities. Devices must support the replacement of trusted root and intermediate certificate authorities. Devices must also support refresh and replacement of the certificate, which represents the identity of the device. This helps to support continued operation of the device including the ability to authenticate to AWS IoT Core.

By creating this association where a unique key or certificate is paired with each thing in AWS IoT Core, you can make sure that one compromised certificate cannot inadvertently assume an identity of another device. It also assists in logging (auditing), troubleshooting and remediation of

incidents. If the MQTT ClientID and the thing name are different, you must plan to have a way to correlate between thing name and ClientID. This is necessary to work with log messages associated with device communication.

To support device identity updates, use AWS IoT Jobs. AWS IoT Jobs is a managed service for distributing OTA updates and binaries to your devices. AWS IoT Jobs is used to define a set of remote operations that are sent to and executed on one or more devices connected to AWS IoT Core. AWS IoT Jobs by default integrate several best practices, including mutual authentication and authorization, device tracking of update progress, logging or auditing, and fleet-wide wide metrics for jobs that are scheduled to be run across fleets of devices.

Use native provisioning mechanisms to onboard devices when they already have a device key or certificate provisioned onto them. For example, you can use just-in-time provisioning (JITP) or just-in-time registration (JITR) that provisions devices when they first connect to AWS IoT Core.

If the devices cannot use X.509 certificates, or you have an existing fleet of devices with a proprietary authentication or access control mechanism, use custom authentication or authorization mechanisms. One example of this is the use of bearer tokens such as OAuth over JWT or SAML 2.0 tokens for authenticating communications. For example, a device which sends a JSON web token (JWT) generated by their identity provider in the HTTP header or query string when it attempts to connect to AWS IoT Core. The signature, validity, issuer, and audience of the JWT (in addition to other custom claims) must be validated by an AWS IoT custom authorizer before the connection is established.

It is important to use a standard set of naming conventions when designing device names and MQTT topics. For example, we recommend using the same client identifier for the device as the IoT Thing Name. This will also allow to include relevant routing information for the device in the topic namespace.

Enable AWS IoT Device Defender audits to track device configuration, device policies, and checking for expiring certificates in an automated fashion. For example, AWS IoT Device Defender can run audits on a scheduled basis and initiate a notification for expiring certificates. With the combination of receiving notifications of revoked certificates or certificates nearing expiration, you can automatically schedule an OTA update using AWS IoT Jobs that can proactively rotate or refresh the certificate.

IOTSEC01: How do you associate IoT identities and permissions with your devices?

Your application is responsible for managing how your devices authenticate and authorize their interactions. By creating a process that makes sure devices have a unique identity and identity-based permissions for accessing the IoT system, you establish the greatest control for managing device interactions.

IOTSEC01-BP01 Assign unique identities to each IoT device

When a device connects to other devices or cloud services, it must establish trust by authenticating using credentials such as X.509 certificates (with proof of having the private key) or security tokens. You can find available options from the IoT solution of your choice, and implement device registry and identity stores to associate devices, metadata and user permissions. The solution should enable each device (or Thing) to have a unique name (or ThingName) in the device registry, and the solution should make sure that each device has an associated unique identity principal, such as an X.509 certificate or security token. Identity principals, such as certificates, should not be shared between devices. Detection of multiple devices using the same identity credentials indicates an issue that requires investigation. This could be due to improper device setup or potential unauthorized cloning of device credentials.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC01-BP01-01 *Use X.509 client certificates to authenticate over TLS 1.2/1.3.*

We recommend that each device be given a unique key or certificate to enable fine-grained device management, including certificate revocation. Devices must support rotation, refresh, and replacement of certificates to support continued operation. For example, AWS IoT Core supports AWS IoT-generated X.509 certificates or your own X.509 certificates for device authentication.

Prescriptive guidance IOTSEC01-BP01-02 *Choose the appropriate certificate vending mechanisms for your use case.*

We recommend using native provisioning mechanisms to onboard devices when they already have a device certificate (and associated private key) on them. For example, you can use just-in-time provisioning (JITP) or just-in-time registration (JITR) that provisions devices when they first connect to AWS IoT.

Prescriptive guidance IOTSEC01-BP01-03 *Use security bearer tokens only if necessary.*

If the devices cannot use X.509 certificates, or you have an existing fleet of devices with a proprietary access control mechanism that requires use of bearer tokens such as OAuth with JWT

or SAML 2.0 tokens, use custom authentication mechanisms. For example, when a device attempts to connect to AWS IoT, it sends a JWT which was generated by their identity provider in the HTTP header or query string of requests. The token is validated by AWS IoT custom authorizer and the connection is established. All communications must be performed over TLS-protected (encrypted) connections using strong cipher-suites.

Prescriptive guidance IOTSEC01-BP01-04 *Use a consistent naming convention that aligns MQTT topics with your device identity.*

It is important to use a standard set of naming conventions when designing device names and MQTT topics. For example, we recommend using the same client identifier (`ClientId`) for the device as the IoT Thing Name (`ThingName`). This will also simplify the design for including relevant routing information for the device in the topic namespace.

IOTSEC02: How do you secure your devices and protect device credentials?

Your IoT devices and identity credentials (certificates, private keys, or tokens) must be secured throughout their lifecycle. To support device authenticity, your IoT hardware must securely store, manage, and restrict access to the identities that the device uses to authenticate itself with the cloud. By securing your devices and storing your device credentials safely, you can reduce the risk of unauthorized users misusing device credentials.

IOTSEC02-BP01 Use a separate hardware or a secure area on your devices to store credentials

A secure element (SE) is any hardware feature you can use to protect information on the device. A common use of an SE is to securely store the device's identity. Secure storage at rest helps reduce the risk of unauthorized use of the device identity. Never store or cache device credentials outside of the SE. If supported, generate public or private key pairs using the SE, and generate the Certificate Signing Requests (CSRs) on the device. With this method, the private key never leaves the SE. If this is not possible, generate and transmit the credentials to the SE in a secure manufacturing facility with Common Criteria EAL certification. Import or install the private key material into the SE in the secure facility. Securely handling a device's identity helps make sure that your hardware and application are resilient to potential security issues that occur in unprotected systems. A SE provides encrypted storage of private information (such as cryptographic keys) at

rest and can be implemented as separate specialized hardware or as part of a system on a chip (SoC).

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC02-BP01-01 *Use tamper-resistant hardware that offloads the cryptographic operations for encryption and communication from the IoT application.*

Device credentials must always reside in a SE, which facilitates usage of the credentials. Using the SE to facilitate the use of device credentials further limits the risk of unauthorized use. As an example, AWS IoT Greengrass supports using a SE to store AWS IoT certificates and private keys.

Prescriptive guidance IOTSEC02-BP01-02 *Use cryptographic API operations provided by the secure element hardware for protecting the secrets on the device.*

Only access security modules using the latest security protocols. For example, in FreeRTOS, use the PKCS#11 APIs provided in the corePKCS11 library for protecting secrets.

Prescriptive guidance IOTSEC02-BP01-03 *Use the AWS Partner Device Catalog to find AWS Partners that offer hardware security modules.*

If you are getting devices that have not been deployed in the field, AWS recommends reviewing the AWS Partner Device Catalog to find AWS IoT hardware partners that either implement a SE or trusted platform module (TPM). Use AWS IoT Partners that offer qualified SEs for storing IoT device identities.

IOTSEC02-BP02 Use a trusted platform module (TPM) to implement cryptographic controls

Generally, a TPM is used to hold, secure, and manage cryptographic keys and certificates for services such as disk encryption, Root of Trust booting, verifying the authenticity of hardware (as well as software), and password management. The TPM has the following characteristics:

1. TPM is a dedicated crypto-processor to help make sure the device boots into a secure and trusted state.
2. The TPM chip contains the manufacturer's keys and software for device encryption.
3. The Trusted Computing Group (TCG) defines hardware-roots-of-trust as part of the Trusted Platform Module (TPM) specification.

A hardware identity refers to an immutable, unique identity for a platform that is inseparable from the platform. A hardware embedded cryptographic key, also referred to as a hardware root of trust, can be an effective device identifier. Vendors such as Microchip, Texas Instruments, and many others have TPM-based hardware solutions.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC02-BP02-01 *Perform cryptographic operations inside the TPM to avoid a third party gaining unauthorized access.*

Store all secret keys from the manufacturer required for secure boot, such as attestation keys, storage keys, and application keys, in the secure enclave of the chip. For example, a device running AWS IoT Greengrass can be used with an Infineon OPTIGA TPM.

Prescriptive guidance IOTSEC02-BP02-02 *Use a trusted execution environment (TEE) along with a TPM to act as a baseline defense against rootkits.*

TEE is a separate execution environment that provides security services and isolates access to hardware and software security resources from the host operating system and applications. Various hardware architectures support TEE such as:

1. ARM TrustZone divides hardware into secure and non-secure worlds. TrustZone is a separate microprocessor from the non-secure microprocessor core.
2. Intel Boot Guard is a hardware-based mechanism that provides a verified boot, which cryptographically verifies the initial boot block or uses a measuring process for validation.

Prescriptive guidance IOTSEC02-BP02-03 *Use physical unclonable function (PUF) technology for cryptographic operations.*

A PUF technology is a physical object that provides a physically defined digital fingerprint to serve as a unique identifier for an IoT device. As a different class of security primitive, PUFs normally have a relatively simple structure. It makes them ideal candidates for affordable security solutions for IoT networks. Generally, a hardware root of trust based on PUF is virtually impossible to duplicate, clone, or predict. This makes them suitable for applications such as secure key generation and storage, device authentication, flexible key provisioning, and chip asset management. Refer to AWS Partner Device Catalog which has various device solutions with PUFs such as LPC54018 IoT Solution by NXP.

IOTSEC02-BP03 Use protected boot and persistent storage encryption

When a device performs a secure boot, it validates that the device is not running unauthorized code from the filesystem. This helps make sure that the boot process starts from a trusted combination of hardware and software, and continues until the host operating system has fully booted and applications are running.

Choose devices with TPM (or TEE) for new deployments. Secure boot also makes sure that if even a single bit in the software boot-loader or application firmware is modified after deployment, the modified firmware will not be trusted, and the device will refuse to run this untrusted code.

Full disk encryption makes sure that the storage and cryptographic elements are secured in absence of a TPM or SE. The disk controller needs to make sure that read accesses to the disk are transparently decrypted and write operations are encrypted at runtime.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC02-BP03-01 *Boot devices using a cryptographically verified operating system image.*

Use digitally signed binaries that have been verified using an immutable root of trust, such as a master root key (MRK) that's stored securely in a non-modifiable memory, to boot devices.

Prescriptive guidance IOTSEC02-BP03-02 *Create separate filesystem partitions for the boot-loader and the applications.*

As an example, configure the device boot-loader to use a read-only partition, and applications to use a separate writable partition for separation of concerns and reducing risks.

Prescriptive guidance IOTSEC02-BP03-03 *Use encryption utilities provided by the host operating system to encrypt the writable filesystem.*

For example, use crypt utilities for Linux such as dm-crypt or GPG, and use BitLocker or Amazon EFS for Microsoft Windows.

Prescriptive guidance IOTSEC02-BP03-04 *Use services that can push signed application code from a trusted source to the device.*

You can use AWS IoT Jobs to push signed software binaries from the cloud to the device.
For microcontrollers using FreeRTOS, make sure that the firmware images are signed before

deployment. Signature verification should also verify that the signer of the package is trusted and the signer's certificate and any intermediate certificate authorities' certificates have not been revoked.

IOTSEC03: How do you authenticate and authorize user access to your IoT application?

Although many applications focus on the device aspect of IoT, in almost all industries using IoT, there is also a human component that needs the ability to communicate to and receive notifications from devices.

For example, consumer IoT generally requires users to onboard their devices by associating them with an online account. Industrial IoT typically entails the ability to analyze hardware telemetry in near real time. In either case, it is essential to determine how your application will identify, authenticate, and authorize users that require the ability to interact with the solution and with particular devices.

Controlling user access to your IoT assets begins with identity. Your IoT application must have in place a store (typically a user registry or identity provider) that keeps track of a user's identity and also how a user authenticates using that identity. The identity store may include additional user attributes that can be used at authorization time. For example, the user's group memberships.

When using AWS to authenticate and authorize IoT application users, you have several options to implement your identity store and how that store maintains user attributes. For your own applications, use Amazon Cognito for your identity store. Amazon Cognito provides a standard mechanism to express identity and to authenticate users. Amazon Cognito enables usage of user identity in a way that can be directly consumed by your app and other AWS services in order to make authorization decisions. When using AWS IoT, you can choose from several identity and authorization services including Amazon Cognito Identity Pools, AWS IoT policies, and AWS IoT custom authorizer to validate tokens (such as JWT or SAML 2.0) for authenticating users. Amazon Cognito supports federation to third party identity providers using the OpenID Connect (OIDC) and SAML 2.0 protocols.

An alternative to using AWS IAM-based role permissions for user interaction with an IoT solution is to define a separate authorization layer for the application which uses Amazon Verified Permissions. Verified Permissions allows for the definition of fine-grained access control policies for accessing resources which are specific to an application. Principals used in AVP policy statements can be defined from Amazon Cognito user pools.

IOTSEC03-BP01 Implement authentication and authorization for users accessing IoT resources

This practice provides users with secure access to connected IoT devices and equipment through different channels such as web or mobile devices. Without valid authentication and authorization, devices can be subjected to risks of unauthorized access.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC03-BP01-01 *Implement an identity store to authenticate users of your IoT application.*

Implement an identity and access management solution for end users. This solution should allow end users with temporary, role-based credentials to access the IoT solution. For example, you can use a service like Amazon Cognito to create user pools for authentication. Or, you can use Amazon Cognito integration with SAML 2.0 or OAuth 2.0 compliant identity providers for authentication as well. If you host your own identity store, use AWS IoT custom authorizers to validate tokens such as JWT and SAML 2.0 for authenticating users. AVP can be used to define fine-grained access control policies to specify who (identity) can perform what actions on which resources (application, data, and devices).

Prescriptive guidance IOTSEC03-BP01-02 *Grant least privilege access*

Authorization is the process of granting permissions to perform some operation or access some information by an authenticated identity. If using AWS IAM credentials, you grant permissions to your users in AWS IoT Core using data plane and control plane IAM policies through the Identity broker. AWS IoT control plane APIs allow you to perform administrative tasks like creating or updating certificates, things, and rules. AWS IoT data plane APIs allow you send data to and receive data from AWS IoT Core.

For example, if you are using Amazon Cognito, use federated identities for user authentication and Amazon Cognito identity pools to establish IAM credentials. If you are using a different Identity provider than Amazon Cognito, use AWS IoT custom authorizers to invoke lambda functions that will create the required IAM policies. To define fine-grained permissions for Amazon Cognito users, use AVP to define authorization policies in which the principal is the Amazon Cognito user or group.

Prescriptive guidance IOTSEC03-BP01-03 *Adopt least privilege when assigning user permissions.*

Adopt the least privilege principle and assign only the minimum required permissions to each IAM role that is used in the solution. This applies to both user and service (For example, Lambda function) roles that are defined in the solution.

For example, with Amazon Cognito Identity Pools this can be achieved by setting up role-based access through IAM policies for authenticated users like consumers or administrators as well as unauthenticated users. Consumers or unauthenticated users should not be allowed to run adminstrative actions against IoT services, such as detaching policies, deleting certificate authorities (CAs), or deleting certificates.

IOTSEC03-BP02 Decouple access to your IoT infrastructure from the IoT applications

For implementing the decoupled view of telemetry for your users, use a mobile service such as AWS AppSync or Amazon API Gateway. With both of these AWS services, you can create an abstraction layer that decouples your IoT data stream from your user's device data notification stream. By creating a separate view of your data for your external users in an intermediary datastore, you can use AWS AppSync to receive user-specific notifications based only on the allowed data in your intermediary store. Examples of intermediate data stores are Amazon DynamoDB, Amazon Relational Database Service, and Amazon OpenSearch Service. In addition to using external data stores with AWS AppSync, you can define user specific notification topics that can be used to push specific views of your IoT data to your external users.

If an external user needs to communicate directly to an AWS IoT endpoint, make sure that the user identity is either an authorized Amazon Cognito Federated Identity that is associated to an authorized Amazon Cognito Identity Pool role and a fine-grained IoT policy, or uses AWS IoT custom authorizer, where the authorization is managed by your own authorization service. With either approach, associate a fine-grained policy to each user that limits what the user can connect as, publish to, subscribe from, and receive messages from concerning MQTT communication.

By decoupling the IoT infrastructure from the end-user IoT applications, you can build an additional layer of security and reliability.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTSEC03-BP02-01 *Use an API layer between the application and IoT layer.*

Build an application interface layer to insulate the IoT data plane from end users. Fundamentally, the primary interface to IoT data plane is MQTT topics. Protecting the data plane essentially means protecting the MQTT topics from unwanted communication.

For example, use Amazon API Gateway or AWS AppSync to provide a REST or GraphQL API interface between the end user application and the IoT layer. With this design, an API implementation, often built using an AWS Lambda function, would communicate with devices using the IoT data plane. This insulates the operations on the IoT data plane from the end user interaction performed at the REST or GraphQL API interfaces. Fine-grained permissions for using the API interfaces can be defined using AVP.

IOTSEC04: How do you apply least privilege to principals that interact with your IoT applicati on?

After registering a device and establishing its identity, it may be necessary to add additional device information needed for monitoring, metrics, telemetry, or command and control. Each resource (for example, device, background task, service, API, and user) requires its own assignment of access control rules. By reducing the actions that a device or user can take in your application, and making sure that each resource is secured separately, you reduce the risks to your system in case a single identity is compromised.

In AWS IoT, create fine-grained permissions by using a consistent set of naming conventions in the IoT registry. The first convention is to use the same unique identifier for a device. Match the MQTT ClientID and AWS IoT thing name. By using the same unique identifier in all these locations, you can easily create an initial set of IoT permissions that can apply to your devices using AWS IoT Thing Policy variables. The second naming convention is to embed the unique identifier of the device into the device certificate. Continuing with this approach, store the unique identifier as the `CommonName` naming element in the subject name of the certificate so that Certificate Policy VariablesX.509 Certificate AWS IoT Core policy variables can be used to bind IoT permissions to each unique device principal.

By using policy variables, you can create a few IoT policies that can be applied to your device certificates while maintaining least privilege. For example, the IoT policy below would restrict devices to connect only using the unique identifier of the device which is stored in the common name as its MQTT ClientID (see policy variable inserted into Resource name) and only if the certificate is attached to the device. This policy also restricts a device to only publish on its individual shadow (see policy variable inserted into MQTT topic):

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Certificate.Subject.CommonName}"
            ],
            "Condition": {
                "Bool": {
                    "iot:Connection.Thing.IsAttached": [
                        "true"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/update"
            ]
        }
    ]
}
```

Attach your device identity (certificate or Amazon Cognito Federated Identity) to the thing in the AWS IoT registry using `AttachThingPrincipal` and attach the policy to principals using `AttachPolicy`.

Although these scenarios apply to a single device communicating with its own set of topics and device shadows, there are scenarios where a single device needs to act upon the state or topics of other devices. For example, you may be operating an edge appliance in an industrial setting, creating a home gateway to manage coordinating automation in the home, or allowing a user to gain access to a different set of devices based on their specific role. For these use cases, leverage a known entity, such as a group identifier or the identity of the edge gateway as the prefix for all of the devices that communicate to the gateway. By making all of the endpoint devices use the same

prefix, you can make use of wildcards, "*", in your IoT policies. This approach balances MQTT topic security with manageability.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["iot:Publish"],
        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
edgegateway123-*/shadow/update"]
    }]
}
```

In the preceding example, the IoT operator would associate the policy with the edge gateway with the identifier, edgegateway123. The permissions in this policy would then allow the edge appliance to publish to other Device Shadows that are managed by the edge gateway. This is accomplished by enforcing that connected devices to the gateway all have a thing name that is prefixed with the identifier of the gateway. For example, a downstream motion sensor would have the identifier `edgegateway123-motionsensor1`, and therefore can now be managed by the edge gateway while still restricting permissions.

IOTSEC04-BP01 Assign least privilege access to devices

Permissions (or policies) allow an authenticated identity to perform various control plane and data plane operations against AWS IoT Core, such as creating devices or certificates via the control plane, and connecting, publishing, or subscribing via the data plane.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC04-BP01-01 *Grant least privilege access to reduce the scope and impact of the potential events.*

Use granular device permissions to enable least privilege access, which can help limit the impact of an error or misconfiguration. Define a mechanism so that devices can only communicate with specific resources such as MQTT topics. If permissions are generated dynamically, make sure that similar practices are followed. For example, create an AWS IoT policy as a JSON document that contains a statement with the following:

1. `Effect`, which specifies whether the action is allowed or denied.
2. `Action`, which specifies the action the policy is allowing or denying.

3. `Resource`, which specifies the resource or resources upon which the action is allowed or denied.

Prescriptive guidance IOTSEC04-BP01-02 *Consider scaling granular permissions across the IoT fleet.*

Reuse permissions across principals where possible rather than creating specific permissions for each individual principal. This helps you avoid creating redundant permissions per device and streamlines applying similar permission changes across multiple devices.

For example, consider an AWS IoT policy allows access based on various thing attributes such as `ThingName`, `ThingTypeName`, and `Thing Attributes`. To allow a device to access its own information, use a policy variable rather than specifying a specific ClientId value and creating a IoT policy for each device.

- Recommended: `arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}`
- Not recommended: `arn:aws:iot:us-east-1:123456789012:client/foo.`

As another example, consider an AWS IoT policy also allows access based on various certificate attributes such as `Subject`, `Issuer`, `Subject Alternate Name`, `Issuer Alternate Name`, and others. Again, use a policy variable rather than specifying a specific `CertificateId` and creating a IoT policy for each device.

- Recommended: `arn:aws:iot:us-east-1:123456789012:topic/${iot:CertificateId}`
- Not recommended: `arn:aws:iot:us-east-1:123456789012:topic/xxxxxxxxxxx`

IOTSEC05: How do you manage device certificates, including installation, validation, revocation, and rotation?

To authenticate IoT device to AWS IoT Core and authenticate AWS IoT Core to an IoT device, AWS IoT Core supports TLS-based mutual authentication using X.509 certificates. TLS-based mutual

authentication authenticates both client and server to one another during the TLS handshake processing of setting up an encrypted TLS communications channel.

To enable TLS-based mutual authentication, device makers must provision a unique identity, including a unique private key and X.509 certificate, into each device. Certificates are relatively long-lived identifiers of principals and are managed using a customer-owned Certificate Authority (CA), a third-party CA, or the AWS IoT Core CA. Any hosted CA chosen must provide you the ability to create (activate), revoke (deactivate), validate, and refresh or rotate certificates. Authentication using certificates requires, at authentication time, that the principal identified by the certificate can prove that it holds the private key associated with the public key found in the certificate.

Authenticated identities are the focal point of device trust and authorization to your IoT application. It's vital to be able to manage identities, such as certificates, centrally. Valid certificates are those which are not revoked, expired, made inactive, and not issued by a CA which has had its certificate revoked or invalidated. As part of a well-architected application, you must have a process for identifying any invalid certificates and have an automated response in place to take some action to address the finding.

In addition to the ability of capturing the events where an invalid certificate is presented, your devices should also have a secondary means of establishing secure communications to your IoT system if mutually-authenticated TLS communications is not possible. This may involve manual, local interaction with the device, either by a consumer or service technician.

A well-architected IoT solution establishes a certificate revocation list (CRL) that tracks all revoked device certificates or certificate authorities (CAs). Use your own trusted CA for on-boarding devices and synchronize the CRL in the device and in your IoT application on a regular basis. Your IoT application must reject connections from identities (certificates or tokens) that are no longer valid.

With AWS, you do not need to manage your entire PKI on-premises. Use AWS Certificate Manager (ACM) or AWS Private Certificate Authority (PCA) to host your CA in the cloud. Or, you can work with an APN Partner to add preconfigured secure elements to your IoT device hardware specification. ACM has the capability to export a certificate revocation list (CRL) to a file in an S3 bucket. The CRL can be used to programmatically revoke certificates configured in AWS IoT Core.

Another state for certificates is to be near their expiry date but still valid. The device certificate must be valid for at least the service lifetime of the device. If a device certificate is nearing its expiration date and the device is to remain in operation, then your IoT application must take some action to update the device certificate with a certificate that has a later expiration date. Use the

AWS IoT Jobs or OTA to perform the necessary operations to carry out this refresh. Be sure to log information about the certificate refresh operations performed for auditing purposes.

Enable AWS IoT Device Defender audits related to device and CA certificate expiry. Device Defender produces an audit log of certificates that are set to expire within 30 days. Use this list to programmatically update devices before certificates are no longer valid. You may also choose to build your own expiry store to manage certificate expiry dates and programmatically query, identify, and trigger an IoT Jobs and OTA for device certificate replacement or renewal.

IOTSEC05-BP01 Perform certificate lifecycle management

Certificate lifecycle includes different phases such as creation, activation, refresh or rotation, revocation, deactivation, or expiry. An automated workflow can be put in place to identify certificates that need attention, along with remediation actions.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC05-BP01-01 *Document your plan for managing certificates.*

As explained earlier, X.509 certificates help to authenticate devices and AWS IoT Core to one another and to set up encrypted communications between the edge and cloud. Planning the lifecycle management of device certificates is essential. Enable auditing and monitoring for compromise or expiration of your device certificates. Determine how frequently you need to refresh/rotate device certificates, audit cloud or device-related configurations and permissions to make sure that security measures are in place. For example, use AWS IoT Device Defender to monitor the health of the device certificates and different configurations across your fleet. AWS IoT Device Defender can work in conjunction with AWS IoT Jobs to help enable refresh/rotation of certificates which are nearing their expiration.

Prescriptive guidance IOTSEC05-BP01-02 *Use certificates signed by your trusted intermediate CA for on-boarding devices*

As a best practice, the all-root CA keys must be locked and protected to secure the chain of trust. Device certificates should be generated using an intermediate CA to sign the device certificates. Define a process to programmatically manage intermediate CA certificates as well. For example, enable AWS IoT Device Defender Audit to report on your intermediate CAs that are revoked but device certificates are still active or if the CA certificate quality is low. You can thereafter use a security automation workflow using mitigation actions in AWS IoT Device Defender to resolve the issues.

Prescriptive guidance IOTSEC05-BP01-03 *Secure provisioning claims (just-in-time provisioning or registration) private keys and disable the certificate in case of misuse and record the event for further investigation.*

- Monitor provisioning claims for private keys when using just in time provisioning or just-in-time registration.
- Be sure to monitor usage on the device as well as in AWS IoT Core.
- For example:
  - Use AWS IoT CloudWatch metrics and logs to monitor for indications of misuse. If you detect misuse, disable the provisioning claim certificate so it cannot be used for device provisioning.
  - Use AWS IoT Device Defender to identify security issues and deviations from best practices.

Resources

- Getting started with AWS IoT Device Defender
- Just-in-Time Registration of Device Certificates on AWS IoT

Detective controls

Due to the scale of data, metrics, and logs in IoT applications, aggregating and monitoring is an essential part of a well-architected IoT application. Proper access controls paired with detection mechanisms help prevent unauthorized access to devices and connected resources. In order to operate an entire IoT solution, you will need to manage detective controls not only for an individual device but also for the entire fleet of devices in your application. You will need to enable several levels of logging, monitoring, and alerting to detect issues at the device level as well as the fleet-wide level.

In a well-architected IoT application, each layer of the IoT application generates metrics and logs. At a minimum, your architecture should have metrics and logs related to the physical device, the connectivity behavior of your device, message input and output rates per device, provisioning activities, authorization attempts, and internal routing events of device data from one application to another. Also, actions performed by the IoT application itself as well as actions performed by users of the IoT application should be logged.

IOTSEC06: How do you analyze application and device logs and metrics to detect security issues?

Your device logs and metrics play a critical role in monitoring security behavior of your IoT application. The way you configure your operations, and how anomalies are surfaced in your system will determine how quickly you can react to a security issue. By configuring your IoT logs and metrics appropriately, you can proactively mitigate potential security issues in your IoT application.

In AWS IoT, you can implement detective controls using AWS IoT Device Defender, Amazon CloudWatch Logs, AWS IoT Greengrass logs and Amazon CloudWatch Metrics. AWS IoT Device Defender processes logs and metrics related to device behavior and connectivity behaviors of your devices. AWS IoT Device Defender also lets you continuously monitor security metrics from devices and AWS IoT Core for deviations from what you have defined as appropriate behavior for sets of devices or each device.

Augment Device Defender metrics with the Amazon CloudWatch Metrics, Amazon CloudWatch Logs generated by AWS IoT Core, AWS IoT Greengrass logs and Amazon GuardDuty. These service-level logs provide important insight into activity about not only activities related to AWS IoT services and AWS IoT Core protocol usage, but also provide insight into the downstream applications running in AWS that are critical components of your end-to-end IoT application. All Amazon CloudWatch Logs should be analyzed centrally to correlate log information across all sources. AWS CloudTrail logs should be used to understand which AWS APIs have been used by which IAM principals as part of the IoT application processing.

Implement logging in any automation created as a part of the IoT application. Most IoT applications include some type of automated processing using, for example, AWS Lambda functions or AWS Step Functions. Add appropriate logging to these function implementations as well.

IOTSEC06-BP01 Collect and analyze logs and metrics to capture authorization errors and failures to enable appropriate response

Device logs and metrics can provide your organization with the insight to be operationally efficient with your IoT workloads by identifying security events, anomalies, and issues from device data.

Record error-level messages from AWS IoT Core to provide operational visibility to potential security issues.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC06-BP01-01 *Enable metrics and create alarms that track authorization and error metrics.*

Observe the trends for these AWS IoT metrics:

- `Connect.AuthError`
- `PublishIn.AuthError`
- `PublishOut.AuthError`
- `Subscribe.AuthError`

Configure CloudWatch alarms for each of the preceding metrics to alarm based on levels higher than normal for your workload.

IOTSEC06-BP02 Send alerts when security events, misconfiguration, and behavior violations are detected

Audit the configuration of your devices and detect and alert when a device behavior or IoT application processing differs from the expected behavior. Audit logs provide visibility into operational data that can indicate potential security issues active in the device fleet.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC06-BP02-01 *Enable metrics to detect security events from the data plane.*

Create IoT Device Defender security profiles to generate events which could indicate security risks. AWS IoT Device Defender Ccoud-side metrics report on device behavior observed by AWS IoT Core. You can detect events based on configured rules. For example, create a security profile in AWS IoT Device Defender, that detects unusual device behavior that may be indicative of a unauthorized access by continuously monitoring activity between the device and AWS IoT Core. You can specify normal device behavior for a group of devices by setting up behaviors (rules) for these metrics. AWS IoT Device Defender monitors and evaluates each data point reported for these metrics against user-defined behaviors (rules) and alerts you if behavior outside the defined rules settings is detected.

Prescriptive guidance IOTSEC06-BP02-02 *Enable auditing to check misconfigurations.*

Audit checks are necessary to determine that devices stay configured according to best practices throughout their lifecycle. For instance, it is necessary to audit devices regularly on basic checks such as logging, use of shared certificates and unique device identifiers. AWS IoT Device Defender audit checks can help you to continuously audit security configurations for compliance with security best practices and your own organizational security policies. Some of the auditing capabilities that are supported natively are `LOGGING-DISABLED-CHECK`, `IOT-POLICY-OVERLY-PERMISSIVE-CHECK`, `DEVICE-CERTIFICATE-SHARED-CHECK`, and `CONFLICTING-CLIENT-IDS-CHECK`.

Prescriptive guidance IOTSEC06-BP02-03 *Facilitate alerting on a behavior violation.*

Enable alarms or notifications when the device behavior is anomalous based on configured IoT Device Defender rules. AWS IoT Device Defender Security Profiles can be set up to define limits for metric values so that alerts are signaled if device behavior is observed to be outside of these limits.

Prescriptive guidance IOTSEC06-BP02-04 *Capture device-side behavior metrics and alert on device behavior violations.*

AWS IoT Device Defender can be configured to monitor device-side metrics which are reported to AWS IoT Device Defender from messages sent to AWS IoT Core by the device. Additional configuration and processing may be needed in the device in order to generate and send these device-side metrics. When available, these metrics can be used to alert you when behavior within the device is determined to be outside of normal ranges. Use AWS IoT Device Defender rules to monitor activity within the device. Appropriate action can then be taken, such as moving the device to a maintenance state or performing a remote OTA update on the device.

IOTSEC06-BP03 Alert on non-compliant device configurations and remediate using automation

Implement continuous monitoring to track device configurations and metrics. Regular auditing helps maintain security baselines and identify necessary updates as technologies evolve and new threats emerge. For example, cryptographic algorithms once known to provide secure digital signatures for device certificates can be weakened by advances in the computing and cryptoanalysis techniques.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC06-BP03-01 *Verify regular auditing is enabled for identifying configuration issues.*

Audit checks are necessary to determine that devices stay configured according to best practices throughout their lifecycle. For instance, it is necessary to audit devices regularly on basic checks such as logging, use of shared certificates and unique device identifiers. AWS IoT Device Defender audit checks can help you to continuously audit security configurations for compliance with security best practices and your own organizational security policies. Some of the auditing capabilities that are supported natively are `LOGGING-DISABLED-CHECK`, `IOT-POLICY-OVERLY-PERMISSIVE-CHECK`, `DEVICE-CERTIFICATE-SHARED-CHECK`, and `CONFLICTING-CLIENT-IDS-CHECK`. A full list of audit features can be found in Audit checks.

Prescriptive guidance IOTSEC06-BP03-02 *Use automation to remediate issues.*

Investigate issues by providing contextual and historical information about the device such as device metadata, device statistics, and historical alerts for the device. For example, you can use AWS IoT Device Defender built-in mitigation actions to perform mitigation steps on Audit and Detect alarms. Mitigations can include actions such as adding things to a thing group, replacing default policy version, and updating a device certificate. Another possible action is to enable a mitigation to re-enable logging and publish the finding to Amazon SNS should the `LOGGING-DISABLED-CHECK` find that logging is not enabled. Defining the actions taken when an alert is signaled is done by creating Lambda functions which are invoked through Amazon SNS when the alert is sent.

Infrastructure protection

Design time is the ideal phase for considering security requirements for infrastructure protection across the entire lifecycle of your device and solution. By considering your devices as an extension of your infrastructure, you can take into account how the entire device lifecycle impacts your design for infrastructure protection. From a cost standpoint, changes made in the design phase are less expensive than changes made later. From an effectiveness standpoint, data loss mitigations implemented at design time are likely to be more comprehensive than mitigations retrofitted. Therefore, planning the device and solution security lifecycle at design time reduces business risk and provides an opportunity to perform upfront infrastructure security analysis before launch.

One way to approach the device security lifecycle is through supply chain analysis. The IoT supply chain includes the actors, processes and assets that participate in the realization (For example, development, design, maintenance, and patch management) of any IoT device.

For example, even a modestly sized IoT device manufacturer or solution integrator has a large number of suppliers that make up its supply chain, whether directly or indirectly. To maximize solution lifetime and reliability, make sure that you are receiving authentic components.

Software is also part of the supply chain. The production firmware image for a device includes drivers and libraries from many sources including silicon partners, open-source aggregation sites such as GitHub and SourceForge, previous first-party products, and new code developed by internal engineering.

To understand the downstream maintenance and support for first-party firmware and software, you must analyze each software provider in the supply chain to determine if it offers support and how it delivers patches. This analysis is especially important for connected devices. Software bugs are expected and considered a normal part of development. Software bugs represent a risk to your customers especially when a vulnerable device can be accessed remotely. Your IoT device manufacturer or solution engineering team must learn about and patch bugs in a timely manner to reduce these risks.

All of the infrastructure protection capabilities available for securing AWS services should be applied to the cloud-hosted components of the IoT application. There are integration points where AWS IoT Core interacts with other AWS services such as AWS Lambda functions which define specific processing for the IoT application. Using the AWS IoT rules engine implies the definition of rules that are analyzed and then trigger downstream actions to other AWS services based on the messages sent over the MQTT topic stream. Since AWS IoT communicates to your other AWS resources, configure the right service role permissions for your application. The same applies for connected devices with AWS IoT Greengrass for cloud services the device needs to talk to.

AWS offers flexible ways and design patterns to establish a secure connection to the AWS environment from the edge. When choosing a secure connection to the AWS environment, take into consideration the use case requirements such as latency and data locality to make sure that the chosen connection solution meets your performance and compliance requirements. Use AWS Systems Manager to carry out routine management tasks on edge computing resources.

IoT devices are often deployed behind restricted firewalls at remote sites. Use secure tunneling for AWS IoT Device Management to access IoT devices for troubleshooting, configuration updates, and other operational tasks. Consider using AWS IoT Greengrass for secure remote application management. Take advantage of on-premises managed infrastructure solutions such as AWS Outposts, AWS Storage Gateway, AWS Snow Family to simplify management and monitoring.

IOTSEC07: What infrastructure protection configuration has been defined for your AWS organization and accounts?

All infrastructure protection controls that are used to secure your AWS organization and AWS accounts are applicable to IoT applications running in those accounts. IoT applications typically use a wide range of serverless technologies as well as cloud-hosted databases. Securing the network connectivity and access to these resources directly impacts the security of your IoT applications.

In addition to those controls, additional consideration should be placed on securing the infrastructure in which or around where IoT devices are deployed and operated. This includes the management of the hardware, firmware, and software installed and running in the IoT devices themselves.

IOTSEC07-BP01 Configure cloud infrastructure to have secure communications

Limit the communications paths and protocols used by the solution to only those which are necessary for the applications. For example, consider using only MQTT publish or subscribe communications when communicating with IoT devices. In addition, if possible, IoT devices should only connect out-bound to trusted and verified and authenticated service endpoints and not set up processes which listen for connections.

When it is necessary for IoT devices to listen for connection requests, the sources of these connections should be strictly limited. Any connecting client which is connecting to the device must be authenticated before the device communicates with that client. This applies whether or not the device itself is acting as a proxy for clients which connect to it. Authentication processing at the device is a device-specific design decision and brings additional complications such as how to authenticate, using what identity provider, and so on. This further supports the recommendation for devices to avoid listening for connection requests.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC07-BP01-01 *Use only MQTT publish/subscribe in IoT devices when possible.*

Configure devices to use MQTT communications and use the IoT Device Client or other MQTT client software to enable this communication.

Prescriptive guidance IOTSEC07-BP01-02 *Design IoT devices and solutions so that devices only connect and do not listen for connections.*

Refrain from creating server-type applications running on IoT devices. If necessary for handling local administration or configuration types of activities, consider only enabling such activities based on being placed into a maintenance mode and then stopping these applications during normal operation.

Prescriptive guidance IOTSEC07-BP01-03 *When listening for connections in IoT devices, authenticate connecting clients.*

Require authentication by any connecting entity which connects to the device. Be sure that there are no default credentials (passwords, keys, or tokens) which could become compromised and then used to access other devices. Authentication processing at the device is a device-specific design decision and brings additional complications such as how to authenticate, using what identity provider, and so on. This further supports the recommendation for devices to avoid listening for connection requests. To enable local administration, initial installation or provisioning should not rely on default credentials.

For example, local initial installation or provisioning on first start or after factory reset may require a local administrator to create a set of credentials or authenticate with a separate identity provider.

Prescriptive guidance IOTSEC07-BP01-04 *For sizeable data transfers like large file transfer, use encrypted HTTPS or SFTP communications with an IoT device as the connecting client.*

Use TCP protocols which are set up for handling bulk or large data transfers for performing those tasks. Connect from the IoT device to the remote system in order to put to or pull from files from that remote system. Verify the contents of those files using digital signatures or file hashes retrieved through a separate channel.

IOTSEC07-BP02 Define networking configuration which restricts communications to only those ports and protocols which are required

Restrict the possible communications protocols, paths, and ports on which IoT devices can communicate. Also, configure network communications so that there are separate network zones, with only the allowed protocols, ports, and connection initiation paths defined between these zones.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC07-BP02-01 *Use a minimum number of protocols for device communication.*

If only MQTT communications is required, restrict communications to only the IP port used for those communications. Consider using a protocol-aware firewall to restrict traffic to only that type of protocol. If HTTPS communications is also necessary, enable only the two protocols/ports.

Prescriptive guidance IOTSEC07-BP02-02 *Configure network zones which have strict protection for inbound/outbound communications and connection initiation.*

Configure network zoning using virtual or physical network connections. Use virtual network connection configuration to restrict connection initiation direction., Allow only outbound connections from IoT devices and restrict inbound connections from outside the local network.

IOTSEC07-BP03 Log and monitor network configuration changes and network communication

The ability to monitor and log communications assists in verifying that only the expected communications is taking place in the infrastructure. Also, having network logs allows for forensic analysis and problem determination if and when a problem is suspected or identified.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTSEC07-BP03-01 *Set up network logging at network zone connection points*

Use network routers or switches and firewalls between network zones. Set up network logging on those devices and appliances.

Prescriptive guidance IOTSEC07-BP03-02 *Send logs to a centralized logging infrastructure to enable remote problem determination and forensic analysis.*

Centralize logs to offload the logs from the remote devices and appliances. This also allows for network communications analysis across the overall solution in addition to looking at individual network zone activity. Centralized logging solutions are available on AWS. For example, Amazon OpenSearch Service Centralized Logging with OpenSearch is a centralized log management solution. Also, Amazon Security Lake can be used to understand, review, and act on security-related events in your computing environment.

IOTSEC08: How is the infrastructure into which your IoT devices are deployed managed and maintained?

After initial installation and configuration of an IoT solution, including the IoT devices, the solution components, networking configuration, and IoT devices themselves will still require ongoing maintenance and management. IoT devices should have a defined method for managing and maintaining their firmware, software, and configuration. This also includes maintaining or updating the identity of the devices, Also, the devices must be able to authenticate on connection start up as well as verify the identity of the endpoints which they connect to and communicate with.

IOTSEC08-BP01 Define an automated and monitored mechanism for deploying, managing, and maintaining networks to which IoT devices are connected

Having an automated and monitored mechanism for deploying network configuration allows for making changes to this network configuration depending on conditions. For example, if there is an issue or event detected in some portion of the network, that network zone could be isolated/quarantined until the situation is resolved. Conversely, separate network zones could be protected from issues or events, at the expense of some degradation in connectivity for a limited time, if an issue or event on that network zone is detected.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC08-BP01-01 *Use virtual network configurations to enable remote management of network configurations.*

In IIOT environments, enable remote management of network configuration. By enabling remote management of network configuration, the network can be adjusted over time to meet the needs of the solution or situation. Be aware, however, that such capability also comes with an added risk in that the remote configuration method itself must be protected. Consider using Amazon VPC, AWS Virtual Private Network, AWS Direct Connect, and Amazon Outposts to configure network connectivity.

IOTSEC08-BP02 Define an automated and monitored mechanism for deploying, managing, and maintaining network configurations for IoT devices

Having an automated and monitored mechanism for deploying, managing, and maintaining network configuration in IoT devices allows for making changes to this network configuration depending on conditions. For example, if there is an issue or event in some portion of the network, the network configuration in the device could be adjusted until the situation is resolved.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC08-BP02-01 *Use IoT Jobs to schedule and run management and update activities in IoT devices.*

IoT Jobs allows actions to be carried out in IoT devices based on both a schedule and the set of devices to which the jobs apply.

Prescriptive guidance IOTSEC08-BP02-02 *Use IoT Secure Tunneling sparingly to remotely access a device to take some corrective action.*

IoT Secure Tunneling allows for direct interaction with the device. However, this should be used only as a last resort since it implies that a human would be remotely attaching to and interacting with the device. Using specific remote command and control mechanisms is preferred to relying on opening up a secure tunnel through which a human operate would remotely access a device to perform some action. Remote command and control allow for much better input/output parameter checking for the operations being requested.

IOTSEC09: What processes are used to manage and maintain the hardware or software deployed and configured in your IoT devices?

After initial installation and configuration of the hardware, firmware, and software into the IoT device, usually at device manufacturing time, there are often new vulnerabilities discovered in the hardware, firmware, or software which has been embedded into those devices. There should be some means of updating the firmware or software in the devices so that a vulnerability, if deemed to be serious enough, can be remediated or mitigated. There are several ways to go about managing and maintaining the firmware or software which range in their cost and convenience.

Using an automated, repeatable, and monitored mechanism which has minimal manual (human) intervention lowers the cost of each deployment and reduces the potential of human error.

IOTSEC09-BP01 Manage and maintain IoT Device software using an automated, monitored, and audited mechanism

Having an automated, monitored, and audited mechanism for deploying, managing, and maintaining device software in IoT devices allows for making changes to this device software over time. The device software installed in each device should be maintained using a software bill of materials (SBOM). New features and fixes or updates can be applied to the device which extend its useful lifetime, address security vulnerabilities, or enable the device to perform more actions. Use AWS IoT Device Management Software Package Catalog (SPC) to aid in maintaining device software inventories.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC09-BP01-01 *Use IoT AWS IoT Greengrass and AWS IoT Greengrass component deployments to update software in IoT edge devices.*

IoT AWS IoT Greengrass supports a runtime environment in which concurrent components can be started, stopped, and updated. Communications is supported between components. This allows for complex parallel processing of multiple tasks within the device. IoT AWS IoT Greengrass has extensive support for defining and managing components and component versions as well as managing the deployment of those components into fleets of IoT AWS IoT Greengrass devices.

Prescriptive guidance IOTSEC09-BP01-02 *Use IoT Jobs to schedule and run management and update activities in IoT devices.*

IoT Jobs allows actions to be carried out in IoT devices based on both a schedule and the set of devices to which the jobs apply.

Prescriptive guidance IOTSEC09-BP01-03 *Use IoT Secure Tunneling sparingly to remotely access a device to take some corrective action.*

IoT Secure Tunneling allows for direct interaction with the device. However, this should be used only as a last resort since it implies that a human would be remotely attaching to and interacting with the device. Using specific remote command and control mechanisms is preferred to relying on opening up a secure tunnel through which a human operate would remotely access a device to perform some action. Remote command and control allow for much better input/output parameter checking for the operations being requested.

IOTSEC09-BP02 Manage IoT device configuration using automated and controlled mechanisms

In addition to managing the networking configuration and firmware or software in the IoT devices, the configuration settings in the device must also be managed and updated. Like updating the firmware or software, this should be done using an automated, monitored, and audited mechanism.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTSEC09-BP02-01 *Use IoT AWS IoT Greengrass and AWS IoT Greengrass component deployments to update configuration in IoT devices.*

IoT AWS IoT Greengrass supports a runtime environment in which concurrent components can be started, stopped, and updated. Communications is supported between components. This allows for complex parallel processing of multiple tasks within the device. IoT AWS IoT Greengrass has extensive support for defining and managing components and component versions as well as managing the deployment of those components into fleets of IoT AWS IoT Greengrass devices. One aspect of component management is the configuration of the components themselves. This can be used to update configuration in the IoT devices.

Prescriptive guidance IOTSEC09-BP02-02 *Use IoT Jobs to schedule and run management and update activities in IoT devices.*

IoT Jobs allows actions to be carried out in IoT devices based on both a schedule and the set of devices to which the jobs apply.

Prescriptive guidance IOTSEC09-BP02-03 *Use IoT Secure Tunneling sparingly to remotely access a device to take some corrective action.*

IoT Secure Tunneling allows for direct interaction with the device. However, this should be used only as a last resort since it implies that a human would be remotely attaching to and interacting with the device. Using specific remote command and control mechanisms is preferred to relying on opening up a secure tunnel through which a human operate would remotely access a device to perform some action. Remote command and control allow for much better input or output parameter checking for the operations being requested.

Data protection

Before architecting an IoT application, data classification, governance, and controls must be designed and documented. These controls should reflect how the data can be persisted in the cloud, and how data should be encrypted. These controls apply to data whether on a device, in transit between the devices and the cloud, processed in cloud-hosted services, or stored in cloud-hosted persistent storage locations. Unlike traditional cloud applications, data sensitivity and governance extend to the IoT devices that are deployed in remote locations outside of your network boundary. Attention to data classification, governance, and controls is important because IoT devices may handle personally identifiable data which is then transmitted from devices. Compliance with regulatory obligations extends across the IoT solution, from initial data capture at the device through handling and usage within the IoT application, and onward to what follow-on usage is allowed.

During the design process, determine how hardware, firmware, and data are handled at device end-of-life. Store long-term historical data in the cloud. Store a portion of current sensor readings locally on a device, namely only the data required to perform local operations. By only storing the minimum data required on the device, the risk of unintended access is limited. Actively delete or destroy data stored in the device so that only the minimum amount of sensor readings necessary is stored on the device.

In addition to reducing data storage locally, there are other mitigations that must be implemented at the end of life of a device such as:

- The device should offer a reset option which can reset the hardware and firmware to a default factory version.
- Your IoT application can run periodic scans for the last logon time of every device. Devices that have been offline for too long a period of time, or are associated with inactive customer accounts, can be isolated or quarantined from the rest of the IoT solution.
- Encrypt sensitive data that must be persisted on the device using a key that is unique to that particular device. Store the key in the device's secure element, or in a format which is encrypted based on key material in the devices' secure element.

In IIoT environments, to allow one-way data flow, access controls can be applied at the connectivity layer using security appliances such as firewalls and data diodes.

IOTSEC10: How do you make sure that device data is protected at rest and in transit?

All traffic to and from AWS IoT must be encrypted using Transport Layer Security (TLS). In AWS IoT, security mechanisms protect data as it moves between AWS IoT and other devices or AWS services. In addition to AWS IoT, you must implement device-level security to protect not only the device's private key but also the data collected and processed on the device.

For embedded development, AWS has several services that abstract components of the application layer while incorporating AWS security best practices by default on the edge. For microcontrollers, AWS recommends using FreeRTOS. FreeRTOS extends the FreeRTOS kernel with libraries for Bluetooth LE, TCP/IP, and other protocols. In addition, FreeRTOS contains a set of security APIs that allow you to create embedded applications that securely communicate with AWS IoT.

For Linux-based devices, AWS IoT Greengrass can be used to accelerate the development and operations of connected device software to extend cloud functionality to the edge of your network. AWS IoT Greengrass implements several security features, including mutual X.509 certificate-based authentication with connected devices, AWS IAM policies and roles to manage communication permissions between AWS IoT Greengrass and cloud applications, and subscriptions, which are used to determine how and if data can be routed between connected devices and AWS IoT Greengrass core.

Protect your data at rest by defining your requirements and implementing controls, including encryption, to reduce the risk of unauthorized access or loss. Protect your data in transit by defining your requirements and implementing controls, including encryption. Defining and implementing these controls reduces the risk of unauthorized data access or exposure. By providing the appropriate level of protection for your data in transit, you protect the confidentiality and integrity of your IoT data.

IOTSEC10-BP01 Use encryption to protect IoT data in transit and at rest

For data at rest, the Storage Networking Industry Association (SNIA) defines storage security as technical controls, which may include integrity, confidentiality and availability controls that protect storage resources and data from unauthorized users and uses. Thus, it is required to protect the confidentiality of sensitive data, such as the device identity, secrets, or user data by encrypting it at rest. For data in transit, use a secure transport mechanism such as TLS to protect

the confidentiality and integrity of data transmitted to and from your devices. Both MQTT and HTTP communications can be protected using TLS-protected forms of those protocols.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC10-BP01-01 *Require the use of device SDKs or client libraries for the device to communicate to cloud.*

Configure the IoT devices to communicate only over TLS to cloud endpoints. For example, use AWS IoT Greengrass or Amazon FreeRTOS SDKs to secure connectivity from your devices to AWS IoT Core over TLS 1.2. The AWS IoT Device SDK also enables the use of TLS-protected secure communications over TLS 1.2.

Prescriptive guidance IOTSEC10-BP01-02 *Encrypt data and secrets at rest on IoT devices.*

As explained earlier in section IOTSEC02-BP03-03, take advantage of encryption utilities provided by the host operating system to encrypt the data stored at rest in the local filesystem. In addition, take advantage of Secure Elements (SEs) and TPMs. Trusted execution environments (TEEs) can add storage protections as well.

IOTSEC10-BP02 Use data classification strategies to categorize data access based on levels of sensitivity

Data classification and governance is the customer's responsibility.

1. Identify and classify data based on sensitivity collected throughout your IoT workload and learn their corresponding business use-case.
2. Identify and act on opportunities to stop collecting unused data, or adjusting data granularity and retention time.
3. Consider a defense in depth approach and reduce human access to device data.

For more information, see Manage data streams on the AWS IoT Greengrass core.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTSEC10-BP02-01 *Implement data classification strategies for all data stored on devices or in the cloud, as well as all data sent over the network. Process data based on the level of sensitivity (for example, highly classified, or personally identifiable data).*

Before architecting an IoT application, data classification, governance, and controls must be designed and documented to reflect how the data can be persisted on the edge or in the cloud, and how data should be encrypted throughout its lifecycle. For example, by using AWS IoT Greengrass stream manager, you can define policies for storage type, size, and data retention on a per-stream basis. For highly classified data, you can define a separate data stream.

IOTSEC10-BP03 Protect your IoT data in compliance with regulatory requirements

Data governance is the rules, processes, and behavior that affect the way in which data is used, particularly as it regards openness, participation, accountability, effectiveness, and coherence. Data governance practices for IoT is important as it enables protecting classified data and complying with regulatory obligations. It helps to determine what data needs protection, or which data needs access control. For more information, see AWS Cloud Enterprise Strategy Blog: Using a Cloud Center of Excellence (CCOE) to Transform the Entire Enterprise.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC10-BP03-01 *Define specific roles for personnel responsible for implementing IoT data governance.*

For example, there might be a need for new roles to monitor security, from both the functional and policy perspectives, to control data when it moves from IoT environments to the cloud.

Prescriptive guidance IOTSEC10-BP03-02 *Define data governance policies to monitor compliance with approved standards.*

For example, you might define a policy that requires security credentials to never be hardcoded, even on edge devices. Thus, use only services like AWS Secrets Manager to retrieve secrets in an encrypted manner.

Prescriptive guidance IOTSEC10-BP03-03 *Define clear responsibilities to drive the IoT data governance process.*

Multiple administrative roles can exist for a single system. For instance, you may define roles for users who can replace defective devices, and separate roles for users who can apply security patches and upgrade device firmware. Note that roles and responsibilities might change over the lifecycle of your IoT systems.

Incident response

Being prepared for incident response in IoT requires planning on how you will deal with two types of incidents in your IoT workload. The first type of incident is an attempt by a threat actor to access an individual IoT device to disrupt the performance or change the device's behavior. The second incident is a broad event, such as network outages or DDoS attacks. In both scenarios, the architecture of your IoT application and infrastructure plays a large role in determining how quickly you will be able to detect and diagnose incidents, correlate the data across the incident, and then subsequently apply runbooks to respond and recover the affected devices in an automated, reliable fashion.

For IoT applications, follow the following best practices for incident responses:

- Organize sets of IoT devices into different groups based on device attributes such as location and hardware version.
- Enable searching of IoT devices by dynamic attributes, such as connectivity status, firmware version, application status, and device health.
- Stage OTA updates for IoT devices and deploy to devices in waves over a period of time. Deployment rollouts should be monitored and aborted if devices fail to maintain the appropriate KPIs.
- Test and verify that the update process is resilient to errors, and devices can recover and roll back from a failed software update.
- Keep detailed logs, metrics, and device telemetry for IoT devices that contain contextual information about how a device is currently performing and has performed over a period of time.
- Montor the overall health of your fleet using fleet-wide metrics. Alert when operational KPIs are not met for a period of time.
- Quarantine IoT devices that deviate from expected behavior. Inspect and analyze the device for potential compromise of the device by hardware tampering, firmware modification or application code modification.
- Test incident response procedures on a periodic basis.

Implement a strategy in which your Information Security team can quickly identify the devices that need remediation. Make sure that the Information Security team has runbooks that consider firmware versioning and patching for device updates. Create automated processes that proactively apply security patches to vulnerable devices as they come online.

Implement a monitoring solution in the operations technology (OT), IoT and IIoT environments to create an industrial network traffic baseline and monitor for anomalies and any deviation from the baseline. Collect security logs and analyze them in real-time using dedicated tools, for example, security information and event management (SIEM) class solutions such as within a security operation center (SOC). AWS works with a number of OT Intrusion Detection System (IDS) and SIEM partners that can be found on AWS Marketplace.

At a minimum, your security team should be able to detect an incident on a specific device based on the device logs and current device behavior. After an incident is identified, the next phase is to quarantine the device. To implement this with AWS IoT services, you can use AWS IoT Thing Groups with more restrictive IoT policies along with enabling custom group logging for those devices. This allows you to only enable features that relate to troubleshooting, as well as gather more data to understand root cause and remediation. Lastly, after an incident has been resolved, you must be able to deploy a firmware and software update to the device to return it to a known good state.

IOTSEC11: How do you plan the security lifecycle of your IoT devices?

The security lifecycle of your IoT devices includes everything, from how you choose your suppliers, contract manufacturers, and other outsourced relationships to how you manage security in your third-party firmware and manage security events over time. With visibility into the full spectrum of actors and activities in your hardware and software supply chain, you can be better prepared to respond to compliance questions, detect and mitigate events, and avoid common security risks related to third-party components.

IOTSEC11-BP01 Build incident response mechanisms to address security events at scale

There are several formalized incident management methodologies in common use. The processes involved in monitoring and managing incident response can be extended to IoT devices. For instance, AWS IoT Device Management capabilities provide fleet analysis and activity tracking to identify potential issues, in addition to mechanisms to enable an effective response.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC11-BP01-01 *Make sure that IoT devices are searchable by using a device management solution.*

Devices should be grouped by dynamic attributes, such as connectivity status, firmware version, application status, and device health.

Prescriptive guidance IOTSEC11-BP01-02 *Quarantine any device that deviates from expected behavior.*

Inspect the device for potential issues in the configurations, firmware or applications using device logs or metrics. If a risk or anomaly is detected, the device can be diagnosed remotely provided that capability exists. For example, Configure AWS IoT Secure Tunneling to remotely diagnose a fleet of devices.

If remote diagnosis is not sufficient or available, the other option is to push a security patch, application or firmware upgrade while the device is quarantined. When sending code to devices, the best practice is to sign the firmware or software and to verify the signature at the device prior to applying the update or code. This allows devices to detect if the code has been modified in transit. For example, With Code Signing for AWS IoT, you can sign code that you create for IoT devices supported by Amazon FreeRTOS and AWS IoT device management. In addition, the signed code can be valid for a limited amount of time to avoid further manipulation.

Prescriptive guidance IOTSEC11-BP01-03 *Over the air (OTA) update should be configured and staged for deployment activation during regular maintenance.*

Whether it's a security patch or a firmware update, an update to a config file on a device, or a factory reset, you need to know which devices in your fleet have received and processed any of your updates, either successfully or unsuccessfully. In addition, a staged rollout is recommended to reduce the scope of a bad update. Rollouts should be able to be aborted with devices returning to a failsafe condition on a failed update. For example, you can use AWS IoT Jobs to roll out OTA updates of security patches and device configurations in a staged manner with required rollout and abort configuration settings.

IOTSEC11-BP02 Require timely vulnerability notifications and software updates from your providers

Components in a device bill of materials (BOM), such as secure elements (SEs) or a trusted platform module (TPM) for key or certificate storage, can make use of updatable software components. Some of this software might be contained in the Board Support Package (BSP) assembled for your device. You can help to mitigate device-side security issues quickly by knowing where the security-sensitive software components are within your device software stack, and by understanding what to expect from component suppliers with regard to security notifications and updates.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC11-BP02-01 *Make sure that your IoT device manufacturer provides security-related notifications to you, and provides software updates in a timely manner to reduce the associated risks of operating hardware or software with known security vulnerabilities.*

Ask your suppliers about their product conformance to the Common Criteria for Information Technology Security Evaluation. In addition, use AWS Partner Device Catalog where you can find devices and hardware to help you explore, build, and go to market with your IoT solutions.

Application security

IoT applications encompass application code running in multiple locations, from on device application code to cloud-hosted processing to work with information from devices, to browser or mobile-device application code used to deliver a user experience for the IoT application. Each of these pieces of the application must be managed, maintained, updated, and improved over time.

In this IoT lens, application security refers specifically to the parts of the application running in devices and gateway devices as well as application which could affect how the devices and gateways are set up, managed, maintained, or configured.

The activities required to manage application security for IoT applications relies on many of the same activities and technologies used for general application security. Usage of code management tools for managing the source code for the application, usage of infrastructure as code (IaC) for defining the infrastructure and runtime environment, and usage of Continuous Integration/Continuous Deployment (CI/CD) tools to automate builds, scans, tests, and deployment of changes are also recommended when working with IoT application code running in IoT devices and gateways.

Static code analysis tools should be used to scan IoT device application code and to build software bill of materials (SBOMs) for the code running in IoT devices. Code should be scanned for vulnerabilities at build time as well as when the application code is held in various distribution locations such as libraries and container images.

IOTSEC12: How do you develop, maintain, manage, and deploy application code to your IoT devices and gateways?

The security lifecycle of your IoT devices depends upon the application code which is deployed into those devices and gateways. Creating, managing, and deploying those updates to your devices requires a trackable, audited, and repeatable set of operations which are applied to your devices and gateways.

IOTSEC12-BP01 Manage IoT device and gateway source code using source code management tools

All source code which implements applications in devices and gateways should be maintained in a version management system. Version management enables an orderly progression of changes to source code along with the ability to understand what updates to code were made when, by whom, and for what reasons. Version management also enables parallel feature development as well as a path for backing out or reverting changes that are found to be incorrect or unwanted.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC12-BP01-01 *Use a code management tool for managing source code.*

Many code management tools are available for use. Choose a tool which allows for your team to collaborate on application source code development, build, deployment, and testing.

Prescriptive guidance IOTSEC12-BP01-02 *Use a problem management tool for prioritizing updates and changes made to source code.*

Many problems and ticket management tools are available for use. Choose a tool which allows for your team to collaborate in problems, bugs or tickets as well as features to be developed. The problem management tool should enable linking between application source code updates and the problems or features which document the reasons for making the source code updates.

IOTSEC12-BP02 Use static code analysis tools and code scanning to check IoT application code

Source code scanning enables teams to identify potential issues in application code long before any build, deployment, and testing is performed. This can greatly reduce the time and effort needed to identify problems. Fixing issues earlier in the development or deployment cycle is generally much more efficient and cost-effective than fixing problems which are found in already-deployed software.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTSEC12-BP02-01 *Configure code scanning to run in multiple points during the development workflow.*

Source code scanning should be performed at multiple points in the development workflow. Code scans using workstation-based or integrated development environment (IDE) based plug-ins help developers identify and fix findings as they are writing the code. Code scans run before commit to version management provide another opportunity to identify issues early. Code scans run on workstations are typically optional or under the control of a developer for their configuration. Code scans run on code as it is pushed to central repositories and also when code is being built or prepared for deployment is a means of scanning code which has been merged from several developers' separate updates. Periodic code scanning as background processing jobs allows for scanning source code even when updates to that code have not been submitted. Periodic scanning helps find latent issues which might have been discovered since the code was developed or find issues which have been recently added as items to be checked or updated in source code.

Prescriptive guidance IOTSEC12-BP02-02 *Feed the results of code scanning tools to problem management systems.*

Provide feedback to the owners of the source code in the most appropriate format for them to be able to take action. If running in IDEs, provide feedback within the IDE. If code scanning is run in build automation, provide feedback in build logs as well as by using notification mechanisms to the developers responsible for that code. This may include creating problem reports or tickets in problem management systems and assigning them to the appropriate owner to resolve.

Prescriptive guidance IOTSEC12-BP02-03 *Establish a process for fixing issues identified from code scans.*

Once problems are identified, use the preferred problem management and backlog management tools for the team to prioritize and work on fixes that are indicated by the issues found. Backlog refinement should take care to not always de-prioritize fixing security-related code scan findings. Use an aging factor for security-related findings which boosts the priority of fixing the finding over time.

IOTSEC12-BP03 Deploy IoT applications using IaC, CI/CD pipelines, and build and deploy automation

Application security problems often arise from faulty or inconsistent deployment when humans are responsible for some set of the steps required to perform the build, test, and deployment of the application. By using build, test, deploy, and staged-roll-out mechanisms, human involvement

can be used for verification and approval processing, with the specific steps required to deploy the application codified into automation scripts and processing. These processing steps can be logged and monitored.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC12-BP03-01 *Use build automation pipelines to build, test, and stage updates for deployment.*

Build automation tools allow human involvement to be focused on verification and approval processing, with the specific steps required to deploy the application codified into automation scripts and processing. Consider using services such as AWS CodePipeline and AWS CodeDeploy for automating build, test, and staging of updates for deployment.

Prescriptive guidance IOTSEC12-BP03-02 *Use deployment automation to deploy updated to IoT application code*

Deployment automation tools enable a staged roll-out of updates to large fleets of devices. By rolling out in stages, issues found in rolling out the updated application can be limited in scope. Deployment automation tools provide information on the progress of deployments for those responsible for the roll-outs to monitor and take appropriate action if necessary. Consider using services such as AWS IoT Greengrass deployments, IoT Fleet Management, and AWS IoT Core Jobs for automating deployments of updated IoT application code.

Vulnerability management

IoT applications, like all other hardware, firmware or software, is susceptible to issues (vulnerabilities) arising from updates or evolutions in technology. Planning for how to maintain high security posture as well as identifying and remediating issues or risks in IoT devices and gateways which are already deployed must be addressed when creating any IoT solution. Maintaining high security posture relies on having automated build, scan, and test pipelines for IoT device firmware and software, accurate device or gateway inventories, consistent and frequent vulnerability scanning, and a workflow for remediating risks which have been identified.

The easiest phase of the project to reduce program code security risks is during development and test of an application or an update to an application. This implies that every build of the firmware and application code should include scanning for vulnerabilities as well as scanning depended-upon libraries and packages for known risks. Library or package versions should be updated regularly to pick up any fixes for vulnerabilities identified.

Updated firmware and applications should be deployed in a coordinated and automated manner across IoT devices and gateways. A roll-back mechanism should be in place to stop a roll-out and roll-back or revert updates which have been deployed and have been found to contain a problem.

Post deployment, newly discovered risks could exist in already deployed IoT device firmware and application code. With an accurate device inventory and SBOM for each device, devices containing such risks can be identified and updated firmware or software for the devices can be rolled out to those devices to address the risk or issue.

IOTSEC13: How do you identify and remediate risks in IoT device firmware, IoT application code, and depended-upon packages or libraries?

Managing the lifecycle of IoT device firmware, application code, and depended-upon packages must include a set of procedures to follow to identify and remediate risks which are discovered in that code. Utilize the build, test, packaging, and deployment automation tools to help in rolling out updates once they are available.

> **Note**
>
> Issues and risks in firmware and software may be discovered even when there have not been any updates made to the firmware or application. New threats and risks are discovered over time so that what was considered to be acceptable in the past may no longer be acceptable.

IOTSEC13-BP01 Use code and package scanning tools during development to identify potential risks during development

Vulnerability scanning enables teams to identify potential issues in device firmware and application. Updates to remediate the risks may require code changes, package version changes, or operating system version updates. Known vulnerability databases are updated regularly. New risks may be discovered in existing firmware and software even when that software has not changed.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC13-BP01-01 *Scan code and packages during every build of the application.*

Perform vulnerability scanning on every build and package of the application. This will provide an early check in the deployment lifecycle so that these findings can be addressed before the application code is deployed into any active environments.

Prescriptive guidance IOTSEC13-BP01-02 *Update depended-upon package versions regularly to pick up fixes for known issues that have been identified.*

Most software now depends on many open-source or third-party packages. Issues, risks, and bugs may be discovered in these packages. Any updates to these packages must then be applied to the applications which are using those packages and the updated applications then deployed to devices where that code is used. Plan to regularly re-build, package, and deploy updates to applications even if no additional features of application source code updates have been added.

IOTSEC13-BP02 Deploy updates to IoT device firmware or software to address identified issues

Use deployment automation to deploy updates which contain fixes for issues which have been discovered and identified. The same automation which is used to deploy firmware or application updates should be used to deploy updates for remediating risks.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTSEC13-BP02-01 *Use an automated, controlled, and staged roll-out of updates to firmware or software.*

Deployment automation tools enable a staged roll-out of updates to large fleets of devices. By rolling out in stages, issues found in rolling out the updated application can be limited in scope. Deployment automation tools provide information on the progress of deployments for those responsible for the roll-outs to monitor and take appropriate action if necessary. Consider using services such as AWS IoT Greengrass deployments, IoT Fleet Management, and AWS IoT Core Jobs for automating deployments of updated IoT application code.

Prescriptive guidance IOTSEC13-BP02-02 *Implement a mechanism for canceling a roll-out and rolling back an update which has been found to contain issues.*

The deployment automation mechanism should include a method for canceling or rolling back a set of updates which have been scheduled to be deployed to a set of devices. Deployment automation tools provide information on the progress of deployments for those responsible for the roll-outs to monitor and take appropriate action if necessary. Consider using services such as

AWS IoT Greengrass deployments, IoT Fleet Management, and AWS IoT Core Jobs for automating deployments of updated IoT application code.

IOTSEC13-BP03 Identify IoT devices which require updates and schedule updates to those devices

An accurate device inventory is needed in order to determine which IoT devices may be affected by a issue or risk which has been identified. Once a fix is available for the identified risk, the identified devices can be targeted to be updated.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC13-BP03-01 *Use an accurate device inventory which includes firmware or software version information to help identify IoT devices which require updates.*

Maintain an accurate inventory of IoT devices and the SBOM for firmware and applications deployed to those devices. Use the inventory to understand which devices are affected by issues which are identified in the firmware of packages in the SBOM. This enables targeted updates to be deployed to those devices which are found to contain the issue.

Prescriptive guidance IOTSEC13-BP03-02 *Consider implementing on-device endpoint detection and response (EDR) technologies to identify risks and request updates to the device firmware or software.*

For device operating systems which enable endpoint detection and response (EDR) technologies, consider using endpoint-based detection in order to identify vulnerabilities in those devices. Be sure to stage the roll-out of updates to devices so that if issues are found with an update the roll-out can be cancelled or rolled back.

Security governance

Managing the security of IoT devices requires a team of people with clearly defined roles and responsibilities. In many organizations, there is already a security governance team. In these cases, security governance of IoT devices and IoT applications should work in coordination with the security governance team for the organization. There will be overlap between governance of cloud-hosted aspects of IoT applications and overall security governance for the organization so having clearly defined roles and responsibilities will enable quick and efficient identification of how must do what actions when an incident is suspected or detected.

Like security governance for the organization, usage of DevOps and DevSecOps mechanisms will spread some of the roles and responsibilities for managing security across both operations

and development teams. The more that these teams work together and use common tools and methods to manage, maintain, and operate the IoT applications, the easier it will be to answer questions, prepare for audits, and respond to incidents. By codifying security policy into testable controls and using policy as code techniques, many parts of security governance can be built into the develop, build, test, and deploy automation used to manage, maintain, and improve IoT applications.

There should be a risk assessment and risk management process for IoT devices or gateways and applications which builds upon existing risk assessment and risk management mechanisms in place for the organization. Special attention should be paid environmental and human safety concerns when assessing and managing risk for IoT applications.

IOTSEC 14: How do you govern the security of your IoT applications?

The governance of IoT applications should be based on and integrated with the governance of other applications built and used by the organization. By using and extending existing governance teams and processes to cover IoT applications, appropriate teams can be informed and aware of potential risks from using IoT services and applications and put in place appropriate controls and mitigations for those risks.

IOTSEC14-BP01 Establish a security governance team for your IoT applications or extend the security governance team for the organization

A security governance team will evaluate IoT applications against a risk management framework. By establishing the potential risks that each IoT application poses, teams can then identify mitigations for those risks and update or remediate IoT applications appropriately. Security governance applies to people, processes, and tools used by the organization to establish, evaluate, and update the security posture of applications.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTSEC14-BP01-01 *Coordinate activities between security governance teams.*

If there are multiple security governance teams throughout the organization, coordinate the activities across these teams so that decisions made by one team are consistent and carried out by other parts of the organization which might be affected by those decisions.

IOTSEC14-BP02 Define security policy so that it can be written into verifiable checks using policy as code techniques

Security policies for an organization generally begin from existing standards such as NIST 800-53, ISO/IEC 27001, ISA/IEC 62443, and CIS. Using the controls identified in those standards along with the specific architecture and implementation of applications, verifiable checks of the configuration of the application can be created which result in the controls being expressed in code. These codified checks can then be automated so that compliance can be evaluated on a repeated and ongoing basis. Reports provide feedback on the compliance status of applications and logs of the automated checks provide evidence of ongoing evaluation of the environment.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC14-BP02-01 *Codify security policy into verifiable checks.*

Use tools such as AWS Config and the rules development kit (RDK) to codify security policies into verifiable checks. Additional services including AWS Security Hub, AWS IoT Device Defender, and Amazon Security Lake help to log compliance checks and provide reports on the compliance status of applications.

Prescriptive guidance IOTSEC14-BP02-02 *Implement security policy checks as part of the develop, build, test, and deploy workflow and automation.*

Security policy checks can also be implemented through source code scanning as well as policy checking during deployment processing. Use build automation and code scanning tools to check for security configuration and report findings into services such as Amazon Security Lake. Use AWS CloudFormation Hooks, a feature of AWS CloudFormation, to add compliance checks into the deployment processing of applications to check for and report issues with the configuration of infrastructure which supports applications.

IOTSEC14-BP03 Implement a risk assessment and risk management process

A risk management process includes procedures for identify, assess, and monitor risks as well as implementing mitigations for those risks. The NIST Risk Management Framework provides an example process which can be worked from if your organization does not already have a process to follow. If your organization has an existing risk management process, evaluate the process for any potential adjustments which are specific to IoT applications. Consider the environmental and human safety risks that may be applicable in IoT application environments.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTSEC14-BP03-01 *Integrate risk assessment and risk management with other problem management tools that are used by the development and operations teams.*

Any work items or tasks which are generated from risk assessment and risk management activities as well as findings from automated compliance scanning performed during development, build, and deployment of applications should be reflected back into the problem management tools used by the application and infrastructure development teams. Depending on the tools used to perform compliance checks, this integration can be built into the environment by using services such as Amazon EventBridge coupled with Amazon Simple Queue Service (SQS) and AWS Lambda.

Prescriptive guidance IOTSEC14-BP03-02 *Identify what environmental and human safety concerns are applicable to the IoT application.*

IoT applications often have direct interaction with humans and the environment. Pay particular attention to the risks related to environmental and human safety caused by the decisions, processing and actions taken by the IoT application.

Security assurance

Assuring security of IoT applications involves identifying the regulations which are applicable to the IoT devices or gateways and application code, establishing the set of controls which must be adhered to from those regulations, and assessing the devices or gateways and application against those controls. While proving that something is secure is, in general, difficult, it is possible to show or exhibit how a solution is architected, built, deployed, and operated. This information can show that relevant security configuration is set up as expected. Additional information can also show how the solution would detect risks and incidents so that appropriate response and recovery activities would take effect. Taken together, a solution can be evaluated for meeting required controls and being prepared to address situations where the controls did not prevent an incident from occurring.

Preparing for security assurance involves configuring for and collecting evidence that is used for logging, monitoring, auditing, and reporting. Performing security assurance activities includes evaluating the evidence provided and verifying those devices or gateways and solution components are configured according to required policies and controls.

Relevant regulations or standards for IoT applications include ISA/IEC 62443, Purdue Model, and the Industrial Internet of Things Security Framework (IISF). In addition, standards and frameworks

such as NIST Cybersecurity Framework, NIST 800-53, and NIST 800-82 and standards related to connected mobile security may also apply. If personal information is involved, privacy regulations such as General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) may also apply and be considered. Industry specific regulations or standards may also apply, such as North American Electric Reliability Corporation – Critical Infrastructure Protection (NERC-CIP).

In order to streamline and automate assurance activities as much as possible, employ compliance as code mechanisms which build automation into performing compliance checks and result in testing or verification logs which can be evaluated for both adherence to required controls and which violated controls must be remediated.

IOTSEC15: What regulations apply to your IoT applications and how do you show compliance with these regulations?

To carry out security assurance, understand which regulations apply to your IoT applications. Establish the set of regulations and plan to re-evaluate the set of regulations whenever the features or information processed by the application changes and at regular time intervals, for example, on a yearly review schedule.

IOTSEC15-BP01 Identify the set of relevant regulations for your IoT applications

Establish both the security and privacy related regulations which apply to IoT applications. If the application processes any information related to humans, then pay particular attention to privacy regulations and the sets of people that will use the application.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC15-BP01-01 *Understand which standards and regulations apply to the IoT application.*

The following standards and regulations are a starting point to use to understand which regulations apply to the application. There may be industry-specific regulations which also apply:

- ISA/IEC 62443
- ISO/IEC 33601

- ISO/SAE 21434
- Purdue Model
- Industrial Internet of Things Security Framework (IISF)
- NIST Cybersecurity Framework
- NIST 800-53
- General Data Protection Regulation (GDPR)
- California Consumer Privacy Act (CCPA)
- NERC-CIP
- Automotive Information Sharing and Analysis Center (Auto-ISAC)
- National Highway Traffic Safety Administration (NHTSA)

Prescriptive guidance IOTSEC15-BP01-02 *Determine which controls within the regulations apply to the IoT devices or gateways and application components.*

Within each regulation, determine which controls apply to the IoT application, the devices and gateways used in the application, and other services that are made use of by the application. Compliance with these controls can then be evaluated.

IOTSEC15-BP02 Set up logging and monitoring to support audit checks for compliance

Logging and monitoring provide a means of building a base of evidence which can be used to show both explicit compliance with regulations and, in the case of attempting to show that something did not happen, showing that there are checks and processing in place which would have identified an issue if that issue have been present.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTSEC15-BP02-01 *Centralize storage of log records used for audit reporting.*

Use a centralized storage mechanism for log records to reduce the potential for log records being lost or tampering with log records by authorized or unauthorized users. By off-loading log records to a separate storage location which has different access controls, the potential for log record tampering can be greatly reduced. Refer to recommendations for audit and logging in the AWS Security Reference Architecture(AWS SRA) for examples of centralizing log management and auditing activities.

Prescriptive guidance IOTSEC15-BP02-02 *Identify which log information can be used to show compliance with which controls.*

For each control which can be checked based on log information collected, identify the log records in the logged information which provide positive evidence of compliance with the control. For controls which require evidence that something did not occur, use log records which show that if a situation did occur, a log record would have signaled that situation. Use log testing events and mock situations to test and verify that this is the case.

IOTSEC15-BP03 Implement automated compliance checking using compliance as code

With an understanding of which logged information provides evidence of compliance with controls, automated compliance checks can be implemented which use the log data and evaluate the compliance check.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTSEC15-BP03-01 *Use automated compliance checking tools to evaluate compliance and produce summary reports and dashboards.*

Use tools such as AWS Config and rules development kit (RDK), Amazon CloudWatch, Amazon EventBridge Rules, and Serverless Computing - AWS Lambda. to automate compliance checks.

Prescriptive guidance IOTSEC15-BP03-02 *Integrate compliance checking with other problem management tools that are used by the development and operations teams.*

Any work items or tasks which are generated from automated compliance of IoT application logs should be reflected back into the problem management tools used by the application and infrastructure development teams. Depending on the tools used to perform compliance checks, this integration can be built into the environment by using services such as Amazon EventBridge coupled with Amazon Simple Queue Service and Serverless Computing - AWS Lambda.

Key AWS services

The essential AWS security services in IoT are the AWS IoT Core, AWS IoT Device Management, AWS IoT Device Defender, AWS Identity and Access Management (IAM), and Amazon Cognito. In combination, these services along with other AWS security services allow you to securely control access to IoT devices, AWS services, IoT applications and resources for your users. Additional AWS services such as AWS IoT Greengrass, Amazon S3, Amazon DynamoDB, and Amazon Relational

Database Service are often used in IoT applications. The following services and features support IoT security:

Design: The AWS Device Qualification Program provides IoT endpoint and edge hardware that has been pre-tested for interoperability with AWS IoT. Tests include mutual authentication and OTA support for remote patching.

Asset inventory: AWS IoT Device Management can be used as an inventory for IoT devices and AWS Systems Manager Inventory can be used to provide visibility into on premises computing resources and edge gateways.

AWS Identity and Access Management (IAM): Device credentials (X.509 certificates, IAM, Amazon Cognito identity pools and Amazon Cognito user pools, or custom authorization tokens) enable you to securely control device and external user access to AWS resources. AWS IoT policies add the ability to implement fine grained access to IoT devices. AWS Private Certificate Authority provides a cloud-based approach to creating and managing device certificates. Use AWS IoT thing groups to manage IoT permissions at the group level instead of individually. Use the AWS IoT credentials endpoint to obtain temporary IAM credentials in an IoT device in order to use AWS services from the IoT device.

Detective controls: AWS IoT Device Defender records device communication and cloud side metrics from AWS IoT Core. AWS IoT Device Defender can automate security responses by sending notifications through Amazon Simple Notification Service (SNS) to internal systems or administrators. AWS CloudTrail logs provide administrative actions of your IoT application. Amazon CloudWatch is a monitoring service with integration with AWS IoT Core and can trigger CloudWatch Events to automate security responses. CloudWatch captures detailed logs related to connectivity and security events between IoT edge components and cloud services.

Infrastructure protection: AWS IoT Core is a cloud service that lets connected devices easily and securely interact with cloud applications and other devices. The AWS IoT rules engine in AWS IoT Core uses IAM permissions to communicate with other downstream AWS services. AWS has created a wide selection of industry leading IoT silicon vendors, device manufacturers, and gateway partners who have integrated AWS IoT Greengrass into their software and hardware offerings. Customers have the option to store their device private key on a hardware secure element and store sensitive device information at the edge with AWS IoT Greengrass Secrets Manager and encrypt secrets using private keys for root of trust security.

Data protection: AWS IoT Core includes encryption capabilities for devices over TLS to protect your data in transit. AWS IoT Core integrates directly with services, such as Amazon S3 and Amazon

DynamoDB, which support encryption at rest. In addition, AWS Key Management Service (KMS) supports the ability for you to create and control keys used for encryption. On devices, you can use AWS edge offerings such as FreeRTOS, AWS IoT Greengrass, or the AWS IoT Embedded C SDK to support secure communication.

Patch management: Implement patch management to fix device vulnerabilities and define appropriate update mechanisms for software and firmware updates using AWS IoT Device Management Jobs service and AWS Systems Manager Patch Manager. Perform deployment of patches only after testing the patches in a test environment before implementing them in production and verify the integrity of the software before starting to run it making sure that it comes from a reliable source (signed by the vendor) and that it is obtained in a secure manner.

Incident response: AWS IoT Device Defender allows you to create security profiles that can be used to detect deviations from normal device behavior and trigger automated responses including Serverless Computing -AWS Lambda. AWS IoT Device Management should be used to group devices that need remediation and then using AWS IoT Jobs to deploy fixes to devices. AWS Security Hub can be used to aggregate security alerts from various AWS services and partner products to help you analyze your security trends and identify the highest priority security issues. AWS Security Hub provides you with a comprehensive view of your security state within AWS and your compliance with security standards and best practices and enables automated remediation. AWS Security Hub has out-of-the-box integrations with ticketing, chat, Security Information and Event Management (SIEM), Security Orchestration Automation and Response (SOAR), threat investigation, Governance Risk and Compliance (GRC), and incident management tools to provide users with a complete security operations workflow.

Business continuity and recovery: To backup IoT data at the edge and in the cloud, customers can use AWS IoT Greengrass stream manager to locally buffer data and send data to local storage destinations and other life cycle management features available in AWS IoT Greengrass to support your data resiliency and backup needs. AWS Backup can be used to centrally manage and automate backups across AWS services and on premise IoT systems.

Resources

Refer to the following resources to learn more about our best practices for security:

Documentation and blogs

- AWS IoT Device Management features

- Security in AWS IoT
- AWS IoT Device Defender
- AWS IoT SiteWise
- Authentication
- MQTT with TLS client authentication on port 443: Why it is useful and how it works
- MQTT 5 supported features
- Assessing OT and IIoT cybersecurity risk
- Detect anomalies on connected devices using AWS IoT Device Defender
- Implement security monitoring across OT, IIoT and cloud with AWS Security Hub
- Using AWS IoT Core with interface VPC endpoints
- AWS IoT secure tunneling
- AWS Systems Manager
- Network-to-Amazon VPC connectivity options
- Ten security golden rules for Industrial IoT solutions
- Ten security golden rules for connected mobility solutions
- AWS Security Incident Response Guide
- AWS Backup

Whitepapers

- Designing MQTT Topics for AWS IoT Core
- Security Best Practices in Manufacturing OT
- Securing Internet of Things (IoT) with AWS
- Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core

Reliability

The reliability pillar focuses on the ability to prevent and quickly recover from failures to meet business and customer demand. Key topics include foundational elements around setup, cross-project requirements, recovery planning, and change management.

Focus areas

Design principles

In addition to the overall Well-Architected Framework design principles, there are three design principles for reliability for IoT:

- **Simulate device behavior at production scale**: Create a production-scale test environment that closely mirrors your production deployment. Leverage a multi-step simulation plan that allows you to test your applications with a more significant load before your go-live date. During development, ramp up your simulation tests over a period of time starting with 10% of overall traffic for a single test and incrementing over time (that is, 25%, 50%, then 100% of day one device traffic). During simulation tests, monitor performance and review logs to make sure that the entire solution behaves as expected.
- **Buffer message delivery from the IoT rules engine with streams or queues**: Leverage-managed services enable high throughput telemetry. By injecting a queuing layer behind high throughput topics, IoT applications can manage failures, aggregate messaging, and scale other downstream services.
- **Design for resiliency**: It is essential to plan for resiliency on the device itself. Depending on your use case, resiliency may entail robust retry logic for intermittent connectivity, ability to roll back

firmware updates, ability to fail over to a different networking protocol or communicate locally for critical message delivery, running redundant sensors or edge gateways to be resilient to hardware failures, and the ability to perform a factory reset.

Definitions

Reliability encompasses four focus areas:

1. Foundations
2. Workload architecture
3. Change management
4. Failure management

To achieve reliability, a system must have a well-planned foundation and monitoring in place, with mechanisms for handling changes in demand, requirements, and protecting your system from high volumes of requests whether intentional or not. The system should be designed to detect the failure and automatically recover.

Foundations

IoT devices should continue to operate in some capacity in the face of network or cloud errors. Design device firmware to handle intermittent connectivity or loss in connectivity in a way that is sensitive to memory and power constraints. IoT cloud applications must also be designed to handle remote devices that frequently transition between being online and offline to maintain data coherency and scale horizontally over time. Monitor overall IoT utilization and create a mechanism to automatically increase capacity to make sure that your application can manage peak IoT traffic.

To help prevent devices from creating unnecessary peak traffic, device firmware must be implemented that helps prevent the entire fleet of devices from attempting the same operations at the same time.

For example, if an IoT application is composed of alarm systems and all the alarm systems send an activation event at 9am local time, the IoT application is inundated with an immediate spike from your entire fleet. Instead, you should incorporate a randomization factor into those scheduled activities, such as timed events and exponential backoff, to permit the IoT devices to more evenly distribute their peak traffic within a window of time.

IOTREL01: How do you make sure that your device consistently keeps its internal clock accurate?

A secure device should have a valid certificate. IoT devices use a server certificate to communicate to the cloud and the certificate presented uses time for certificate validity. Having reliable and accurate time is compulsory to be able to validate certificates. Because IoT data is not ordered, including an accurate timestamp with the data will enhance your analytic capabilities.

IOTREL01-BP01 Use NTP to maintain time synchronization on devices

IoT devices need to have a client to keep track of time—either using Real Time Clock (RTC) or Network Time Protocol (NTP) to set the RTC on boot. Failure to provide accurate time to an IoT device could help prevent it from being able to connect to the cloud.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL01-BP01-01 *Prefer NTP to RTC when NTP synchronization is available.*

Many computers have an RTC peripheral that helps in keeping time. Consider that RTC is prone to clock drift of about 1 second a day, which can result in the device going offline because of certificate invalidity.

Prescriptive guidance IOTREL01-BP01-02 *Use Network Time Protocol for connected applications.*

- Select a safe, reliable NTP pool to use, and a one that addresses your security design
- Many operating systems include an NTP client to sync with an NTP server
- If the IoT device is using GNU/Linux, it is likely to include the NTPD daemon
- You can import an NTP client to your system if using FreeRTOS
- The device's software needs to include an NTP client and should wait until it has synchronized with an NTP server before attempting a connection with AWS IoT Core
- The system should provide a way for a user to set the device's time so that subsequent connections can succeed
- Use NTP to synchronize RTC on the device to help prevent the device from deviating from UTC
- Consider the following The NTP Pool for vendors

- Chrony is a different implementation of NTP than what NTPD uses and it is able to synchronize the system clock faster and with better accuracy than NTPD. Chrony can be set up as a client and server.

IOTREL01-BP02 Provide devices access to NTP servers

An NTP server should be available for clients to use for local time. NTP servers are required by NTP clients to synchronize device time and function properly.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL01-BP02-01 *Provide access to NTP services.*

- ntp.org can be used to synchronize your computer clocks.

- Amazon Time Sync Service: a time synchronization service delivered over NTP, which uses a fleet of redundant satellite-connected and atomic clocks in each Region to deliver a highly accurate reference clock. This is natively accessible from Amazon EC2 instances and this can be pushed to edge devices.
- Chrony is a different implementation of NTP than what NTPD uses and it is able to synchronize the system clock faster and with better accuracy than NTPD. Chrony can be set up as a server and client.

IOTREL02: How do you manage service quotas and limits for peaks in your IoT workload?

AWS IoT provides a set of soft and hard limits for different dimensions of usage. AWS IoT outlines the data plane limits on the IoT limits, see AWS service quotas. Data plane operations (for example, MQTT Connect, MQTT Publish, and MQTT Subscribe) are the primary driver of your device connectivity. Therefore, it's important to review the IoT limits and make sure that your application adheres to any soft limits related to the data plane, while not exceeding any hard limits that are imposed by the data plane.

IOTREL02-BP01 Manage service quotas and constraints

For cloud-based workload architectures, there are service quotas (which are also referred to as service limits). These quotas exist to help prevent accidentally provisioning more resources than you need and to limit request rates on API operations so as to protect services from abuse.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTREL02-BP01-01 *Follow the Reliability Foundations Best Practices defined in the AWS Well-Architected Framework.*

The most important part of your IoT scaling approach is to make sure that you architect around any hard limits because exceeding limits that are not adjustable results in application errors, such as throttling and client errors. Hard limits are related to throughput on a single IoT connection. Consider restructuring your MQTT topics, or implementing cloud-side logic to aggregate or filter messages before delivering the messages to the interested devices.

Soft limits in AWS IoT traditionally correlate to account-level limits that are independent of a single device. For any account-level limits, you should calculate your IoT usage for a single device and then multiply that usage by the number of devices to determine the base IoT limits that your application will require for your initial product launch. AWS recommends that you have a ramp-up period where your limit increases align closely to your current production peak usage with an additional buffer. To make sure that the IoT application is not under provisioned:

- Consult published AWS IoT CloudWatch metrics for all limits: AWS IoT metrics and dimensions
- Monitor CloudWatch metrics in AWS IoT Core: Logging and Monitoring
- Alert on CloudWatch throttle metrics, which would signal if you need a limit increase.
- Set alarms for all thresholds in IoT, including MQTT connect, publish, subscribe, receive, and rule engine actions.
- Monitoring AWS IoT MQTT Traffic and Automating Quota and Throttling Notifications
- Monitoring your IoT Fleet using CloudWatch
- Make sure that you request a limit increase in a timely fashion, before reaching 100% capacity. See the AWS documentation on Requesting a quota increase: Requesting a quota increase

In addition to data plane limits, the AWS IoT service has a control plane for administrative APIs. The control plane manages the process of creating and storing IoT policies and principals, creating

the thing in the registry, and associating IoT principals including certificates and Amazon Cognito federated identities. Because bootstrapping and device registration is critical to the overall process, it's important to plan control plane operations and limits. Control plane API calls are based on throughput measured in requests per second. Control plane calls are normally in the order of magnitude of tens of requests per second. It is important for you to work backward from peak expected registration usage to determine if any limit increases for control plane operations are needed. Plan for sustained ramp-up periods for onboarding devices so that the IoT limit increases align with regular day-to-day data plane usage.

To protect against a burst in control plane requests, your architecture should limit the access to these APIs to only authorized users or internal applications. Implement back-off and retry logic, and queue inbound requests to control data rates to these APIs.

IOTREL03: How do you design workloads to operate efficiently within network bandwidth and storage constraints?

IOTREL03-BP01 Down sample data to reduce storage requirements and network utilization

Data should be down sampled where possible to reduce storage in the device and lower transmission costs and reduce network pressure.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL03-BP01-01 *Use device edge software capabilities for down sampling.*

- Use compression as a means of down sampling data
  - Data transmitted to the cloud can be in JSON format, or in other formats such as Protocol Buffers.
- Using AWS IoT Greengrass for device software to down sample data.
  - Applications built using Components can be used on AWS IoT Greengrass to down sample the data before sending it to the cloud.
  - ETL with AWS IoT Extract, Transform, Load with AWS IoT Greengrass Solution Accelerator helps to quickly set up an edge device with AWS IoT Greengrass to perform extract, transform, and load (ETL) functions on data gathered from local devices before being sent to AWS.

IOTREL04: How do you optimize and control message delivery frequency to IoT devices?

Devices can be restricted in message processing capacity and messages from the cloud might need to be throttled. The cloud-side message delivery rate might need to be architected based on the type of devices that are connected.

IOTREL04-BP01 Target messages to relevant devices

Devices receive information from shadow updates, or from messages published to topics they subscribe to. Some data are relevant only to specific devices. In those cases, design your workload to send messages to relevant devices only, and to remove any data that is not relevant to those devices.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL04-BP01-01 *Preprocess data to support the specific needs of the device.*

- Use AWS Lambda to pre-process the data and hone-in specifically to attributes and variables that are needed by the device to act upon

IOTREL04-BP02 Implement retry and back off logic to support throttling by device type

Retry and back off logic should be implemented in a controlled manner so that when you need to alter throttling settings per device type, you can easily do it. Using data storage of any chosen kind gives you flexibility on what data to publish down to the device.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL04-BP02-01 *Use storage mechanisms that enable retry mechanisms.*

- Using DynamoDB, you can hold data in key value format where device ID is the key. Retry logic can be applied to only certain device ID's.
- Using Amazon Relational Database Service, you have the flexibility to use a variety of database engines. The retry messages can have new real-time data augmented with historic data from previous device interactions stored in Amazon RDS.

- AWS IoT Events provides state machines with built-in timers to hold back data and retry based on timers.

Workload architecture

A reliable workload starts with upfront design decisions for both software and infrastructure. Your architecture choices will impact your workload behavior across all six Well-Architected pillars. For reliability, there are specific patterns you must follow:

The following sections explain best practices to use with these patterns for reliability.

IOTREL05: How do you manage data ingestion and processing throughput for IoT workloads to other applications?

Although IoT applications have communication that is only routed between other devices, there will be messages that are processed and stored in your application. In these cases, the rest of your IoT application must be prepared to respond to incoming data. All internal services that are dependent upon that data need a way to seamlessly scale the ingestion and processing of the data.

IOTREL05-BP01 Decouple IoT applications from the Connectivity Layer through an Ingestion Layer

In a well-architected IoT application, internal systems are decoupled from the connectivity layer of the IoT system through the ingestion layer. The ingestion layer is composed of queues and streams that enable durable short-term storage while allowing compute resources to process data independent of the rate of ingestion.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL05-BP01-01 *Decouple application consumers using streaming data services.*

To optimize throughput, use AWS IoT rules to route inbound device data to services such as Amazon Kinesis Data Streams, Amazon Data Firehose, Amazon Simple Queue Service, or Amazon Managed Streaming for Apache Kafka before performing any compute operations. Make sure that all the intermediate streaming points are provisioned to handle peak capacity. This approach creates the queueing layer necessary for upstream applications to process data resiliently.

Prescriptive guidance IOTREL05-BP01-02 *Make use of MQTT features to support reliable delivery of messages.*

AWS IoT Core supports MQTT persistent sessions, which store a client's subscriptions and messages that haven't been acknowledged by the client. Messages are stored according to account limits, and the Persistent session expiry period what can be adjusted between 1 hour and 7 days. This allows for clients to publish messages that will be persisted by the AWS IoT Core Broker for up to the account limits and expiry period, for later processing. Read more about persistent sessions in the AWS IoT Core developer guide.

IOTREL06: How do you facilitate reliable processing and delivery of IoT messages across your workload?

Data sent from devices should be processed and stored without excessive loss. Services that queue and deliver IoT data to compute and database services should be used to support the processing of data. IoT devices send lots of data in small sizes without order, and the cloud application should be able to handle this.

IOTREL06-BP01 Dynamically scale cloud resources based on the utilization

The elastic nature of the cloud can be used to increase and decrease resources on demand. Use the ability to increase and decrease cloud resources based on data, number of messages, and size of messages and number of devices.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL06-BP01-01 *Know the mechanisms that can be used to monitor cloud resource usage and methods to scale the resources.*

- Use Amazon CloudWatch Logs to trigger based on rate of data flow to auto-scale cloud resources as needed.
- Use AWS IoT Rules engine error actions to provision additional cloud resources and message retries as needed.
- Examine IoT logs for errors in communicating to resources and provision resources based on that data.

- Use AWS Lambda to automatically scale your application by running code in response to each event.
- Use automatic scaling where possible. Kinesis Data Streams and Amazon DynamoDB are two services that provide automatic scaling.

Prescriptive guidance IOTREL06-BP01-02 *Use MQTT 5 Shared Subscriptions to effectively load balance MQTT messages across several subscribers.*

Using *Shared Subscriptions* in MQTT is an effective way to *load balance* messages across multiple subscribers in a way that optimizes resource usage, improves scalability, and supports more efficient message delivery.

IOTREL07: How do you provision storage strategies for IoT data in the cloud?

IoT devices send a lot of small messages with no guarantee of delivery order. This data might not be immediately useful, but the data volume is typically low enough to economically store against a future need. It will be beneficial to store the data so that the data can processed in order. Stored data can be reprocessed as new requirements are developed.

IOTREL07-BP01 Store data before processing

Make sure that the data from the devices is stored before processing. As new requirements and capabilities are added, stored data can be analyzed to meet the new requirements.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL07-BP01-01 *Use IoT Core Rules Engine to send data to Firehose to batch and store data on Amazon S3.*

- IoT Rules Engine can send data to Firehose to batch and store data on Amazon S3. Intelligent tiering can be enabled in Amazon S3 to reduce storage costs.
- Understand the latency to access data and choose the Region to store the data in based on device location.
- If data will be processed in Amazon EC2 instances, consider using the highly available and low-latency Amazon Elastic Block Store (Amazon EBS).

- NoSQL data can be stored in Amazon DynamoDB, which is a key-value and document database that delivers single-digit millisecond performance at scale. IoT Core Rules engine can write all or part of an MQTT message to a DynamoDB table.

IOTREL07-BP02Implement storage redundancy and failover mechanisms for IoT data persistence

There should be recovery plans for failures in storing and accessing device data in the cloud. Understand the Recovery Point Objective (RPO) and Recovery Time Objective (RTO) needed by your application to access data to be used for analysis.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL07-BP02-01 *Know how to monitor and take action on cloud storage failures for IoT data.*

- AWS Health Dashboard provides notification and remediation guidance when AWS is experiencing events that might impact you. Storage and access of data can be modified based on the notification.
- Use Amazon CloudWatch Logs to trigger on events on writing and reading data and take appropriate error handling action.
  - Use AWS IoT rules engine error actions to provision data storage to other locations if primary storage is unavailable.

Change management

IOTREL08: How do you update device firmware on your IoT device?

It is important to implement the capability to revert to a previous version of your device firmware or your cloud application in the event of a failed rollout. If your application is well-architected, you will capture metrics from the device, as well as metrics generated by AWS IoT Core and AWS IoT Greengrass. You will also be alerted when your device canaries deviate from expected behavior after any cloud-side changes.

IOTREL08-BP01 Use a mechanism to deploy and monitor firmware updates

When performing over-the-air (OTA) updates to remote devices' firmware, make sure that the updates are controlled and reversible to avoid functional impact of the device to the user, or the device entering a non-recoverable state. Use tools that allow you to deploy and track management tasks in your device fleet.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL08-BP01-01 *Use a cloud-based update orchestrator to deploy your firmware.*

- You can use AWS IoT Jobs to send remote actions to one or many devices at once, control the deployment of jobs to your devices, and track the current and past status of job executions for each device.
- Using FreeRTOS OTA using AWS IoT Jobs: By using AWS IoT Jobs for FreeRTOS, you have reliability and security provided out of the box where OTA update job will send firmware to your end device over secure MQTT or HTTPS and system reserved topics are provided to keep track on the status of the job schedule.
- Using custom IoT jobs with AWS IoT connected devices: By using AWS IoT Jobs with one or more devices connected to AWS IoT gives you the ability to track the full roll out of the update.

Prescriptive guidance IOTREL08-BP01-02 *Version all of the device firmware artifacts.*

- Version all of the device firmware using Amazon S3.
- Version the manifest or execution steps for your device firmware.
- Implement a known-safe default firmware version for your devices to fall back to in the event of an error.
- Implement an update strategy using cryptographic code-signing, version checking, and multiple non-volatile storage partitions, to deploy software images and rollback.
- Version all IoT rules engine configurations in CloudFormation.
- Version all downstream AWS Cloud resources using AWS CloudFormation.
- Implement a rollback strategy for reverting cloud side changes using AWS CloudFormation and other infrastructure as code tools.

Treating your infrastructure as code on AWS allows you to automate monitoring and change management for your IoT application. Make sure that updates can be verified, installed, or rolled back when necessary.

Devices will need new features over time for better user experience and the firmware will need to be updated remotely. Devices should be designed to receive and update their firmware and the IoT application should be designed to send firmware updates and monitor the success of such an update send.

IOTREL08-BP02 Configure firmware rollback capabilities in devices

Augment hardware with software to hold two versions of firmware and the ability to switch between them. Devices can rapidly roll back to older firmware if the new firmware has issues.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL08-BP02-01 *Leverage an RTOS with functionality to roll back device firmware.*

By combining OTA agents provided by FreeRTOS or using AWS IoT Device SDK, you can create flexibility to hold two versions of firmware with the hardware that is capable of storing it.

IOTREL08-BP03 Implement support for incremental updates to target device groups

It is a good practice to test new firmware on a small group of devices. Using a smaller group of devices for firmware updates helps make sure that the firmware as well as the upgrade process is well tested before the entire fleet is updated.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL08-BP03-01 *Use a cloud orchestrator in conjunction with device settings augmentation. Cloud services can help you control and manage jobs in tandem with the devices running the jobs.*

- The AWS IoT Jobs API provides a granular level of control from the cloud to the device for carrying out firmware update incrementally and roll back as needed.
- A job document created as part of AWS IoT job details the remote operations the device needs to perform. This includes shutting down rollouts based on timeouts, number of updates per device among other things. Devices can use this information to reject or accept firmware updates.

IOTREL08-BP04 Implement dynamic configuration management for devices

Deploying software changes to devices constitutes a high-risk operation due to the recovery cost associated with remotely deployed devices. When possible, prefer mechanisms for making changes using command-and-control channels to reduce the risk that comes with software deployments and firmware upgrades. This approach enables you to push some changes to devices while minimizing the risk of entering fault states that require on-premises recovery actions. Configuration changes reduce the amount of bandwidth compared to firmware updates.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL08-BP04-01 *Use cloud tools to command and control devices. Changing configuration of devices is less error prone and easier to trace back than updating firmware.*

- Use Secure Tunneling or Systems Manager to facilitate patching of the operating system instead of pushing a new image to be loaded on the device.
- Use Device Shadows to command-and-control devices rather than sending commands directly to device.
- Use AWS IoT Device Management jobs to rotate expiring device certificates instead of pushing a new image with updated certificates.
- AWS IoT secure tunneling
- AWS IoT Device Shadow service

IOTREL09: How do you perform functional testing for your IoT solution?

Testing IoT applications and backend services is expensive and can be a challenge due to the large pool of physical, connected devices required. Simulation helps test device integration and IoT backend services, without the need for physical devices. You can also monitor devices from the simulator or observe how backend services are processing the data.

IOTREL09-BP01 Implement device simulation to synthesize the entire flow of IoT data

Simulation scenarios can be configured to generate high volumes of traffic, simulating a large number of IoT devices interacting with the infrastructure simultaneously. By analyzing metrics such as message throughput, latency, and error rates during load testing, users can identify potential bottlenecks and optimize their infrastructure for reliability and responsiveness.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL09-BP01-01 *To augment your production device deployments, implement IoT simulations on Amazon Elastic Compute Cloud (Amazon EC2) as device canaries across several AWS Regions.*

- These device canaries are responsible for mirroring several of your business use cases, such as simulating error conditions like long-running transactions, sending telemetry, and implementing control operations. The device simulation framework must output extensive metrics, including but not limited to successes, errors, latency, and device ordering and then transmit all the metrics to your operations system.
- You must implement a variety of device simulation canaries that continue to test common device interactions directly against your production system. Device canaries assist in narrowing down the potential areas to investigate when operational metrics are not met. Device canaries can be used to raise preemptive alarms when the canary metrics fall below your expected SLA.

Prescriptive guidance IOTREL09-BP01-02 *The IoT Device Simulator simulates diverse scenarios to validate the logic and functionality of their IoT applications.*

- Launch fleets of virtually connected devices from a user-defined template and then simulate them to publish data at regular intervals to AWS IoT
- Simulation scenarios can be utilized to generate synthetic data for training ML models used in IoT applications. By simulating different environmental conditions, device behaviors, and data patterns, users can generate diverse datasets to train and validate ML algorithms.

For more information see, IoT Device Simulator.

Failure management

IOTREL10: How do you implement your IoT workload to withstand component and system faults?

Understanding and predicting the fault scenarios in the system helps you to architect for failure conditions and use service features to handle them. Therefore, the handling of such predicted system faults and recovering from them should be architected into the system.

IOTREL10-BP01 Use cloud service capabilities to handle component failures

An IoT design consists of device software, connectivity and control services, and analytics services. Test the entire IoT landscape for resiliency, starting with device firmware, data flow, the cloud services used, and error handling. Vendors have services integrated with each other to provide a simplified integration and fault handling.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTREL10-BP01-01 *Understand and apply the standard libraries available to manage your device firmware or software.*

- Devices can be built on FreeRTOS which provides connectivity, messaging, power management and device management libraries that are tested for reliability and designed for ease of use.
- AWS provides IoT device SDKs and Mobile SDKs, comprised of open-source libraries, developer guides, sample apps, and porting guides to help you build IoT solutions with AWS IoT and your choice of hardware systems.

Prescriptive guidance IOTREL10-BP01-02 *Use log levels appropriate to the lifecycle stage of your workload.*

- AWS IoT logs can be set up per region and per account with the logging level set to DEBUG during product development phase to provide insights on data flow and resources used. This data can be used to improve the IoT system security and performance.
- AWS IoT Secure Tunneling can be used to test and debug devices that are behind a restrictive firewall in the field.

IOTREL11: How do you verify that your IoT device operates with intermittent connectivity to the cloud?

IoT solution reliability must also encompass the device itself. Devices may be deployed in remote locations and deal with intermittent connectivity, or loss in connectivity, due to a variety of external factors that are out of your IoT application's control.

For example, if an ISP is interrupted for several hours, how will the device behave and respond to these long periods of potential network outage? Implement a minimum set of embedded operations on the device to make it more resilient to the nuances of managing connectivity and communication to AWS IoT Core.

IOTREL11-BP01 Implement device logic to automatically reconnect to the cloud

Your IoT device will likely become disconnected due to networking issues, power loss, or other unforeseen situations. This might be true of a single device, or for your entire fleet of devices. Whether a single device or the entire fleet becomes disconnected, the following best practices will make sure that the entire fleet is able to automatically reconnect.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL11-BP01-01 *Use an exponential backoff with jitter and retry logic to connect remote devices to the cloud.*

Consider implementing a retry mechanism for IoT device software. The retry mechanism should have exponential backoff with a randomization factor built in to avoid retries from multiple devices occurring simultaneously. Implementing retry logic with exponential backoff with jitter allows the IoT devices to more evenly distribute their traffic and help prevent them from creating unnecessary peak traffic.

Prescriptive guidance IOTREL11-BP01-02 *Use device edge software and the SDK to use built in exponential backoff logic.*

- Exponential backoff logic is included in the AWS SDK, including the AWS IoT Device SDK, and edge software, such as AWS IoT Greengrass Core and FreeRTOS.
- AWS SDK handles the exponential backoff

- AWS IoT Device SDK C: MQTT uses IOT-MQTT-RETRY-MS-CEILING for setting maximum retry interval limit.

IOTREL11-BP02 Design devices to use multiple methods of communication

Devices hardware can be designed to make use of multiple networking interfaces. Consider a device that provides multiple network interface types when selecting device hardware according to the needs of your IoT application.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL11-BP02-01 *Establish alternate network channels to meet requirements.*

- Have a separate failover network channel to deliver critical messages to AWS IoT. Failover channels can include Wi-Fi, cellular networks, or a wireless personal network.
- For low latency workload, use AWS Wavelength for 5G devices and AWS Local Zones to keep your cloud services closer to the user.

IOTREL11-BP03 Automate alerting for devices that are unable to reconnect

In the event that devices are unable to reconnect, fleet operators are to be automatically notified to begin troubleshooting the device and to re-establish device connectivity.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL11-BP03-01 *Implement logic in the cloud to notify the device operator if a device has not connected for an extended period of time.*

- Lifecycle events can be enabled to monitor device lifecycle events, including connect and disconnect events.
- AWS IoT Fleet Indexing can be used to identify device connectivity status
- AWS IoT Events can be used to monitor devices remotely.
- Remote monitoring using AWS IoT Events: CloudWatch Metrics connector

IOTREL12: How do you verify that required data is transmitted to the cloud after a device has been disconnected?

Your IoT device must be able to operate without internet connectivity. To make sure that required data is not lost when devices become disconnected from the cloud, they should store important messages durably offline and, once reconnected, send those messages to AWS IoT Core. Connection to the cloud can be intermittent and devices should be designed to handle this. Choose devices with firmware designed for intermittent cloud connection and that have the ability to store data on the device if you cannot afford to lose the data.

IOTREL12-BP01 Provide adequate device storage for offline operations

Store important messages durably offline and, once reconnected, send those messages to the cloud. Device hardware should have capabilities to store data locally for a finite period to help prevent loss of information.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL12-BP01-01 *Use the device edge software capabilities for storing data locally.*

- Design your edge applications according to your device constraints to store and forward critical data when devices become disconnected from the cloud.
  - If your device has sufficient storage available, your application may implement a local cache of messages written to disk to make sure that data is not lost when the device is operating in a disconnected state.
  - To make sure that the disk is not accidentally filled with this persisted data, design your application to make use of only a set amount of total disk space, and consider implementing a FIFO overwrite strategy.
  - When the device comes back online, a background process should be implemented to transmit data that was stored locally to the cloud, emptying the local cache as messages are successfully published to the cloud.
- If using AWS IoT Greengrass for device software, AWS IoT Greengrass components can help collect, process, and export data streams, including when devices are offline.
  - Messages collected on the device are queued and processed in FIFO order.

- By default, AWS IoT Greengrass Core stores unprocessed messages destined for AWS Cloud targets in memory.
- Configure AWS IoT Greengrass to cache messages to the local file system so that they persist across core restarts.
- AWS IoT Greengrass stream manager makes it easier and more reliable to transfer high-volume IoT data to the AWS Cloud.
- Configure AWS IoT Greengrass core
- Manage data streams on AWS IoT Greengrass Core
- AWS IoT Greengrass Developer Guide
- Run Lambda functions on the AWS IoT Greengrass core
- The ETL with AWS IoT Greengrass solution accelerator (For more information, see Unlock the value of embedded security IP to build secure IoT products at scale)helps to quickly set up an edge device with AWS IoT Greengrass to perform extract, transform, and load (ETL) functions on data gathered from local devices before being sent to AWS.

Prescriptive guidance IOTREL12-BP01-02 *Consider using AWS IoT SiteWise for data coming from disparate industrial equipment.*

AWS IoT SiteWise Edge software collects local equipment data and sends it to AWS IoT SiteWise in the cloud. You can use SiteWise Edge gateways to collect data from multiple OPC Unified Architecture (UA) servers and publish it to AWS IoT SiteWise. The SiteWise Edge gateway runs on either AWS IoT Greengrass V2 or Siemens Industrial Edge can be used to cache data locally in the event of intermittent network connectivity. You can configure the maximum disk buffer size used for caching data. If the cache size exceeds the maximum disk buffer size, the connector discards the earliest data from the queue. For more information, see Use AWS IoT SiteWise Edge gateways.

IOTREL12-BP02 Synchronize device states upon connection to the cloud

IoT devices are not always connected to the cloud. Design a mechanism to synchronize device states every time the device has access to the cloud. Synchronizing the device state to the cloud allows the application to get and update device state easily, as the application doesn't have to wait for the device to come online.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL12-BP02-01 *Use a digital devices state representation to synchronize device state using the below capabilities.*

- AWS provides device shadow capabilities that can be used to synchronize device state when the device connects to the cloud. The AWS IoT Device Shadow service maintains a shadow for each device that you connect to AWS IoT and is supported by the AWS IoT Device SDK, AWS IoT Greengrass core, and FreeRTOS.
- Synchronizing device shadows - Device SDKs and the AWS IoT Core take care of synchronizing property values between the connected device and its device shadow in AWS IoT Core.
- AWS IoT Greengrass – AWS IoT Greengrass core software provides local shadow synchronization of devices and these shadows can be configured to sync with cloud.
- FreeRTOS - The FreeRTOS device shadow API operations define functions to create, update, and delete AWS IoT Device Shadow services.

Prescriptive guidance IOTREL12-BP02-02 *Use MQTT Persistent Sessions.*

MQTT's persistent session feature allows a client to retain its subscriptions, undelivered messages, and other session data across different connections. If a device (client) disconnects and later reconnects, it can pick up where it left off without having to re-subscribe or miss critical messages.

IOTREL13: How do you remotely adjust message frequency to your IoT devices?

Because IoT is an event-driven workload, your application code must be resilient to handling known and unknown errors that can occur as events are permeated through your application. A well-architected IoT application has the ability to log and retry errors in data processing. An IoT application will archive data in its raw format. By archiving data, valid and invalid, an architecture can more accurately restore data to a given point in time.

IOTREL13-BP01 Configure cloud services to reliably handle message processing

When devices send an unexpected influx of messages, or when your device fleet grows, it becomes necessary to add error handling to support the reliable delivery of messages in your IoT applications.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL13-BP01-01 *Configure error actions with IoT Rules Engine.*

With the IoT rules engine, an application can enable an IoT error action. If a problem occurs when invoking an action, the rules engine will invoke the error action. This allows you to capture, monitor, alert, and eventually retry messages that could not be delivered to their primary IoT action. We recommend that an IoT error action is configured with a different AWS service from the primary action. Use durable storage for error actions such as Amazon SQS or Amazon Kinesis.

Beginning with the rules engine, your application logic should initially process messages from a queue and validate that the schema of that message is correct. Your application logic should catch and log any known errors and optionally move those messages to their own dead-letter queue (DLQ) for further analysis. Have a catch-all IoT rule that uses Amazon Data Firehose to transfer raw and unformatted messages into long-term storage in Amazon S3, or Amazon Redshift for data warehousing.

IOTREL13-BP02 Send logs directly to the cloud

It is common for device developers to log application errors at the edge, but that increases the complexity for reliably troubleshooting device issues, especially as device fleets increase in size. Storing log files on the device itself then requires a specialized process to request a device to transmit logs, which it may not be able to accomplish during failure states, or to open remote access to the device to access those logs. Instead, transmit logs as events to the cloud and automate alerts based on those log events to improve reliability of your IoT applications.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL13-BP02-01 *Use MQTT to send log messages to the cloud.*

Regardless of the underlying cause for device failures, if the device can communicate to your cloud application, it should send diagnostic information about the hardware failure to AWS IoT Core using a diagnostics topic. If the device loses connectivity because of the hardware failure, use Fleet Indexing with connectivity status to track the change in connectivity status. If the device is offline for extended periods of time, trigger an alert that the device may require remediation.

IOTREL13-BP03 Design devices to allow for remote configuration of message publication frequency

Devices may be developed with initial assumptions around how frequently messages need to be delivered, such as at a rate of 1Hz (1 message per second). When the device is deployed into its

destination environment, whether that is in a smart home setting, or a remote industrial asset, the network variability and other challenges may then require the need to alter this publication frequency. Planning ahead to allow for this type of configuration to be remotely managed will help with the reliability aspect of your IoT architecture.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL13-BP03-01 *Use either AWS IoT Jobs or AWS IoT device shadows to allow for the remote configuration of message publication frequency.*

AWS IoT Jobs can be used to push remote configuration changes to devices. AWS IoT device shadows can also be used to maintain device configuration. AWS IoT device SDKs provide support for integration with both of these features.

IOTREL14: How do you plan for disaster recovery in your IoT workloads?

When companies run their core production operations and cybersecurity functions in the cloud, it is important to design resilience at the edge & cloud in IoT systems. IoT implementations must allow for loss of internet connectivity, local data storage and processing.

IOTREL14-BP01 Design server software to initiate communication only with devices that are online

Communication should be server initiated with devices that are online rather than client-server requests. It enables you to design client software to accept commands from the server.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL14- BP01-01 *Design client software to accept commands from the server.*

- FreeRTOS provides pub/sub and shadow library to connected devices.
- AWS IoT Core provides device shadow capability to persist device states.
- AWS IoT Device Registry contains a list of devices connected to AWS IoT Core. AWS IoT Device Registry lets you manage devices by grouping them.

IOTREL14-BP02 Implement multi-Region support for IoT applications and devices

Cloud service providers have the same service in multiple Regions. You can use this architecture to divert device data to a Regional endpoint that is in not down. Data consumers should be enabled in all Regions that consume the diverted device data.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTREL14-BP02-01 *Architect device software to reach multiple Regions in case one is not available.*

- AWS IoT is available in multiple Regions with different endpoints. If an endpoint is not available, divert device traffic to a different endpoint.
- AWS IoT configurable endpoints can be used with Amazon Route 53 to divert IoT traffic to a new Regional endpoint.
- AWS IoT Configurable Endpoints: Domain configurations

Prescriptive guidance IOTREL14-BP02-02 *Enable device authentication certificates in multiple Regions.*

- AWS IoT provides devices with authentication certificates to verify on connection. Deploy the device certificates in the Regions where the device will connect.
- Setup the cloud side IoT data consumers to accept and process data in multiple Regions.
- AWS IoT device registration: Simplify IoT device registration and easily move devices between AWS accounts with AWS IoT Core Multi-Account Registration.

Prescriptive guidance IOTREL14-BP02-03 *Use device services in all Regions the device connects to.*

- AWS IoT Rules Engine diverts device data to use multiple services. Set up AWS IoT Rules Engine in the respective Regions to divert traffic to the appropriate services.
- Rules for AWS IoT

IOTREL14-BP03 Use edge devices to store and analyze data

Edge storage can provide additional storage for device data. Data can be stored at the edge during large-scale network events and streamed later, when network is available.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTREL14-BP03-01 *Use an edge device as a connection point to store and analyze data.*

- AWS IoT Greengrass can be used for local processing for serverless functions, containers, messaging, storage, and machine learning inference.
- Data can be stored in AWS IoT Greengrass and sent to the network when it's available.
- AWS IoT Greengrass features and components such a Stream Manager can be used to help design resilient solutions at the edge.

Key AWS services

Use Amazon CloudWatch to monitor runtime metrics and support reliability. Other services and features that support the three areas of reliability are as follows:

Foundations: AWS IoT Core enables you to scale your IoT application without having to manage the underlying infrastructure. You can scale AWS IoT Core by requesting account level limit increases.

Change management: AWS IoT Device Management enables you to update devices in the field while using Amazon S3 to version firmware, software, and update manifests for devices. AWS CloudFormation lets you document your IoT infrastructure as code and provision cloud resources using a CloudFormation template.

Failure management: Amazon S3 allows you to durably archive telemetry from devices. The AWS IoT rules engine Error action enables you to fall back to other AWS services when a primary AWS service is returning errors.

Resilience at the edge: AWS IoT Greengrass offers several features to help support data resiliency and backup needs with features which allow devices to communicate over the local network even after loss in internet connectivity, allowing the core to receive messages sent while the core is offline and using stream manager to process data locally until the connection is restored and send data to cloud or local storage destinations.

Resources

Refer to the following resources to learn more about our best practices related to reliability:

Documentation and blogs

- Using Device Time to Validate AWS IoT Server Certificates
- AWS service quotas
- Rules for AWS IoT
- Fleet Indexing
- IoT Atlas
- Resilience in AWS IoT Greengrass
- How to implement a disaster recovery solution for IoT platforms on AWS
- How to build smart applications using Protocol Buffers with AWS IoT Core
- Monitoring AWS IoT Events to maintain reliability, availability, and performance

Videos

- Implementing Multi-Region AWS IoT, ft. Analog Devices (IOT401)
- Ensuring IoT Application Edge Resilience with the Open Source AWS IoT Device Client
- AWS re:Invent 2023 - Scaling connected products and solutions with AWS IoT (IOT213)
- How to Get Started with AWS IoT Core Device Advisor

Performance efficiency

The performance efficiency pillar focuses on using resources efficiently. Key topics include selecting the right resource types and sizes based on workload requirements, monitoring performance, and making informed decisions to maintain efficiency as business and technology needs evolve. The performance efficiency pillar focuses on the efficient use of resources to meet the requirements and the maintenance of that efficiency as demand changes and technologies evolve.

IoT solutions involve heterogeneous infrastructure as it often integrates distributed field devices - such as smart sensors and gateways -, on-premises edge components and cloud services that must be orchestrated. Scalability is a key factor as these workloads frequently involve many decoupled components that may increase produced data dynamically through the device's operational lifetime

Focus areas

Design principles

In addition to the overall Well-Architected Framework performance efficiency design principles, there are four design principles for performance efficiency for IoT:

- **Use managed services**: AWS provides several managed services across databases, compute, and storage that can assist your architecture in increasing the overall reliability and performance.
- **Decouple ingestion and processing**: Decouple the connectivity portion of IoT applications from the ingestion and processing portion in IoT. By decoupling the ingestion layer, your IoT

application can handle data in aggregate and can scale more seamlessly by processing multiple IoT messages at once.

- **Use event-driven architectures**: IoT systems publish events from devices and permeate those events to other subsystems in your IoT application. Design mechanisms that cater to event-driven architectures include using queues, message handling, idempotency, dead-letter queues, and state machines.
- **Push device complexity away from deployed devices**: IoT hardware devices are frequently resource constrained. The overall logic tasks workflow of your IoT application must push compute and energy-intensive activities away from field-deployed devices and concentrate them in the cloud. However, data engineering, data contextualization, data enrichment and aggregation can be performed at the edge. This is often the case for OT/IIoT environments.

Definitions

There are five focus areas for performance efficiency:

- Architecture selection
- Compute and hardware
- Data management
- Networking and content delivery
- Process and culture

When designing your cloud architecture, take a data-driven approach to selecting the appropriate architecture pattern that aligns with your application's requirements. Carefully evaluate the infrastructure and connectivity options, such as virtual private clouds, subnets, and network gateways, to foster secure and efficient communication between your resources. Implement comprehensive monitoring solutions to track the performance, availability, observability, and security of your workloads, enabling proactive identification and resolution of issues. Adopt best practices for data management, including storage, backup, and data life cycle management, for improved data integrity, durability, and accessibility. Establish robust operational and maintenance processes, using automation and infrastructure as code (IaC), to streamline deployments, updates, and incident responses, which provides scalability and reliability for your IoT workloads.

Well-Architected IoT solutions are made up of multiple systems and components, such as devices, connectivity, databases, data processing, and analytics. In AWS, there are several IoT services,

database offerings, and analytics solutions you can use to quickly build solutions that are - architected Well-Architected so that you can focus on business objectives. AWS recommends that you use a mix of managed AWS services that best fit your workload. The following questions focus on these considerations for performance efficiency.

Architecture selection

The optimal embedded software for a particular system varies based on the hardware footprint of the device. For example, network security protocols, while necessary for preserving data privacy and integrity, can have a relatively large RAM footprint. For intranet and internet connections, use TLS with a combination of a strong cipher suite and minimal footprint.

IOTPERF01: How do your architectural decisions adapt to device hardware resources?

AWS IoT supports elliptic curve cryptography (ECC) for devices connecting to AWS IoT using TLS. A secure software and hardware system on device should take precedence during the selection criteria for your devices. AWS also has a number of IoT partners that provide hardware solutions that can securely integrate to AWS IoT.

In addition to selecting the right hardware partner, you might choose to use a number of software components to run your application logic on the device, including FreeRTOS and AWS IoT Greengrass. You can orchestrate native OS processes on specific hardware to improve performance and you also can run containerized workloads for isolation.

Defining and analyzing key performance metrics for your IoT applications helps you understand the performance characteristics for your application. Logging and end-to-end application monitoring are key for measuring, evaluating, and optimizing the performance of your IoT applications.

IOTPERF01-BP01 Optimize for device hardware resources utilization

When designing IoT solutions, it's crucial to consider the limited hardware resources available on edge devices, such as processing power, memory, and battery life. Optimizing resource utilization can significantly improve performance, efficiency, and overall device longevity. AWS offers several services and tools to help architects and developers optimize their solutions for device hardware constraints.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTPERF01-BP01-01 *Apply efficient runtimes and code language for embedded devices.*

When making architectural decisions, thoroughly evaluate the hardware capabilities of the target devices, and select appropriate AWS services and configurations to optimize resource utilization. This approach can lead to improved performance, extended battery life, and cost savings by reducing cloud processing and data transfer requirements. Some key tools to consider in your device components are:

- **AWS IoT Device Client SDK**: Provides lightweight, optimized libraries for various programming languages. These libraries enable efficient communication with AWS IoT Core and other AWS services, minimizing resource consumption on edge devices. For optimal performance, select based on your device constraints, prioritizing lower-level languages for battery-powered or resource-limited deployments:
  - **Embedded C SDK**: Best for highly constrained devices with minimal RAM (256KB+). Offers lowest memory footprint and power consumption.
  - **C++ SDK**: Balances performance with developer productivity for embedded Linux applications and gateways.
  - **Python, JavaScript, or Java SDKs**: Choose only when device resources permit, as they trade performance for ease of development, in general.
- **FreeRTOS:** A real-time operating system for microcontrollers that is designed to be resource-efficient and highly configurable. It allows developers to tailor the OS to specific hardware requirements, reducing the overall footprint.
- For more complex scenarios, AWS IoT Greengrass is an open source edge runtime and cloud service that helps you build, deploy, and manage device software. It manages devices to act locally on the data they generate, run predictions based on machine learning models, filter and aggregate device data, and only transmit necessary information to the cloud. By processing data locally, AWS IoT Greengrass minimizes network latency and optimizes resource utilization, particularly for time-sensitive or bandwidth-constrained applications.

Prescriptive guidance IOTPERF01-BP01-02 *Leverage edge gateways as hubs to bridge communications with the cloud.*

Edge gateways act as intermediaries, aggregating data from multiple devices, performing local processing and filtering, and intelligently managing communication with the cloud. This approach

offloads workloads from the cloud and reduces network traffic and latency. By implementing edge gateways with AWS IoT Greengrass, you can deploy AWS Lambda functions, machine learning models, and other application components directly on these gateways, enabling real-time processing and decision-making at the edge. This architecture not only enhances performance but also improves resilience by allowing continued operation during intermittent cloud connectivity, maintaining data integrity, and minimizing potential disruptions.

Resources

- AWS IoT Device SDKs, Mobile SDKs, and AWS IoT Device Client
- AWS IoT Greengrass GitHub
- FreeRTOS GitHub

Compute and hardware

IOTPERF02: How do you measure and maintain the performance of your IoT solution?

There are several key types of performance monitoring related to IoT deployments including device, cloud performance, and storage and analytics. Create appropriate performance metrics using data collected from logs with telemetry. Start with basic performance tracking and build on those metrics as your business core competencies expand.

Use Amazon CloudWatch Logs metric filters to transform your IoT application standard output into custom metrics through regular expression pattern matching. Create Amazon CloudWatch alarms based on your application's custom metrics to gain quick insight into your IoT application's behavior.

Set up fine-grained logs to track specific thing groups. During IoT solution development, enable `DEBUG` logging for a clear understanding of the progress of events about each IoT message as it passes from your devices through the message broker and the rules engine. In production, change the logging to `ERROR` and `WARN`.

In addition to cloud instrumentation, run instrumentation on devices prior to deployment to verify that the devices make the most efficient use of their local resources and that firmware code does not lead to unwanted scenarios such as memory leaks. Deploy code that is highly optimized for

constrained devices and monitor the health of your devices using device diagnostic messages published to AWS IoT from your embedded application.

IOTPERF02-BP01 Implement comprehensive monitoring solutions to collect performance data from your IoT devices

It is important to establish performance baselines and key performance indicators (KPIs) specific to your IoT devices and application requirements. These metrics may include device CPU and memory utilization, network bandwidth consumption, battery life, and embedded software-level metrics such as data throughput and latency. Additionally, depending on the programming language used, other memory metrics to consider are heap usage or garbage collection frequency (Java), memory leak detection and dynamic memory allocation ratio (C/C++), and memory pool utilization (Python).

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTPERF02-BP01-01 *Analyze device metrics and compare to a standard baseline.*

Collecting historical performance data from your devices helps you understand regular behavior for your deployments and potentially detect anomalies by using machine learning strategies and tools. Use AWS IoT Device Defender to audit device configurations, monitor device metrics, and detect deviations from expected behavior. Additionally, services like Amazon CloudWatch can be integrated to collect and analyze device performance metrics, set alarms, and run automated actions based on predefined thresholds.

IOTPERF02-BP02 Evaluate the runtime performance of your application

Application performance in production can be different from what you observe in a controlled test environment. Actively analyzing the performance of your application based on device health, network latency, and payload size provides insight on how to obtain performance improvements. By using different types of metrics, the health of each device in a multi-device setting can be obtained.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTPERF02-BP02-1 *Analyze connection patterns, sensor data and set up a device security profile to detect anomalies.*

- Measuring changes in connection patterns of devices might indicate some devices having a jittery network connection.
- Comparing device-side timestamps from multiple devices to arrival times on the cloud-side might indicate local network latency or additional hops in device path.

Resources

- Configure AWS IoT logging
- Gather system health telemetry data from AWS IoT Greengrass core devices
- AWS IoT Device Defender Detect
- How to detect anomalies in device metrics and improve your security posture using AWS IoT Device Defender

Data management

In the domain of IoT architectures, data management stands as a cornerstone for the workload's success. As devices proliferate and generate an ever-increasing volume of information, the way data is handled, processed, and stored becomes crucial for maintaining operational efficiency and extracting meaningful insights. Modern IoT solutions must navigate the complexities of managing data across different layers of the architecture, from edge devices to cloud storage, while facilitating optimal performance and cost-effectiveness.

The journey of IoT data, from its creation at the device level to its final destination in storage systems, requires careful consideration of multiple factors that impact the overall solution's effectiveness. These considerations become especially critical as organizations scale their IoT deployments from hundreds to thousands or even millions of connected devices. A well-designed data management strategy must address not only the immediate needs of data collection and storage but also account for future growth, regulatory requirements, and the evolving needs of downstream applications and analytics.

IOTPERF03: Does transmitted content include auditable metadata?

In IoT deployments, data integrity and traceability are fundamental aspects impacting operational reliability, compliance, and troubleshooting. Messages transmitted within an IoT landscape

must include essential metadata enabling comprehensive auditing and verification. This becomes increasingly critical as organizations face growing regulatory scrutiny and data lineage requirements.

The way IoT devices handle metadata throughout the transmission process is crucial for building trustworthy systems. The ability to track and verify message origin, timing, and handling distinguishes robust, enterprise-grade solutions from basic implementations. Organizations must implement metadata management strategies that support both operational requirements and compliance needs.

Comprehensive metadata practices influence multiple aspects of IoT solutions, from firmware design to cloud processing. A well-structured approach enables advanced capabilities like message deduplication, sequence verification, and temporal analysis, while providing crucial audit trails. As deployments scale, proper metadata becomes essential for maintaining system reliability and enabling sophisticated data analysis.

IOTPERF03-BP01 Add timestamps to each published message

Timestamps (ideally in UTC time) help in determining delays that might occur during the transmission of a message from the device to the application. Timestamps can be associated with the message and to fields contained in the message. If a timestamp is included, the sent timestamp is recorded on the cloud-side along with the sensor or event data.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTPERF03-BP01-01 *Add timestamps on the server side.*

- If the devices lack the capability to add timestamps to the messages, consider using server-side features to enrich the messages with timestamps that correspond to receiving the message.
- For example, AWS IoT Rules SQL language provides a `timestamp()` function to generate a timestamp when the message is received.
- When using AWS IoT Greengrass:
  - Use AWS IoT Greengrass stream manager to batch timestamped messages during connectivity interruptions while preserving message sequence integrity
  - Consider local AWS Lambda functions to process and enrich messages with timestamps closer to the source, minimizing latency between event occurrence and timestamp application

Prescriptive guidance IOTPERF03-BP01-02 *Have a reliable time source on the device.*

- Without a reliable time source, the timestamp can only be used relative to the specific device. For example:
  - Devices should use the Network Time Protocol (NTP) to obtain a reliable time when connected.
  - Real-time clock (RTC) devices can be used to maintain an accurate time while the device lacks network connectivity.
- Depending on the application, timestamps can be added at the message level or at the single payload field level. Delta encoding can be used to reduce the size of the message when multiple timestamps are included. Choosing the right approach is a trade-off between accuracy, energy efficiency, and payload size.

Resources

- AWS IoT Rules Developer Guide – timestamp() function
- The Implementation of Timestamp, Bitmap and RAKE Algorithm on Data Compression and Data Transmission from IoT to Cloud
- Delta encoding

IOTPERF04: Is there a mechanism for payload filtering or stream prioritization?

Firmware updates are critical, and filtering messages at the edge might subject the devices to unnecessary load. This result could be counterproductive from a power and memory consumption perspective. Sending only messages that the device makes use of reduces the load on the resources and supports better performances.

IOTPERF04-BP01 Have mechanisms to prioritize specific payload types

One strategy to address payload stream prioritization is to create multiple queues or data streams to separate and channel different payload types. For example, you could have dedicated queues or streams for real-time sensor data, firmware updates, and configuration messages. You can use this separation to apply different prioritization policies and processing rules based on the payload type's criticality and performance requirements.

Additionally, use protocol-level features such as Quality of Service (QoS) in MQTT to provide reliable and prioritized message delivery. By setting different QoS levels for different payload

types, you can prioritize the delivery of critical messages over non-critical ones and transmit high-priority data reliably and with minimal latency.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTPERF04-BP01-01 *Create multiple queues or data streams on the application side to channel different payload types.*

When working with publisher-subscriber type architectures, make sure to structure topics in the message broker following a scope and verb approach. With this strategy, you can subscribe to messages for a given scope (for example, a device) or refine the subscription on a given scope and verb.

Prescriptive guidance IOTPERF04-BP01-02 *Choose the right Quality of Service (QoS) for publishing the messages.*

- QoS 0 should be the default choice for all telemetry data that can cope with message loss and where data freshness is more important than reliability.
- QoS 1 provides reliable message transmission at the expense of increased latency, ordered ingestion in case of retries, and local memory consumption. It requires a local buffer for all unacknowledged messages.
- QoS 2 provides once and only once delivery of messages but increases the latency.

Resources

- Designing MQTT Topics for AWS IoT Core
- OASIS MQTT Version 5.0 QoS Specification

IOTPERF05: How do you optimize telemetry data ingestion?

IoT solutions drive rich analytics capabilities across vast areas of crucial enterprise functions, such as operations, customer care, finance, sales, and marketing. At the same time, they can be used as efficient exit points for edge gateways. Careful consideration must be given to architecting highly efficient IoT implementations where data and analytics are pushed to the cloud by devices and where machine learning algorithms are pulled down to the device gateways from the cloud.

Individual devices can be constrained by the throughput supported over a given network. The frequency at which data is exchanged must be balanced with the transport layer and the ability of the device to optionally store, aggregate, and then send data to the cloud. Send data from devices to the cloud at timed intervals that align to the time required by backend applications to process and take action on the data. For example, if you need to see data at a one-second increment, your device must send data at a more frequent time interval than one second. Conversely, if your application only reads data at an hourly rate, you can make a trade-off in performance by aggregating data points at the edge and sending the data every half hour.

The speed at which enterprise applications, business, and operations need to gain visibility into IoT telemetry data determines the most efficient point to process IoT data. In network constrained environments where the hardware is not limited, use edge solutions such as AWS IoT Greengrass to operate and process data offline from the cloud. In cases where both the network and hardware are constrained, look for opportunities to compress message payloads by using binary formatting and grouping similar messages together into a single request.

For visualizations, several AWS services can be used. Amazon Managed Service for Apache Flink can process streaming data in real-time using SQL. Additionally, QuickSight provides business intelligence dashboards for IoT data visualization with minimal setup. AWS IoT SiteWise offers purpose-built visualization tools for industrial equipment data. For operational monitoring, Amazon Managed Grafana enables time-series data visualization with pre-built IoT dashboards, while Amazon CloudWatch Dashboards can display IoT metrics and alarms.

Evaluating and optimizing your IoT application for its specific needs, whether telemetry data ingestion or controlling devices in the field, improves your outcomes in balancing performance and reliability within your hardware and network constraints. Separating the way that your application handles data collected through sensors or device probes from command-and-control flows helps achieve more reliable performance.

IOTPERF05-BP01 Identify the ingestion mechanisms that best fit your use case

Identify which data ingestion method best fits with your use case to obtain the best performance and operational complexity tradeoff. Multiple mechanisms might be needed. This method provides the optimal ingestion path for the data generated by your devices to obtain the best trade-offs between performance and cost.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTPERF05-BP01-01 *Evaluate ingestion mechanism for telemetry data.*

- Determine if the communication pattern is unidirectional (device to backend) or bi-directional. For example:
  - HTTPS should be considered over MQTT when your device is acting as an aggregator. Use multiple threads and multiple HTTP connections to maximize the throughput for high delay networks as HTTP calls are synchronous.
- Consider the APIs provided by the destination for your data and adopt them if you can securely access them. For example:
  - AWS IoT SiteWise provides an HTTP API to ingest operational data from industrial applications which needs to be stored for a limited period and processed as a time series with hierarchical aggregation capabilities.
  - Real-time video (for example, video surveillance cameras) has specific characteristics that makes it more suitable to ingest in a dedicated service, such as Amazon Kinesis Video Streams.
- Consider the need for data to be buffered locally while the device is disconnected and the transmission resumed as soon as the connection is re-established. For example:
  - AWS IoT Greengrass stream manager provides a managed stream service with local persistence, local processing pipelines, and exporters to Amazon Kinesis Data Streams, AWS IoT SiteWise, and Amazon S3.
- Consider the latency, throughput, and ordering characteristics of the data you want to ingest. For example:
  - For applications with a high ingestion rate (high-frequency sensor data) and where message ordering is important, Amazon Kinesis Data Streams provides stream-oriented processing capabilities and the ability to act as temporary storage.
  - For applications that do not have any real time requirements (such as logging, large images) and when the devices have the possibility to store data locally, uploading data directly to Amazon S3 can be both performant and cost efficient.

Resources

- Device communication protocols
- AWS IoT SiteWise adds support for 10 new industrial protocols with Domatica EasyEdge integration
- Amazon Kinesis Video Streams system requirements

IOTPERF05-BP02 Optimize data sent from devices to backend services

Optimizing the amount of data sent by the devices at the edge allows the backend to better meet the processing targets set by the business. Detailed data generated at the edge might have little value for your application in its raw form.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTPERF05-BP02-01 *Aggregate or compress data at the edge.*

Aggregate data points at the edge before sending them to the cloud, such as performing statistical aggregation, frequency histograms, signal processing to reduce payload size and consequently the load of the data transmission.

Resources

- Use AWS IoT SiteWise Edge gateways
- Cost-effectively ingest IoT data directly into Amazon S3 using AWS IoT Greengrass

IOTPERF06: How do you efficiently make sure stored data is usable by business?

You may have multiple databases in your IoT application, each selected for attributes such as the write frequency of data to the database, the read frequency of data from the database, and how the data is structured and queried. Consider some of these other criteria when selecting a database offering:

- Volume of data and retention period
- Intrinsic data organization and structure
- Users and applications consuming the data (either raw or processed) and their geographical location and dispersion
- Advanced analytics needs, such as machine learning or real-time visualizations
- Data synchronization across other teams, organizations, and business units
- Security of the data at the row, table, and database levels
- Interactions with other related data-driven events such as enterprise applications, drill-through dashboards, or systems of interaction

IOTPERF06-BP01 Store data in different tiers following formats, access patterns and methods

AWS has several database offerings that support IoT solutions. For structured data, you should use Amazon Aurora, a highly scalable relational interface to organizational data. For semi-structured data that requires low latency for queries and will be used by multiple consumers, use Amazon DynamoDB, a fully managed, multi-Region database that provides consistent single-digit millisecond latency and offers built-in security, backup and restore, and in-memory caching.

AWS also provides specific data storage solutions for industrial use cases with AWS IoT SiteWise. For equipment data, three tiers are available:

- A hot storage tier optimized for real-time applications
- A warm storage tier optimized for analytical workloads
- A cold storage tier using Amazon Simple Storage Service (Amazon S3) for operational data applications with high latency tolerance

SiteWise helps you to reduce storage cost by keeping recent data in the hot storage tier for at least 30 days and moving historical data to a cost-optimized warm storage tier based upon user-defined data retention policies.

Use Amazon SageMaker AI AI to build, train, and deploy machine learning models based on your IoT data, in the cloud, and on the edge using AWS IoT services, such as machine learning inference in AWS IoT Greengrass.

Consider storing your raw formatted time series data in a data warehouse solution such as Amazon Redshift. Unformatted data can be imported to Amazon Redshift using Amazon S3 and Amazon Data Firehose. By archiving unformatted data in a scalable, managed data storage solution, you can begin to gain business insights, explore your data, and identify trends and patterns over time.

In addition to storing and leveraging the historical trends of your IoT data, you should have a system that stores the current state of the device and provides the ability to query against the current state of all of your devices. This supports internal analytics and customer facing views into your IoT data.

The AWS IoT Device Shadow service is an effective mechanism to store a virtual representation of your device in the cloud. AWS IoT Device Shadow service is best suited for managing the current state of each device.

In addition, for internal teams that need to query against the shadow for operational needs, use the managed capabilities of fleet indexing, which provides a searchable index incorporating your IoT registry and shadow metadata. If there is a need to provide index-based searching or filtering capability to a large number of external users, such as for a consumer application, dynamically archive the shadow state using a combination of the IoT rules engine, Amazon Data Firehose, and Amazon OpenSearch Service to store your data in a format that allows fine grained query access for external users.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTPERF06-BP01-01 *Create automated mechanisms to transition data from tiers to implement lifecycles.*

In an IoT solution, data often follows a lifecycle, transitioning from real-time ingestion to historical storage and archiving. To efficiently verify that stored data is utilizable throughout its lifecycle, it's crucial to implement automated mechanisms that transition data across different storage tiers based on predefined rules and policies.

For example, you can use AWS IoT rules and AWS Lambda functions to automatically route incoming real-time data to Amazon DynamoDB or Amazon Timestream for low-latency access and processing. As the data ages, you can initiate automated processes to transition it to Amazon S3 or Amazon Glacier for cost-effective, long-term archival storage.

Additionally, you can implement data retention policies and lifecycle management rules within your data storage solutions. For instance, in Amazon DynamoDB, you can configure Time to Live (TTL) settings to automatically expire and remove data after a specified period, improving the efficiency of storage utilization and reducing costs.

By creating automated mechanisms to transition data across different storage tiers, you can optimize storage costs, make data accessible based on its lifecycle stage, and maintain data integrity and availability for various analytical and operational use cases throughout the data lifespan.

Resources

- Managing the lifecycle of objects
- Backing up and restoring Timestream tables: How it works
- RetentionProperties

- Manage data storage in AWS IoT SiteWise
- Configure storage settings in AWS IoT SiteWise

Networking and content delivery

Maintaining optimal connectivity between edge devices and cloud infrastructure is ideal for the performance and reliability of IoT solutions.

IOTPERF07: How do you provide optimal connectivity for edge devices communicating to cloud infrastructure?

To address connectivity issues, architects should consider the following best practices.

IOTPERF07-BP01 Optimize network topology for distributed devices

Carefully design the network topology to minimize latency and foster efficient data transfer between edge devices and the cloud. This may involve implementing edge gateways or hubs, using AWS IoT Greengrass, and optimizing network configurations based on the geographical distribution and density of devices.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance IOTPERF07-BP01-01 *Configure deployed devices to connect to the lowest latency cloud endpoint of your application's cloud infrastructure.*

To minimize latency and provide optimal performance, configure edge devices to connect to the nearest edge endpoint of your application's cloud infrastructure. This can be achieved by using AWS IoT Core's device data endpoint feature, which allows devices to connect to the closest AWS IoT Core endpoint based on their geographic location. By connecting to the nearest cloud endpoint, data transmission times are reduced, improving responsiveness and overall application performance, especially for time-sensitive or latency-critical use cases.

Resources

- AWS IoT Coredata plane endpoints

IOTPERF07-BP02 Perform timely connectivity verification for devices

Implement mechanisms to regularly verify and monitor the connectivity status of IoT and edge devices to quickly detect connectivity issues. This can be achieved through periodic heartbeat messages, device shadows, or using AWS IoT Device Management fleet indexing for continuous monitoring and alerting.

Timely connectivity verification enables proactive troubleshooting and minimizes potential disruptions in data flow between edge devices and the cloud. AWS IoT Device Management fleet indexing can query a group of devices and aggregate statistics on device records that are based on different combinations of device attributes, including state, connectivity, and device violations. With fleet indexing, you can organize, investigate, and troubleshoot your fleet of devices.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance IOTPERF07-BP02-01 *Set up a heartbeat message publishing routine to analyze connectivity status and quality.*

Alternatively, to effectively monitor device connectivity, implement a heartbeat mechanism where devices periodically send messages containing a monotonically increasing counter and a timestamp. This approach enables validating message loss by detecting gaps in the counter sequence and assessing consecutive message delays by analyzing timestamp differences. The heartbeat frequency can be adjusted based on the application's requirements. This mechanism provides visibility into connectivity status, message integrity, and latency for individual devices, facilitating proactive issue detection and remediation. This second approach may be useful for custom monitoring of devices that require specific polling frequencies. However, it is important to notice there is a significant tradeoff associated with this choice in terms of higher messaging costs when compared to the usage of the optimized AWS IoT Device Management fleet indexing.

Resources

- Example: Device HeartBeat to monitor device connections with AWS IoT Events
- Fleet indexing

Process and culture

IoT applications can be simulated using production devices set up as test devices (with a specific test MQTT namespace), or by using simulated devices. Incoming data captured using the IoT rules engine is processed using the same workflows that are used for production.

The frequency of end-to-end simulations must be driven by your specific release cycle or device adoption. You should test failure pathways (code that is only run during a failure) to verify that the solution is resilient to errors. You should also continually run device canaries against your production and pre-production accounts. The device canaries act as key indicators of the system performance during simulation tests. Document the outputs of the tests, draft remediation plans, and perform user acceptance tests.

IOTPERF08: How do you make sure application operates within its scaling limits?

IOTPERF08-BP01 Load test your IoT applications

Applications can be complex and have multiple dependencies. Testing the application under load helps identify problems before going into production. Load testing your IoT applications verifies that you understand the cloud-side performance characteristics and failure modes of your IoT architecture. Testing helps you understand how your application architecture operates under load, identify performance bottlenecks, and apply mitigating strategies prior to releasing changes to your production systems.

Prescriptive guidance IOTPERF08-BP01-01 *Simulate the real device behavior.*

- A device simulator should implement the device behavior as closely as possible. Test not only message publishing, but also connections, reconnections, subscriptions, enrollment, and other contextual events such as constrained network bandwidth. Start testing at a lower load, and progressively increase to 100%. Additionally, consider exercising the workload beyond the traditional expected load by performing stress tests.
  - Start the load test at a low percent of your estimated total device fleet (for example, 10%).
  - Evaluate the performance of your application using operational dashboards created to measure end-to-end delivery of device telemetry data and automated device commands.
  - Make any necessary changes to the application architecture to achieve desired performance goals.
  - Iterate these steps increasing the load until you get to 100%.
  - For further workload development, consider performing stress tests beyond usual load expected

Resources

- IoT Device Simulator

IOTPERF08-BP02 Monitor and manage your IoT service quotas using available tools and metrics

Be aware of the adjustable and unadjustable quotas of the AWS service, and continuously monitor the key performance indicators so that you can anticipate when actions must be taken to request increases in the service quotas and re-evaluate your architecture. Verify that your application operates within the quotas of the services that you are building on to provide the optimal performance to your users.

Monitoring keeps you aware of which service quotas you might be reaching so that you can change your application to cope with the unadjustable quotas or to request the increase of an adjustable quota with sufficient lead time.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTPERF08-BP02-01 *Be aware of the service quotas of the different IoT services.*

- Pay attention to which limits are adjustable quotas and which are unadjustable quotas as they require different approaches. For example:
  - An unadjustable *quota*, such a control plane request rate, requires changes in the application behavior to avoid the event repeating too often. Workarounds for unadjustable quotas might require different design decisions, such as using multiple accounts. It's good to know the unadjustable and adjustable quotas in advance so that you can make these design decisions as early as possible in the development process.
  - *Adjustable quotas* should be monitored to anticipate the need for additional capacity and provide sufficient notice so that a request for a limit increase can be made well ahead of time. For example:
    - For AWS IoT Core, alert on `RulesMessageThrottles`, `Connect.ClientIDThrottle`, `Connect.Throttle`, `PublishIn.Throttle`, `Subscribe.Throttle`, `Unsubscribe.Throttle`.
    - For AWS IoT Device Management, monitor active continuous jobs, and active snapshot jobs in Service Quotas

Resources

- AWS IoT Core endpoints and quotas
- AWS IoT Device Defender endpoints and quotas
- AWS IoT Device Management endpoints and quotas
- AWS IoT Greengrass V2 endpoints and quotas
- AWS IoT SiteWise endpoints and quotas

IOTPERF09: How do you maintain visibility over the distributed infrastructure deployed?

IOTPERF09-BP01 Have device inventory in the IoT system that centralizes device configuration and diagnostics

As the number of devices increases, monitor for performance bottlenecks when all the devices connect to the cloud-side. These devices could generate a large aggregate amount of data. To verify that you understand where to improve, gather device diagnostics to determine the immediate health of a device and any other devices in its proximity.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance IOTPERF09-BP01-01 *Deploy an agent to the device to start capturing the relevant diagnostic data.*

- For microprocessor-based applications, consider deploying the AWS Systems Manager Agent (SSM Agent) so that you can continuously monitor your device's performance metrics.
- There are sample agents provided to use on the device-side (device or gateway). If device-side diagnostic metrics cannot be obtained, then it is possible to obtain limited cloud-side metrics. For example:
  - TCP connections
    - Connections
    - Local-interface
  - Listening TCP/UDP ports
    - Listening-TCP/UDP-ports

    - Interface
  - Network statistics
    - Bytes-in/out
    - Packets-in/out
    - Network-statistics
- To define and monitor metrics that are unique to your fleet or use case, use custom metrics, such as number of devices connected to Wi-Fi gateways, charge levels for batteries, or number of power cycles for smart plugs.

Prescriptive guidance IOTPERF09-BP01-02 *Measure, evaluate, and optimize device firmware updates with strategies such as canary deployment.*

Firmware updates are critical to keep your IoT devices performant over time, but these updates might not always have the expected impact. As you deploy firmware updates to your devices, monitor your KPIs to verify that updates do not have any unintended impacts to the performance of your hardware devices or to your IoT applications.

- Deploy new firmware to a limited set of devices, and monitor the impact on performance before rolling the update out to the entire fleet. Stop deployment if degradation is detected.
- Use AWS IoT Jobs to manage over-the-air (OTA) updates and configure it to deploy to a limited set of devices.
- After the update, evaluate end-to-end performance of the system using your previously identified KPIs.
- If performance characteristics appear to have been impacted after the firmware release, use AWS IoT secure tunneling, a feature of AWS IoT Device Management, to remotely troubleshoot the device.
- Release additional firmware updates to remediate identified issues.

Resources

- Custom metrics
- Using Continuous Jobs with AWS IoT Device Management
- Using Device Jobs for Over-the-Air Updates

- Introducing Secure Tunneling for AWS IoT Device Management, a new secure way to troubleshoot IoT devices

Key AWS services

The key AWS services for performance efficiency include Amazon CloudWatch, AWS IoT Core, AWS IoT Device Defender, AWS IoT Device Management, AWS Lambda, and Amazon DynamoDB. Amazon CloudWatch provides visibility into your application's overall performance and operational health. The following services support performance efficiency for IoT workloads:

- **Architecture selection:** :AWS hardware partners provide production ready IoT devices that can be used as part of you IoT application. Consider using AWS device software such as AWS device SDK's, FreeRTOS, AWS IoT Greengrass for software on IoT devices.
- **Compute and hardware:** AWS IoT Greengrass allows you to run local compute, messaging, data caching, synchronization, and ML at the edge. Amazon CloudWatch metrics and Amazon CloudWatch Logs provide metrics, logs, filters, alarms, and notifications that you can integrate with your existing monitoring solution. These metrics can be augmented with device telemetry to monitor your application.
- **Data management:** AWS IoT Rules and AWS IoT Core support for MQTT QoS levels facilitate data routing, filtering, and prioritization to applications. Amazon Kinesis Data Streams enables real-time ingestion of high-velocity IoT data, while Amazon DynamoDB's time-to-live (TTL) feature optimizes storage utilization by automatically removing expired data. Amazon Timestream, a purpose-built time series database, offers configurable data retention properties, providing cost-effective historical data storage in the memory or in the magnetic store. AWS IoT SiteWise is a managed service that enables industrial enterprises to collect, store, organize, and visualize thousands of sensor data streams across multiple industrial facilities.
- **Networking and content delivery:** AWS IoT Core is a managed IoT service that supports MQTT, a lightweight publish and subscribe protocol for device communication. For optimal performance efficiency, AWS Direct Connect provides dedicated network connections for high-throughput IoT workloads. AWS IoT Core's support for MQTT (both v3.1.1 and v5.0 specifications, with documented differences) can be complemented by Amazon API Gateway for RESTful communications and AWS AppSync for GraphQL-based real-time data synchronization across IoT fleets.
- **Process and culture:** AWS IoT Device Management enables inventory control, remote device fleet management and updates at scale, while Amazon CloudWatch provides comprehensive monitoring with custom metrics and alarms to proactively identify performance bottlenecks

across your IoT infrastructure. The AWS IoT Device Simulator allows simulating large-scale IoT deployments for thorough testing. AWS IoT Core Jobs facilitates reliable over-the-air device software updates, complemented by AWS Systems Manager for automated patch management and operational tasks. AWS X-Ray offers deep tracing capabilities to analyze and debug IoT applications, helping identify latency issues across distributed components. AWS Config enables tracking configuration changes that might affect efficiency. Amazon EventBridge can orchestrate automated responses to operational events, supporting rapid remediation of performance issues across your IoT landscape.

Resources

- Get Started with FreeRTOS
- What is AWS IoT Greengrass?
- The Internet of Things on AWS – Official Blog

Cost optimization

The cost optimization pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your first proof of concept to the ongoing operation of production workloads, adopting the practices in this paper helps you build and operate cost-aware systems that achieve business outcomes and minimize costs, allowing your business to maximize its return on investment.

Focus areas

Design principles

In addition to the overall Well-Architected Framework cost optimization design principles, there are three design principle for cost optimization for IoT:

- **Manage manufacturing cost tradeoffs**: Business partnering criteria, hardware component selection, firmware complexity, and distribution requirements all play a role in manufacturing cost. Minimizing that cost helps determine whether a product can be brought to market successfully over multiple product generations. However, making inefficient decisions in the selection of your components and manufacturer can increase downstream costs. For example, partnering with a reputable manufacturer helps minimize downstream hardware failure and customer support cost. Selecting a dedicated crypto component can increase bill of material (BOM) cost, but reduce downstream manufacturing and provisioning complexity, since the part may already come with an onboard private key and certificate.
- **Avoid unnecessary data access, storage, and transmission**: Identify and classify data collected throughout your IoT landscape and learn their corresponding business use-case. Collect the data

that is required, applying filtering mechanisms at the edge to minimize sending redundant data to the cloud.

- **Process data at the edge whenever possible**: Processing data at the edge can help to save costs. Process large volumes of data locally, upload only relevant insights and high-value data to the cloud.

Definitions

There are four focus areas for cost optimization:

- Practice Cloud Financial Management
- Expenditure and usage awareness
- Cost-effective resources
- Managing demand and supplying resources

There are tradeoffs to consider. For example, do you want to optimize for speed to market or cost? In some cases, it's best to optimize for speed (going to market quickly, shipping new features, or meeting a deadline) rather than investing in upfront cost optimization. Design decisions benefit from empirical data and thoughtful benchmarking rather than being rushed because of deadline, enabling teams to identify cost-efficient solutions that deliver optimal performance while maximizing return on investment. Rushed decisions could lead to over-provisioned and under-optimized deployments. The following sections provide techniques and strategic guidance for your deployment's initial and ongoing cost optimization.

Practice Cloud Financial Management

There are no Cloud Financial Management best practices specific to the IoT Lens.

Expenditure and usage awareness

There are no expenditure and usage awareness best practices specific to the IoT Lens.

Cost-effective resources

Given the scale of devices and data that can be generated by an IoT application, using the appropriate AWS services for your system is key to cost savings. In addition to the overall cost for

your IoT solution, IoT architects often look at connectivity through the lens of bill of materials (BOM) costs. For BOM calculations, you must predict and monitor what the long-term costs will be for managing the connectivity to your IoT application throughout the lifetime of that device. AWS services can help you calculate initial BOM costs, make use of cost-effective services that are event driven, and update your architecture to continue to lower your overall lifetime cost for connectivity.

The recommended approach to increase the cost-effectiveness of your resources is to group IoT events into batches and process data collectively. By processing events in groups, you are able to lower the overall compute time for each individual message. Aggregation can help you save on compute resources and enable solutions when data is compressed and archived before being persisted. This strategy decreases the overall storage footprint without losing data or compromising the query ability of the data.

IOTCOST01: How do you choose cost-efficient tools for data aggregation of your IoT workloads?

AWS IoT is best suited for streaming data for either immediate consumption or historical analyses. There are several ways to batch data from AWS IoT Core to other AWS services and the differentiating factor is driven by batching raw data (as is) or enriching the data and then batching it. Enriching, transforming, and filtering IoT telemetry data during (or immediately after) ingestion is best performed by creating an AWS IoT rule that sends the data to other AWS services such as Kinesis Data Streams, Firehose or Amazon SQS. These services allow you to process multiple data events at once.

When dealing with raw device data from this batch pipeline, you can use Amazon Data Firehose to transfer data to S3 buckets and Amazon Redshift. To lower storage costs in Amazon S3, an application can use lifecycle policies that archive data to lower cost storage, such as Amazon S3 Glacier.

Raw data from devices can also be processed at the edge using AWS IoT Greengrass thus alleviating the need to send all the data to the cloud for storage and processing. This can result in lower network cost and lower cost in cloud services. Customers can dynamically change or update that logic, as well as frequency of transmission using AWS IoT Greengrass since it's not hardcoded and can be adjusted as needed by the use case. This gives customers added flexibility for cost optimization.

Methods and tools for how data is acquired, validated, categorized, and stored impacts the overall cost of your application. Focusing on tools that can automatically vary in scale and cost with demand and support your data with a minimum of administrative overhead can help you achieve lowest cost for your application. By considering the data pipeline from origination to archival, you can make informed decisions and examine tradeoffs among technical and business requirements to identify the most effective solution.

IOTCOST01-BP01 Use a data lake for raw telemetry data

A *data lake* brings different data sources together and provides a common management framework for browsing, viewing, and extracting the sources. An effective data lake enables IoT cost management by storing data in the right format for the right use case. With a data lake, storage and interaction characteristics can be aligned to a specific dataset format and required interfaces.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

- For each telemetry stream, identify key features of telemetry using the 4Vs of big data—velocity, volume, veracity, and variety.
- Map each stream into the appropriate storage capability.
- For example, a stream that sends an MQTT message with a JSON payload every second would be an ideal candidate for being batched, compressed then stored in Amazon S3.
- For high velocity data streaming, utilize IoT Basic Ingest and AWS IoT rules to route data to the appropriate storage service such as Amazon Timestream or Kinesis Data Streams.

Resources

- AWS storage types
- AWS re:Invent 2018: Building with AWS Databases: Match Your Workload to the Right Database (DAT301)
- AWS IoT rule actions

IOTCOST01-BP02 Provide a self-service interface for end users to search, extract, manage, and update IoT data

With flexible cloud computing resources, pay-as-you-go pricing, and strong identity and encryption controls, your organization should allow groups to define and share data models in the format that makes the most sense for them. Self-service interfaces encourage experimentation and speed up change by removing barriers for teams to gain access to the data they need to make decisions.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

- Use an architecture that allows various end users to easily find, obtain, enhance, and share data
- Use a subscriber model, which allows teams to subscribe to data feeds and receive notification of updates, to reduce the need for active polling and re-synching with data sources

Resources

- Creating a data lake from a JDBC source in AWS Lake Formation
- AWS Data Lake Quick Start
- AWS Data Exchange offers subscriptions to third-party data sources with notification on updates

IOTCOST01-BP03 Track and manage the utilization of data sources

Applications and users evolve over time, and IoT solutions can generate large volumes of data quickly. As your application matures, it's important for cost management of your IoT workload to track that data collected is still being used. Consistent tracking and review of data utilization provides an objective basis for cost optimization decisions.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

- Track access rates and storage trends for your data lake sources.
- Use automated guidance tools, such as AWS Cost Explorer and AWS Trusted Advisor, to identify under-utilized or resizable components of your workload. AWS Cost explorer has a forecast

feature that predicts how much you will use AWS services over the forecast time period you selected.

- Use AWS Budgets and Cost Anomaly detection to help prevent surprise bills.

Resources

- Monitoring Amazon S3 metrics with Amazon CloudWatch
- Find cost of your S3 buckets using AWS Cost Explorer
- Forecast your spend using Cost explorer
- Best practices for AWS Budget

IOTCOST01-BP04 Aggregate data at the edge where possible

Data aggregation is an architectural decision that can have impacts on data fidelity. Aggregations should be thoroughly reviewed with engineering and architectural teams before implementation. If the device can aggregate data before sending for processing using methods such as combining messages or removing duplicate or repeating values, that can reduce the amount of processing, the number of associated resources, and associated expense.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

- A common mechanism includes combining multiple status updates to a final status, or combining a series of measurements generated by the device into a single message.
- For example, 10000 of device telemetry data might be packaged as one 10000 message, two 5000 messages, or ten separate 1000 messages. Each packaging format has implications outside of cost such as network traffic (ten 1000 messages will each add their own header messaging as opposed to a single 10000 message with one header) and the impact of a lost or delayed message. Optimizing message size should consider how a lost message impacts the functional or non-functional characteristics of the system.
- Use cost calculators to model different approaches for message size and count

IOTCOST02: How do you optimize cost of raw telemetry data?

Raw telemetry is an original source for analytics but can also be a major component of cost. Analyze the flow and usage of your telemetry to identify the right service and handling process required. Select storage and processing mechanisms that match the needs of your specific telemetry case.

IOTCOST02-BP01 Use lifecycle policies to archive your data

When selecting an automated lifecycle policy for data, there are tradeoffs to consider. For example, do you want to optimize for speed to market or cost? In some cases, it's best to optimize for speed rather than investing in upfront cost optimization. Use your organization's data classification strategies to define a lifecycle policy to take raw telemetry measurements through various services. Setting milestones by time sets expectations and encourages aggregation and production of data over mere collection.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

- Check your organization's data management policy for requirements on retention, deletion, and encryption, and align your retention policies and tools with those guidelines.
- S3 Lifecycle policies or S3 Intelligent-Tiering can move the data to the most cost-effective Amazon S3 storage class or Amazon S3 Glacier for long-term storage.

IOTCOST02-BP02 Evaluate storage characteristics for your use case and align with the right services

Not all data needs to be stored in the same way, and data storage needs change through a data item's lifecycle. A growing fleet of devices can exponentially scale its messaging rate and device operation traffic. This scaling of message volumes can also mean an increase in storage costs.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

- For data at high scale of devices, time, or other characteristics, consider a data warehouse such as Amazon Redshift or Amazon S3 with Amazon Athena. The data partitioning and tiering features of AWS storage services can help reduce storage costs.

- For data at lower scale of time, devices, or other characteristics, consider Amazon DynamoDB, Amazon OpenSearch Service (OpenSearch Service), or Aurora for short-term historical data. Use your data lifecycle policies to optimize what is kept in the short-term storage.

IOTCOST02-BP03 Store raw archival data on cost effective services

Using the right storage solution for a specific data type will align costs with usage.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

- Use an object store, such as Amazon S3, for raw archival storage. Object stores are immutable and often more efficient and cost-effective than block storage, especially for data which doesn't require editing.
- Avoid costs by using a schema-on-read service, such as Amazon Athena, to query the data in its native form. Using Athena can help reduce the need for large-scale storage arrays or always-on databases to read raw archival data.

IOTCOST03: How do you optimize cost of interactions between devices and your IoT cloud solution?

Interactions to and from devices can be a significant driver of your workload's overall cost. Understanding and optimizing interactions between devices and cloud solution can be a significant factor of cost management.

IOTCOST03-BP01 Select services to optimize cost

Understand how services use and charge for messaging, as well as operating modes that offer cost benefits. Understanding service billing characteristics can help you identify ways to optimize messaging, which could result in considerable cost savings at scale.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

- Review your IoT architecture to find communication patterns and scenarios that could map to service discount features.
- With AWS IoT Core Basic Ingest, you can publish directly to a rule without messaging charges.
- Use your registry of things only for primarily immutable data, such as serial Number.
- For your device's shadow, minimize the frequency of reads and writes to reduce the total metered operation and your operating costs.

Resources

- AWS IoT Rules Engine Basic Ingest
- AWS IoT Pricing

IOTCOST03-BP02 Implement and configure telemetry to reduce data transfer costs

Matching the precision of telemetry data, such as number of decimal places, to the precision of the required calculation can help address both the overall message size and the precision of calculations.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

- Reduce string lengths and decimal precision where feasible. For example, strings sent regularly such as POWER or CHARGE could be reduced to P or C strings. Similarly, decimal values such as 21.25 or 71.86 could be reduced to 21 or 72 if the additional precision is not required for the application. This is common in room temperature readings where precision beyond is whole number is rarely required. Across many millions of messages, the savings from removing a few letters can make a significant difference in message size and cost.

IOTCOST03-BP03 Use shadow only for slow changing data

Shadow is used in IoT applications as a persistence mechanism of device state. The shadow maintains data that remains consistent across multiple points in time. Device shadow operations can be billed and metered differently than publish or subscribe messages. Reducing the shadow

update frequency from the device can reduce the number of billed operations while maintaining an acceptable level of data freshness.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

- Avoid using shadow as a guaranteed-delivery mechanism or for continuously fluctuating data. As a workload scales up, the cost of frequent shadow updates could exceed the value of the data.
- Consider MQTT Last Will and Testament (LWT) as a mitigation for the risk of loss of device communication instead of using shadow.
- Use the AWS Pricing Calculator to compare device shadow interactions versus telemetry messages to understand cost implications.

IOTCOST03-BP04 Group and tag IoT devices and messages for cost allocation

You can use an IoT billing group to tag devices by categories related to your IoT application. Create tags that represent business categories, such as cost centers. Visibility into devices and messages by category makes cost dimensions easier to understand and visualize.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

- Use AWS IoT Core Billing Groups to tag IoT devices for cost allocation. Add tracking elements to messages and devices to help trace source, such as using MQTT5 User Properties to add product model and serial number.
- Verify that your system can group and organize data by both technical and business entity.

IOTCOST03-BP05 Implement and configure device messaging to reduce data transfer costs

Charges for different cloud and data transporter providers can vary based on specifics of message size and frequency. IoT workloads can cross multiple communication, such as cell networks, and each layer can have its own metering and pricing standards.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

- Evaluate tradeoffs between message size and number of messages. Frequency optimization is performed with payload optimization to both accurately assess the network load and identify adequate trade-offs between frequency and payload size.
- For example, your devices might send one message per second. If you could aggregate those messages on the device and send five observations in a single message every five seconds, that could drastically reduce your message count and cost.
- Use MQTT5 and topic aliases to reduce message size and cost by replacing long topic strings with integers.
- Use the AWS Cost Calculator to compare the cost of using messaging services like Kinesis and API Gateway to offload components of your IoT workload.

Managing demand and supplying resources

Optimally matching supply to demand delivers the lowest cost for a system. However, given the susceptibility of IoT workloads to data bursts, solutions must be dynamically scalable and consider peak capacity when provisioning resources. With the event driven flow of data, you can choose to automatically provision your AWS resources to match your peak capacity, and then scale up and down during known low periods of traffic.

IOTCOST04: How do you optimize cost by matching the supply of resources with device demand?

Serverless technologies, such as AWS Lambda and Amazon API Gateway, help you create a more scalable and resilient architecture, and you only pay when your application uses those services. AWS IoT Core, AWS IoT Device Management, AWS IoT Device Defender, and AWS IoT Greengrass are also managed services that charge based on usage, not for idle compute capacity. The benefit of managed services is that AWS manages the automatic provisioning of your resources. If you use managed services, you are responsible for monitoring and setting alerts for limit increases for AWS services.

IOTCOST04-BP01 Plan expected usage over time

When architecting to match supply against demand, proactively plan your expected usage over time, and the limits that you are most likely to exceed. Factor those limit increases into your future planning.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

Evaluating new AWS features helps you optimize cost by analyzing how your devices are performing. You can use this analysis to make changes to how your devices communicate with your IoT.

To optimize the cost of your solution through changes to device firmware, review the pricing components of AWS services, such as AWS IoT, determine where you are below billing metering thresholds for a given service, and then weigh the trade-offs between cost and performance.

IOTCOST05: How do you optimize payload size between devices and your IoT system to save cost?

IoT applications must balance the networking throughput that can be realized by end devices with the most efficient way that data should be processed by your IoT application. We recommend that IoT deployments initially optimize data transfer based on the device constraints. Begin by sending discrete data events from the device to the cloud, making minimal use of batching multiple events in a single message. Later, if necessary, you can use serialization frameworks to compress the messages prior to sending it to your IoT system.

From a cost perspective, the MQTT payload size is a critical cost optimization element for AWS IoT Core. An IoT message is billed in five KB increments, up to 128 KB. For this reason, each MQTT payload size should be as close to possible to any five KB. For example, a payload that is currently sized at 6 KB is not as cost efficient as a payload that is ten KB because the overall costs of publishing that message is identical despite one message being larger than the other.

IOTCOST05-BP01 Balance networking throughput against payload size to optimize efficiency

The specific use case drives the balance between frequency and payload size. Consider and test different payload optimization strategies. Additionally, consider trade-offs between compression and processing overhead.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

- Shorten values while keeping them legible. If five digits of precision are sufficient, then you should not use 12 digits in the payload.
- Use serialization frameworks to compress payloads to smaller sizes if you do not require IoT rules engine payload inspection.
- Send data less frequently and aggregate messages together within the billable increments. For example, sending a single two KB message every second can be achieved at a lower IoT message cost by sending two separate two KB messages every other second.

This approach has tradeoffs that should be considered before implementation. Adding complexity or delay in your devices may unexpectedly increase processing costs. A cost optimization exercise for IoT payloads should only happen after your solution has been in production and you can use a data-driven approach to determine the cost impact of changing the way data is sent to AWS IoT Core.

IOTCOST06: How do you optimize the costs of storing the current state of your IoT device?

Well-Architected IoT applications have a virtual representation of the device in the cloud. This virtual representation is composed of a managed data store or specialized IoT application data store. In both cases, your end devices must be programmed in a way that efficiently transmits device state changes to your IoT application. For example, your device should only send its full device state if your firmware logic dictates that the full device state may be out of sync and would be best reconciled by sending all current settings. As individual state changes occur, the device should optimize the frequency it transmits those changes to the cloud.

In AWS IoT, device shadow and registry operations are metered in one KB increments and billing is per million access and modify operations. The shadow stores the desired or actual state of each device and the registry is used to name and manage devices.

IOTCOST06-BP01 Optimize shadow operations

Cost optimization processes for device shadows and registry focus on managing how many operations are performed and the size of each operation. If your operation is cost-sensitive to shadow and registry operations, explore ways to optimize shadow operations. For example, for the shadow you could aggregate several reported fields together into one shadow message update instead of sending each reported change independently. Grouping shadow updates together reduces the overall cost of the shadow by consolidating updates to the service.

Level of risk exposed if this best practice is not established: High

Prescriptive guidance

- **Use named shadows:** Separate logical elements, and reduce the size of updates.
- **Aggregate shadow updates:** Look for opportunities to put several reported fields together into one shadow message update instead of sending each reported change independently. Grouping shadow updates together reduces the overall cost of the shadow by consolidating updates to the service.
- **Send only what is needed, when it is needed:** For example, your device should only send its full device state if your firmware logic dictates that the full device state may be out of sync and would be best reconciled by sending all current settings. As individual state changes occur, the device should optimize the frequency it transmits those changes to the cloud.
- **Immutable data:** Use the AWS IoT device registry device attributes for immutable data such as a serial number.
- **Minimize the frequency of reads and writes:** Where possible, limit updates to device's shadow document to reduce the total metered operations.
- **Choose the right service:** Avoid using shadows as a guaranteed-delivery mechanism or for continuously fluctuating data. Consider MQTT Last Will and Testament (LWT) as a mitigation for the risk of loss of device communication instead of using shadows.

Key AWS services

The key AWS feature supporting cost optimization is cost allocation tags, which help you to understand the costs of a system. The following services and features are important in the three areas of cost optimization:

- **Cost-effective resources**: Amazon Kinesis and Amazon S3 are AWS services that enable you to process multiple IoT messages in a single request in order to improve the cost effectiveness of compute resources.
- **Managing demand and supplying resources**: AWS IoT Core is a managed IoT service for managing connectivity, device security to the cloud, messaging routing, and device state.
- **Optimize over time**: The AWS IoT Blog section on the AWS website is a resource for learning about what is newly launched as part of AWS IoT.

Resources

Refer to the following resources to learn more about AWS best practices for cost optimization.

- AWS IoT Blogs
- Cost Optimization Tips for AWS IoT Workloads

Sustainability

The sustainability pillar includes the ability to continually improve sustainability impacts by reducing energy consumption and increasing efficiency across all components of a workload by maximizing the benefits from the provisioned resources and minimizing the total resources required.

The term sustainability encompasses a wide range of factors such as economic viability, social equity, biodiversity, resilience, and environmental impact. We focus specifically on how to design your IoT solutions to improve sustainability by lowering their carbon footprint while simultaneously reducing operational costs.

Reducing carbon footprint is a key aspect of mitigating climate change, which is a pressing global environmental challenge. One way to achieve this is to use resources in a responsible and efficient manner, minimize waste and maximize the use of renewable resources. As described later in this paper, the actions taken to improve sustainability usually have a positive effect on operational costs, not just through resource efficiency, but also by optimizing operational processes.

While IoT solutions span the range from sensors and devices at the edge to applications in the cloud, this paper focuses largely on sustainability at the edge. Sustainability of and through the cloud is covered in the Sustainability Pillar of the AWS Well-Architected Framework.

It is important to recognize that the carbon footprint of an IoT device is distributed across its entire lifecycle, consisting of the design and build phase, the operational (in-use) phase, and the disposal phase.

- The design and build phase deals with design, component sourcing, manufacturing, and other supply chain activities. This phase usually accounts for most of the carbon footprint associated with a device, often referred to as the *embodied carbon*.
- The operational phase of the device lifecycle has a significant carbon footprint impact as well. Reaching Net-Zero Carbon by 2040: Decarbonizing and Neutralizing the Use Phase of Connected Devices shows that this phase accounts for 10-15% of the entire lifecycle carbon footprint for re-chargeable battery powered devices and 60-80% for plugged-in devices.
- The disposal phase of IoT devices refers to the stage when a device is no longer needed or usable in operation.

Implementing the best practices outlined here involves trade-offs between cost, reliability, performance, carbon footprint, and any current or future regulatory requirements. Your organization should examine these best practices, both holistically and individually, and agree on how to best meet your sustainability goals for each use case.

Focus areas

- Design principles
- Definitions
- Region selection
- Alignment to demand
- Software and architecture optimization
- Software and architecture – Cloud
- Data management
- Hardware and services - Hardware optimization
- Hardware and services - Power management
- Process and culture - User guidance
- Key AWS services
- Resources

Design principles

The IoT Lens sustainability pillar includes 24 design principles that advise you on how to judiciously use and right size your AWS services to manage your IoT workloads and resources. These cover processor, storage, power, OS and programming environments, OTA and networking strategies, and messaging parameters for your IoT workloads.

Design principles

- Right-size your hardware
- Considerations for General Purpose IoT Devices
- Choose the right CPU
- Choose a processor to minimize the energy used by your workload
- Choose a processor with advanced power management features

- Use accelerators for machine learning inference
- Choose storage that supports device longevity
- Choose a power source with high efficiency
- Dimension and manage batteries to maximize battery life
- Choose an operating system that is appropriate for the type of device's features and functionality
- Use an event driven architecture in your IoT devices
- Choose a power efficient programming language
- Optimize ML models for the edge
- Use Over-The-Air device management
- Adopt power conservation practices appropriate to your wireless technology
- Choose a lightweight protocol for messaging
- Reduce the amount of data transmitted
- Reduce the distance traveled by data
- Optimize log verbosity
- Buffer and spool messages
- Optimize the frequency of messages for your use case
- Use gateways to offload and pre-process your data at the edge
- Perform analytics at the edge
- Monitor and manage your fleet operations to maximize sustainability

Right-size your hardware

The choices made in hardware component selection can greatly influence the operational phase impact of devices and therefore must be carefully considered. The use case should dictate the required CPU/MCU, peripherals and communication modules on the device. When choosing the processor, amount of RAM, and amount of flash storage, the designer must focus on resource optimization; over-provisioning hardware can lead to choosing a processor that has a large power draw but stays mostly idle or has an excess of memory that never gets used, increasing the overall carbon footprint of the device unnecessarily. Under-sizing hardware can lead to long execution times or a lack of extensibility necessary to insure the longevity of the device.

Your design will typically fall into one of three broad classes of devices – Microcontroller (MCU) class devices, Microprocessor (MPU) class devices, and devices that use hardware acceleration for machine learning use cases (which we refer to as "Inference class" devices).

Considerations for General Purpose IoT Devices

For general purpose IoT devices, the application and use case will determine the choice between microcontroller (MCU) and microprocessor (MPU). MCU devices are typically designed for low-power, often autonomously battery-operated, and resource constrained applications. MPU class devices are designed for applications that require higher computational power, multitasking, and higher-level services provided by more capable operating systems.

For MCU devices, there are several features that can contribute to improved energy efficiency and sustainability.

- Support for multiple low power modes, allowing power consumption to be reduced when the processor is not active.
  - This should include a mode that retains volatile memory content, allowing for quick restoration of application state when required.
- A Real-Time Clock (RTC) to wake-up the device from a low power mode only when needed.
- On-chip or on-board crypto-accelerators to enable secure communication while minimizing energy consumption.
- Connectivity module low-power and sleep modes to reduce unnecessary power drain when not in use.
- Connectivity module should utilize quick reconnect protocols
- If required, support for a floating point unit (FPU) for more efficient and faster processing of numerical calculations, contributing to improved energy efficiency.

Similar criteria also apply to the selection of MPUs, though there are differences. Some MPUs utilize a combination of high-performance and low-power cores, such as those using the big.LITTLE architecture. In such architectures, make surethat tasks are assigned to the appropriate core based on the workload. This reduces power consumption while maintaining sufficient processing capabilities. The FreeRTOS operating system can take advantage of this architecture through Asymmetric Multiprocessing (AMP). Each core runs an instance of FreeRTOS and communicates through shared memory space and inter-process communication (IPC). The smaller core can be

assigned to run non-intensive applications and the big core can be put into low-power mode until needed for more compute intensive tasks.

Choose the right CPU

To optimize energy consumption, choose a CPU that meets the performance requirements of your application without exceeding them. Keep in mind that a more powerful processor may have higher performance per watt compared to a less powerful processor, and can perform more work per unit of power consumed, resulting in lower energy consumption. Newer CPU architectures or higher-end processors may have better power efficiency at higher workloads due to advancements in technology or microarchitecture, resulting in a more energy-efficient solution overall. Use performance per watt as a guideline, and test CPUs against your actual workload to make a final choice.

Choose a CPU that is designed for power efficiency. These CPUs feature sleep and idle states to reduce power consumption during periods of inactivity. These states allow the CPU to minimize power usage by either reducing the clock frequency, halting execution, or shutting down certain components when they are not required.

The Instruction Set Architecture (ISA) of a CPU, which defines its instruction set and programming model, can also impact power consumption. Some features, such as complex out-of-order execution and speculative execution, can increase power consumption due to the increased hardware complexity and activity. RISC (Reduced Instruction Set Computing) architectures, such as ARM and RISC-V, are generally known for their power efficiency compared to CISC (Complex Instruction Set Computing) architectures, such as x86.

Choose a processor to minimize the energy used by your workload

Workload characteristics impact the power effectiveness of a processor. Some workloads, such as data-intensive or compute-intensive tasks, may require more processing power to achieve a desired level of performance. In such cases, a more powerful processor with higher power consumption may be able to complete the workload faster, resulting in shorter overall runtime and potentially lower total energy consumption than a less powerful processor running the same workload for a longer duration. If a workload is optimized for a particular processor architecture or has specific requirements that can be better met by a more powerful processor, then using that processor may result in lower energy consumption overall, despite its higher power consumption. Test with your expected workload to help narrow down a processor choice.

Choose a processor with advanced power management features

Look for CPUs with advanced power management features, such as dynamic voltage and frequency scaling (DVFS), clock gating, and idle state management. These features can help dynamically adjust the CPU's performance and power consumption based on workload requirements and system conditions, leading to improved power efficiency.

The Thermal Design Power (TDP) of a CPU specifies the amount of heat it is expected to dissipate under maximum or typical load conditions. Higher temperatures impact power efficiency negatively due to increased leakage currents and resistance. Look for CPUs with a low TDP, as these typically consume less power and generate less heat, which can help the overall power efficiency of a system. In addition, low TDP CPUs can help alleviate the need for active cooling systems and the associated power consumption.

Finally, analyze the entire system, including components such as memory, storage, and peripherals, as they can also impact the overall power consumption. Optimizing the system-level power management, including power states, sleep modes, and power management settings, can contribute to improved power efficiency.

Use accelerators for machine learning inference

Performing Machine Learning (ML) inference on the IoT device can greatly reduce the amount of data transmitted to the cloud. ML inference is a computationally intensive process which might not be energy optimal when run on a CPU without the right instruction set. For ML applications, the use of a CPU with additional vector operations and specialized acceleration hardware can lower the energy consumption of IoT devices.

Inference-class devices can come in various forms such as GPUs (Graphics Processing Units), NPUs (Neural Processing Units), Digital Signal Processors (DSPs) and FPGA (Field-Programmable Gate Array) devices or CPUs with vector manipulation operators.

If there are no performance or latency constraints in the workload it might be preferable to run inference on standard MCU devices, for cost and energy savings. For further optimization, resources such as TinyEngine and other frameworks can be used to run machine learning models directly on the microcontroller.

When lower latency than what is achievable by the MCUs is desired, specialized hardware accelerators for ML tasks, such as built-in Digital Signal Processors (DSPs) or ML accelerators, can provide efficient ML inferencing at the edge with lower power consumption.

DSPs are designed to perform mathematical functions quickly without using the host MCU's clock cycles. They are more power efficient than an MCU. In a typical DSP-accelerated ML application, such as wake-word detection in a smart speaker, the DSP first processes an analog signal such as an audio or voice signal and then wakes up the host MCU from a deep-sleep mode via an interrupt. The processor can thus remain in a low-power mode while performing inference, and only wakes up when necessary for further processing or connection to the cloud.

MPUs are suitable for edge ML applications that require higher processing capabilities, such as running more complex ML models or handling larger input datasets. MPUs may also have built-in hardware accelerators for ML tasks, such as Neural Processing Units (NPUs) which improves ML inferencing performance.

NPUs are optimized for artificial neural networks. If your application involves inference using deep neural networks, such as image recognition, natural language processing, or recommendation systems, an NPU can provide inference acceleration and an order of magnitude or more energy efficiency compared to general-purpose CPUs or GPUs.

GPUs are specialized processors designed for graphics-intensive tasks, but can offer high performance for ML inference. If you are already using a deep learning framework or software that is optimized for GPUs, it may be more convenient to continue using GPUs. GPUs are not power efficient and should only be selected for the highest intensity workloads.

Choose storage that supports device longevity

There are several low-power memory storage options that are suitable for IoT devices due to their energy-efficient characteristics. When choosing storage, the use case and workload dictates much of the decision. For example, for a device that is deployed to a harsh environment that is not easy to access, choose storage that is durable and resilient. Also consider the wear and tear on flash memory due to the workload, and its impact on device longevity. Provision the amount of storage beyond current requirements for the application to allow for additional use cases in the future, thereby extending the device's useful life. In summary, the appropriate type and amount of storage for the device application should be carefully chosen as it can have an impact on the operational phase of the device as well as the overall cost.

File system choices can impact processor and memory requirements for the device, in turn impacting the power draw and ultimately sustainability. Choosing a lightweight file system such as littleFS or Reliance Edge for embedded systems can help reduce power consumption and increase the life-time of the storage through features such as wear leveling, power-efficient file updates, power-safe operations, small footprint, and customizable configuration options.

For IoT use cases that demand higher power edge gateways, various disk drives or solid-state drives may also be used. Using techniques like RAID, though more power intensive, can improve the performance and resiliency of gateways or servers and in turn improve the overall sustainability of the system by reducing the need for truck rolls.

A key consideration for storage selection is to support dual partitioning of persistent storage for Over the Air Update (OTA) capability. In addition, the flash subsystem must be capable of independent erase and write operations for each partition (or image block).

When deciding on storage options for a use case, satisfying just the requirements of the application will not always be the most sustainable practice. Think about the performance of the storage over the life of the device, whether the device has enough storage to support buffering/analysis of data to alleviate the need to transmit, and if the storage supports extensibility and longevity of devices to reduce the operational impact and carbon footprint of upgrades and replacements.

Choose a power source with high efficiency

Efficiency of the power source is a crucial factor to consider in IoT device design. It is recommended to choose power supplies with at least 90% efficiency across all load conditions, such as 80PLUS Titanium-rated power supplies. This can help minimize wasted energy and reduce the overall power consumption of the IoT device, thus lowering the device's carbon footprint.

Dimension and manage batteries to maximize battery life

If the IoT device is powered by a battery or secondary battery, it is important to consider the minimum useful lifetime of the battery under normal deployment conditions. This impacts how frequently a technician visit is required for battery replacement, which in turn affects operational costs and potential waste from battery replacements. In addition, implementing techniques such as low-power sleep modes can extend battery life and reduce required battery dimensions. This consideration can help reduce operational costs and minimize the environmental impact of your IoT devices.

Choose an operating system that is appropriate for the type of device's features and functionality

The operating system can greatly impact how power efficient an IoT device is.

Try to choose an operating system that is appropriate for the type of device's features and functionality. For example, choosing Android or Linux for a simple temperature sensor may be more convenient for the developer, but is not sustainable practice due to its impact on the device's resource dimensions and consequent carbon footprint.

Consider features in the operating system that support low power modes, power management, and operational efficiency. Operating systems must support low power modes that allow the system to conserve power when idle or when certain subsystems are not in use. For example, an embedded system might use a sleep mode to conserve power when the user is not interacting with the system, or it might shut down peripherals or subsystems that are not currently in use. The operating system should expose the hardware's low power modes such that applications can take advantage of these modes.

Some operating systems support tick-less operation, where the operating system avoids unnecessary timer interrupts, further reducing power consumption. For example, FreeRTOS stops the periodic tick interrupt during periods when there are no application tasks that are able to execute. Stopping the tick interrupt allows the microcontroller to remain in a deep power saving mode until an event occurs or the kernel is ready to execute a task.

Dynamic voltage and frequency scaling (DVFS) operations adjust the CPU performance and frequency based on the application workload's demands. Operating systems that support DVFS can decrease the CPU's voltage in real time during decreased workloads, reducing power consumption.

Use an event driven architecture in your IoT devices

Using an event-driven architecture in IoT device firmware reduces processing and communication overhead, and helps reduce energy consumption. There are a number of mechanisms that can be used in IoT devices to realize an event-driven architecture.

Design device software to minimize the amount of energy IoT devices consume while checking for events or data. By using interrupts instead of continuous polling, the CPU can sleep until an event occurs, reducing the time the device spends actively checking for data or events.

Another mechanism is Pub/Sub topics (such as those used by the MQTT protocol). Devices that subscribe to specific topics receive notifications when new messages are available, allowing for appropriate processing to be triggered.

Asynchronous callbacks are another technique used to realize an event driven architecture. In this model, the device firmware registers callback functions to be executed when specific events occur. This enables efficient processing of events without the need for polling or continuous querying.

Real time operating systems such as FreeRTOS are built for event driven firmware by employing the use of priority interrupts, pointers to callback functions, and multi-threading.

Choose a power efficient programming language

Programming languages have varying levels of efficiency when it comes to power consumption. Some languages, such as C and C++, are known for their low-level, close-to-hardware nature, which allows for fine-grained control over system resources and can result in more power-efficient code. On the other hand, interpreted languages like Python or JavaScript, may have higher levels of abstraction and runtime overhead, which can result in several times higher power consumption, as outlined in this AWS blog.

Modern compilers often provide optimization options that can reduce code size, remove unnecessary computations, and optimize register usage, which can lead to more power-efficient code. For code sections that are executed at a high frequency (or are time sensitive), enabling compiler optimizations, such as loop unrolling, function inlining, and dead code reduction, can result in more efficient code execution and lower power consumption.

Optimize ML models for the edge

Leveraging edge computing for machine learning (ML) inference and analytics can offer several benefits compared to processing data in the cloud. Pre-processing and real-time data analysis and decision making can be done locally, reducing the need for frequent data transfers to the cloud, thereby reducing messaging costs, computing power requirements, and the energy consumption associated with data transmission as well as processing in the cloud.

Machine learning is a computationally expensive task. Choose machine learning frameworks and algorithms that are optimized for low-power consumption and can run on hardware accelerators. Models can be developed in the cloud, and then optimized for edge devices with frameworks such as the Open Neural Network Exchange, ONNX, before being deployed to the device.

The size and complexity of ML models can impact their suitability for edge deployment. Techniques such as model quantization, compression, and pruning can help reduce the size of ML models, making them more suitable for deployment on edge devices.

ML models should be observable while deployed on edge devices in order to detect if the model is being affected by changing data or model drift. When negative model performance is detected, updating the model over the air increases the device's useful lifetime, reducing the need for replacement.

Use Over-The-Air device management

Over-the-air device management refers to operations to update, secure and configure your IoT devices from the cloud. Devices that go offline unexpectedly or have critical security vulnerabilities often require expensive site visits or shipment of a device to a refurbishment center. To reduce the carbon footprint and operational costs across your device fleet, your device must support one or more capabilities for over-the-air device management.

Over-the-air (OTA) updates

OTA updates enable IoT devices to receive and install software or configuration updates without the need for physical access to the devices. This mechanism is supported across a variety of embedded operating systems, including embedded Linux and FreeRTOS, and can be powered by the AWS IoT Jobs agent and cloud service .

The OTA update mechanism for IoT devices must be resilient, reliable, and secure in order to help prevent a failed update from requiring a truck roll to fix the device. There are several techniques that device makers can use to build a resilient OTA mechanism.

OTA updates may encounter errors or failures during the update process, such as network errors, data corruption, or other issues. The device's firmware should have error handling and recovery mechanisms in place to detect and recover from potential errors, such as checksum verification, error correction codes (ECC), redundancy, or other suitable techniques to support data integrity and reliability.

Atomic updates are updates that are either fully applied or fully rolled back, without leaving the system in an inconsistent state. This can be achieved by storing a new version of the firmware on an inactive partition and then swapping it with the active firmware partition upon successful completion. This approach also supports safe roll-back in case the new firmware encounters errors.

Once an OTA update is installed and running correctly, the device must help prevent it from being rolled back to a previous vulnerable version. This can be achieved through mechanisms such as secure bootloaders, cryptographic signatures, or other techniques that help prevent the device from reverting to older, potentially less secure firmware/software versions.

OTA updates can also be used for certificate rotation when there is a security incident or to renew an expiring certificate. A certificate rotation mechanism extends the device's lifetime without the need to use long-lived certificates and allows the device to benefit from improvements in security algorithms and ciphers that might not have been available at the time the device was manufactured.

Devices on networks such as LoRaWAN and NB-IoT face additional challenges. These networks are constrained in bandwidth, making OTA updates power inefficient for large file transfers. Sending large files over the air is also problematic for battery-powered devices connected to more power-hungry Wi-Fi and cellular networks. To overcome this issue, instead of sending the entire firmware image to a device, send only the portions of the image that have actually changed, reducing communication and processing required, and reducing power consumption. The Delta Over the Air Updates feature supported by FreeRTOS uses this approach.

Remote Access

To reduce the need to send people on site in case of a malfunctioning system, it is recommended to provide a remote access capability on the device. When IoT devices are deployed in the field, remote access provides a way to troubleshoot, change the configuration, access files such as logs, and perform other operational tasks even if the device is behind a firewall or private network. Users can update devices through its command line interface or access the device's package manager to add new software via Secure Shell (SSH) or Remote Desktop Protocol (RDP)

Use AWS IoT Secure Tunneling to establish bidirectional communication to remote devices over a secure connection that is managed by AWS IoT. Secure tunneling does not require updates to your existing inbound firewall rules, so you can keep the same security level provided by firewall rules at a remote site without adding operational overhead

AWS Systems Manager is another AWS service that you can use to view and control your edge devices. Systems Manager enables you to view operational data, automate operation tasks, and maintain security and compliance through remote device access.

Choose a communication technology that is optimal for your use case

The communications layer deals with connectivity to a network, message routing among remote devices, and routing between devices and the cloud. Communications, whether over a wired or wireless network, can be a significant consumer of power and compute for an IoT device. Care must be taken to minimize this power draw when designing the hardware of a device as well as the application. Choose an optimal connectivity type from options available at the device's operating location to support data transfer with minimal power, optimal connected time and minimal retries.

Wireless Connectivity

For wireless connectivity there are many design choices that need to be made. For nearly all cases, the most power efficient applications are those that minimize the amount of time that the radio or

network interface is on. For example, an LTE-M module's active transmission mode has thousands of times higher power consumption than its Power Save Mode (PSM). Typically, the transmission phase utilizes the most power and can be particularly penalizing to battery powered devices.

There are a few best practices applicable to Low Power Wide Area (LPWA) use cases:

- Reduce the amount of time the radio is on
- Use the appropriate technology for the use case. Using the wrong technology type can lead to data retransmission and reduced power efficiency
- Reduce the amount of data transmitted
- Use LTE network session resumption mechanisms as much as possible to reduce lengthy handshakes
- Use advanced error handling techniques and advanced buffering techniques to properly manage degraded network conditions
- Optimize application and network settings to improve the chances of successful communication
- Monitor your device fleet via network information to continuously optimize device applications

Given below are details on specific wireless technologies, along with guidance from the above list that is specific to each technology.

Cellular technologies

There are several different generations of cellular technology that can be used in IoT applications. Given this range of choices, it is important to pick the right technology for your use case in order to optimize power consumption and cost, as different technologies will have different power implications. These technologies are discussed below.

LTE categories above Cat. 1, 5G Sub-6GHZ, and 5G mm Wave

These categories of cellular technologies are high power and high bandwidth technologies that are suited for use cases such as wireless cameras that require high bandwidth mobility. Using these types of radio access technologies is not appropriate for low power use cases. These higher bandwidth technologies leverage many different features such as carrier aggregation and various frequency deployments to deliver high bandwidth. Applications should buffer and segment any data that is streaming to the device such as audio and video. The application should also keep the radio off or the radio should be allowed to enter into an idle state (RRC Idle or RRC Inactive) if the use case does not dictate that the device remain available.

LTE Cat 1, Cat 1-bis, and 5G RedCap

The technologies in this category fall just above LTE-M and Narrowband IoT in terms of data upload and download throughput supported. These technologies are full featured LTE technologies and support use cases broadly from fleet asset tracking to voice in use cases like alarm panels. The technologies in this category should be chosen when connectivity in transit is required and devices require data throughput speeds that are more than 1 Mbps but less than roughly 5Mbps. 5G RedCap will offer significantly improved data throughput in this category.

LTE-M

LTE-M is a category of LTE that falls solidly into the set of LPWA technologies. It uses less power than LTE Cat. 1 and provides less throughput as well. LTE-M uses a feature known as coverage enhancement to provide better coverage to devices by means of repeating signaling. LTE-M should be used for devices that require long battery life or deep penetrating coverage. LTE-M also supports mobility, and use cases like asset tracking which do not have tight deadlines for real time location streaming can take advantage of low power and low cost LTE-M radios.

Narrowband IoT

Narrowband IoT (NB-IoT) is another category of LTE that is an LPWA technology. It uses similar coverage enhancement techniques as LTE-M but improves upon overall power consumption by utilizing a narrower bandwidth. Consequently, NB-IoT devices support lower data throughput compared to LTE-M. NB-IoT does not support connected mode mobility and is much less suited to mobile use cases. If a device must be mobile the device's application must manage the mobility and should expect session resumptions at each cell. NB-IoT is well suited to stationary use cases that require long battery life, enhanced coverage, and utilize optimized software to reduce overall data transmission.

2G

2G network deployments are still a source of connectivity for global IoT deployments and are used widely in some legacy device deployments. Many new LPWA radio modules still support 2G along with LTE-M and Narrowband-IoT to offer as close to global coverage as possible, offering a single SKU solution that is a more sustainable choice than building multiple SKUs. Only choose 2G as a primary technology if there is no other coverage type in the deployment area; otherwise leverage the newer technologies for better power optimization features.

3G

In most cases, carriers have sunset or have plans to sunset 3G networks, and 2G networks that exist will likely outlive their 3G counterparts. It is not recommended to design new 3G products or to expect 3G fallback for LTE devices as the phase-out of 3G will impact the longevity of the device and ultimately the overall sustainability of the solution due to replacements.

Application Note for LPWA Cellular Networks

Applications should not be considered immediately portable between LPWA technology types and the higher bandwidth technology types. Applications leveraging LTE-M or Narrowband IoT should have advanced error processing to accommodate for slow throughput and high retransmission rates. The application should also use advanced methods for reduction of the size of packets sent. This includes reducing the size of TLS handshakes using pre-shared keys as well as using CBOR or some other application layer mechanism to reduce the data payload size. Developers should make the application aware of when the device enters into coverage enhancement modes. Coverage enhancement modes can greatly reduce the data throughput rate and greatly increase various errors within the TLS stack, IP stack, and the application due to timeouts or retransmissions. In order to maintain an LPWA device and improve its longevity, developers should take note of the elements called out in this section in order to reduce the overall time the radio is on and sending or receiving data.

Adopt power conservation practices appropriate to your wireless technology

Cellular

In addition to the broad use cases mentioned above, there are a number of considerations to take into account to make an energy efficient choice for cellular connectivity. For devices that communicate using LTE modems, select network operators that support eDRX (Extended Discontinuous Reception) and PSM (Power Save Mode). These are two complementary features that can be used together to optimize power consumption in LTE devices.

In PSM, LTE devices enter a low-power sleep mode for extended periods of time, waking up at pre-defined intervals to reconnect to the network without a full negotiation, greatly reducing power consumption. With PSM, while the device is asleep the radio can be considered to be completely off, in contrast to eDRX, which continues a paging cycle. This means that if a page or network-initiated downlink request were to be made, the device in PSM mode would not receive anything. This can result in an increased disconnected window compared to eDRX.

eDRX allows the device to more precisely define the interval between listening periods, which can reduce power consumption while still maintaining connectivity. This is useful for devices that need to remain connected to the network for longer periods of time, such as mobile devices. This increased paging window means that latency between a request to the device is increased but the device is nonetheless available.

PSM is more power efficient than eDRX. However, there is an inflection point at which it may make more sense for a device to use one versus the other. While building an application you should test your particular hardware for overall power consumption during a typical window of sleep and active states. Use both PSM and eDRX at the predefined eDRX paging windows and then choose the methodology that best optimizes your device's power usage during its typical usage. eDRX is also a more recent technology and might not be supported by all network providers.

Wi-Fi

Wi-Fi is a widely used connectivity option for applications that require high speed local connectivity within a limited range. These include verticals such as Smart Home, Industrial IoT, Healthcare, and Retail.

Wi-Fi has several power-saving features that can help to reduce power consumption in wireless devices. Some of the most common power-saving features in Wi-Fi include:

- Power Save Mode (PSM): This feature allows Wi-Fi devices to enter a low-power sleep mode when not in use, reducing power consumption. During PSM, the device periodically wakes up to check for incoming data, and then returns to sleep mode. u-APSD allows devices to selectively wake up from PSM to receive only the specific data packets that are intended for them. This allows devices to remain in PSM for longer periods of time and further reduces power consumption.
- Wake-on-Wireless-LAN (WoWLAN): This feature allows devices to wake up from sleep mode when a wireless LAN signal is detected, rather than requiring a physical button press or other intervention.
- Dynamic Frequency Selection (DFS): This feature allows devices to avoid using certain Wi-Fi channels that may be subject to interference, thus reducing power consumption.
- Transmit Beamforming (TxBF): This feature allows devices to optimize their transmission patterns based on the location of the receiving device, which can improve signal quality and reduce power consumption.
- 802.11n Power Save: This feature is designed specifically for 802.11n networks and allows devices to enter a low-power sleep mode during periods of low network activity.

- Target Wake Time (TWT) is a feature of Wi-Fi 6 that allows devices to schedule when they will communicate with the wireless network. TWT lets a device turn on its radio interface only when it needs to communicate with the network, thereby reducing power consumption and extending the battery life of IoT devices.
- OFDMA is a key feature of Wi-Fi 6 that improves the way data is transmitted between access points and multiple IoT devices. OFDMA divides the Wi-Fi channel into smaller sub-channels, known as resource units (RUs), and assigns each RU to a specific device or group of devices. This way, an AP can communicate with multiple devices simultaneously, and each device gets a fair share of the channel's capacity.
- BSS coloring is a feature introduced in the 802.11ax Wi-Fi standard that helps reduce interference from neighboring access points (APs) and improve coexistence between multiple APs. The basic idea behind BSS coloring is that each BSS or AP uses a unique color (6-bit code) which is carried by the signal preamble or SIG field. The color allows client devices to differentiate between the signals of neighboring APs and avoid interference.

The power consumption in idle mode can be reduced through features such as frame bursting and low power listen, while the power consumption in sleep mode can be reduced through features such as deep sleep and wake-on-Wi-Fi.

LoRaWAN

LoRaWAN is suitable for long-range, low power applications such as smart agriculture, asset tracking and smart cities.

To maximize power savings on a LoRaWAN gateway or device, you can consider the following configurations:

- Choose the appropriate device class. LoRaWAN defines class A, B, and C device types. These device types have different receive window configurations that significantly impact the power consumption of a LoRaWAN device.
- Where possible use FUOTA (Firmware Update Over The Air) over multicast for efficient file transfer to many devices receiving the same firmware.
- Choose hardware components, such as processors and radios, that have low power consumption to minimize energy usage.
- Position the gateway antenna for maximum coverage and signal strength, to reduce the need for high transmit power.

- Set the gateway to use a lower transmit power that still provides good coverage in the target area. This reduces energy usage and can increase the lifetime of the gateway.
- Use Listen Before Talk (LBT) and Adaptive Data Rate (ADR) - these features can help minimize the number of retransmissions and collisions, which can reduce energy usage.
- Configure the gateway to use the appropriate packet size and data rate for the application. Large packet sizes and high data rates can increase energy usage.
- Configure the gateway to use deep sleep mode when it is not actively transmitting or receiving data. This can significantly reduce energy usage.

Amazon Sidewalk

Amazon Sidewalk allows endpoint devices to utilize the pre-existing Amazon Sidewalk network, enabling service providers and utilities to deploy monitoring networks without the operational expense, carbon footprint and overhead of deploying and managing gateways. Amazon Sidewalk makes use of existing Amazon Echo and Ring devices to create a low bandwidth network that devices can use to communicate with the AWS cloud. The Amazon Sidewalk network is built such that devices can roam onto existing Sidewalk gateways and the gateway can use its network connection to route data to cloud services. This creates a low power, best-effort community network.

Amazon Sidewalk devices inherently support a number of features that help optimize power consumption. Sidewalk uses power-efficient wireless communication protocols such as Bluetooth Low Energy (BLE). It also uses adaptive transmission power, which adjusts the transmit power of connected devices based on the distance between devices, reducing power consumption and extending battery life. Devices connected to Amazon Sidewalk also enter a low-power sleep mode when not in use, which also extends battery life.

Choose a lightweight protocol for messaging

IoT devices can use a number of application layer protocols to communicate with each other as well as the cloud. The most common protocol for IoT devices is MQTT.

MQTT is designed to be a lightweight and power efficient protocol for IoT applications that require low power consumption and low bandwidth usage. It uses a publish-subscribe model, which allows devices to exchange small messages with each other using a minimal amount of network bandwidth.

HTTP, on the other hand, is a more heavyweight protocol that is used for web-based applications. While HTTP is not specifically designed for IoT applications, it is widely used for sending and receiving data over the internet. HTTP uses a request-response model, which requires devices to send larger amounts of data back and forth over the network. This can result in higher power consumption and shorter battery life for IoT devices.

Reduce the amount of data transmitted

Data transmission is one of the most power-intensive operations for IoT devices, especially when using wireless communication protocols such as Wi-Fi, Bluetooth, or cellular networks. The overarching principles to be followed for messaging are to reduce the amount of data transmitted and reduce the distance traveled by that data.

The amount of data transmitted between an IoT device and the cloud can be reduced using message compression, using binary protocols, reducing the frequency at which messages are sent, and using transport and application layer protocols that are efficient.

IoT devices often collect large amounts of sensor data. By storing the data on the device, it can be processed and analyzed in real-time, without the need to transmit it to the cloud for analysis. This can help reduce transmission power and costs, particularly in cases where cellular or satellite connectivity is used. Storing sensor data on the device can provide an additional level of redundancy and fault tolerance, which can reduce the need for human intervention or site visits in case of extended connectivity loss.

Given that most IoT devices are resource constrained, apply compression to data generated by IoT devices, including sensor data, image or video data, and log data. There are many compression techniques available - choose one that fits your device capabilities and the use case.

Compression can not only reduce network bandwidth requirements, but can also improve data storage efficiency by reducing the amount of storage space required for data. This can be particularly useful in scenarios where large amounts of data are generated, such as in video surveillance or industrial IoT applications.

Message formats should be chosen such that they reduce processing as well as storage requirements both locally and in the cloud. It is possible that a single message format or encoding scheme may not be the most optimal for your solution.

One commonly used format is protobuf (Protocol Buffers). Protobuf is a language and platform neutral, extensible mechanism for serializing structured data. It allows efficient binary encoding of device messages with low communication overhead and low CPU usage. Protobuf data structures

can change over time with backward compatibility, reducing operational overhead. AWS IoT Core supports the ingestion and translation of protobuf messages natively, making it straightforward for applications to use this feature.

The Concise Binary Object Representation (CBOR) is an IETF defined, self-describing data format designed for constrained devices. It allows for small code size and small message sizes. Each item is preceded by a tag that defines its type, allowing for extensibility without the need for version negotiation. CBOR is based on the JSON data model, which uses numbers, strings, arrays, maps (called objects in JSON), and values such as false, true, and null.

Between the two options, protobuf is the recommended choice for achieving speed and efficient use of resources, while CBOR is preferred for its flexibility and extensibility.

If using MQTT, it is advisable to use MQTT5 topic aliases for frequently used topics. This reduces topic lengths, resulting in reduced transmission time, lower power consumption, decreased communication costs, and improved latency.

Reduce the distance traveled by data

There are multiple techniques that can be applied to reduce the distance traveled by data between IoT devices and the cloud. These include moving processing to the edge, filtering data at the edge and using intermediate elements such as gateways to reduce the number of connections to the cloud as well as perform local caching functions. You should also make sure that your IoT solution uses regions that reduce the distance that network traffic must travel. If appropriate for your use case, select a region with close physical proximity to the deployed devices to minimize the distance data needs to travel across the network. The right choice of cloud services and architecture can reduce the network resources required to support your workload.

Optimize log verbosity

Use debug levels to increase/decrease metric and log verbosity based on the context. In normal operation, send the minimum set of metrics and logs required to identify system health. If runtime issues require additional verbosity, a more detailed log level could be set either dynamically by the device software, or by sending a new configuration to the devices that require it.

Buffer and spool messages

Message buffering and spooling can contribute to power consumption optimization in IoT devices in several ways.

Instead of immediately sending each message as it is generated, buffering allows for the aggregation of multiple messages into batches, if appropriate for the use case. By sending data in larger chunks or batches, the device can stay in a low-power state for longer periods, reducing the frequency of transitioning between active and sleep modes. This also allows the device to minimize network activity - instead of establishing a connection and sending data for every individual message, the device can consolidate multiple messages and initiate network communication less frequently. This reduction in network activity results in power savings, since establishing and maintaining network connections can be energy-intensive. It also reduces the amount of processing required, which lowers the power consumption.

In scenarios where the use of spooling is appropriate, it can be used to write data to disk in batches, rather than writing each data point as it is generated. This reduces the number of disk access operations required, which can help reduce power consumption.

Along with this, IoT devices can optimally time their data transmissions. Devices can assess factors like network availability, signal strength, or time-based considerations to select when to send accumulated messages. This approach helps to minimize energy consumption by avoiding unnecessary transmissions during periods of poor network conditions or low power availability (such as low battery charge levels).

Optimize the frequency of messages for your use case

To optimize the performance of a device, it is essential to adjust the input sampling time and make sure that it is sending messages and checking for updates at an optimal rate. The optimal rate should be determined by the use case and not necessarily by the rate at which inputs or sensor values change. Alternatively, an update on change (Change of Value) approach can be used, in which case an interpolation technique should be selected, and the device must be configured with a parameter that determines what a change is. Interpolation helps fill in the gaps in the time series data to provide a more complete and continuous representation of the data. By restricting the application related data that is transmitted to only what is required by the application, there are also benefits in terms of reduced data storage as well as the amount of processing required in the cloud, which contribute to a reduced carbon footprint.

Use gateways to offload and pre-process your data at the edge

The decision to connect a device directly to the cloud or via a gateway will depend on a variety of factors, including the specific requirements of the application and the characteristics of the sensor and network. It may be more power-efficient to use a gateway to receive and preprocess

data locally, reducing the need for higher power radios on the device, reducing long-haul communications traffic and extending battery life.

If the device generates a large amount of data, it may be more efficient to use a gateway to pre-process and filter the data before sending it to the cloud, reducing long-haul network traffic. In addition, if the long-haul network connection is unreliable, it may be more practical to use a gateway with AWS IoT Greengrass to buffer data and make sure that it is delivered reliably to the cloud.

Perform analytics at the edge

Data preparation is usually required before analytics can be performed at the edge. There are various types of preparation and transformation that can be done at the edge to improve the sustainability of the solution:

- Data filtering: IoT edge devices can filter out irrelevant data or noise from sensor data streams. For example, temperature sensors may monitor and report temperature at a high rate, but reflecting those changes as they come may not be useful to a specific application or use case. By filtering out the unnecessary data points, edge devices can reduce the amount of data that needs to be transmitted and processed, reducing communication related power and cloud processing costs.
- Data aggregation: Devices can aggregate sensor data from multiple sources and over time to reduce the amount of data sent to the cloud. This can involve combining data from multiple sensors to create a single data stream, or aggregating data from multiple devices to provide a system-level view of performance or behavior. This can reduce the number of connection requests as well as messages sent to the cloud.
- Data enrichment: Sensor data can be enriched with additional contextual information, such as location or time data. This can enable more accurate analysis and insights, as well as provide additional context for cloud applications or systems. This reduces the need to enrich data in the cloud. Local processing and data enrichment should be considered when the cost of doing the same operation in the cloud out-weighs the impact of larger payloads and additional local computation and resources.
- Data normalization: Devices can normalize sensor data from different devices or sources to support consistency and compatibility with cloud applications or systems. This can involve converting data formats, units of measurement, or other data attributes to a common standard, reducing the need to do this in the cloud.

Devices can perform real-time analytics, predictive maintenance, anomaly detection, optimization, and security monitoring by analyzing sensor data and triggering alerts or actions based on predefined rules or machine learning models, without sending the source data to the cloud.

To perform analytics in an efficient and sustainable manner on the edge, use lightweight algorithms to improve performance and resource efficiency on IoT devices. Examples include algorithms such as decision trees or regression models that are less computationally intensive than deep learning models. In addition, optimizing data structures can improve the performance and efficiency of analytics algorithms on IoT devices. Examples include data structures such as binary trees or hash tables that require less memory and processing power.

Monitor and manage your fleet operations to maximize sustainability

It is recommended that your solution be monitored and managed efficiently across its lifecycle to reduce the carbon footprint of your operations. Common operational tasks such as inspecting the connectivity state of a device, verifying device credentials are configured correctly, and querying devices based on their current state must be in place before launch so that your system has the required visibility to troubleshoot applications and reduce the need for site visits. From a sustainability point of view, inaccurate information about device status can result in delayed or erroneous actions such as unnecessary truck rolls or mis-directed corrective actions. This can lead to decreased efficiency and increased costs.

Having an accurate view of the state of all devices is therefore an imperative for operating at scale. This is best done through the use of tools, services and automation. Here are some of the key ways to monitor IoT operations using AWS IoT services:

- Use AWS IoT Device Management services to manage the entire lifecycle of your IoT devices, including firmware updates in the field. This helps in the remote management of devices, reducing the need for site visits and lowering the carbon footprint of your operations. For additional observability into your device's resource allocation, you can install the AWS X-Ray daemon to see how edge applications perform and where CPU utilization or other optimizations can be made to decrease power consumption or better allocate device resources.
- Monitor resource utilization periodically and set up an alerting system to notify appropriate stakeholders if the utilization goes above a specific threshold. For simple threshold conditions, detection can be done on the device itself and alerts sent to the cloud. During the development phase, cloud-based monitoring can be used to determine the appropriate threshold values for your workload. It is recommended to configure automatic actions such as optimizing resource allocation or disabling processes to avoid system downtime caused by anomalous conditions.

Regularly reviewing monitoring data and threshold adjustments will support accurate and timely notifications that reflect the current performance of the IoT device.

- AWS IoT Core provides a set of metrics that you can use to monitor the performance and health of your IoT devices and applications. These metrics include device connections, message delivery, and rule engine metrics. You can view these metrics using Amazon CloudWatch or the AWS IoT Core console. In addition, you can monitor device metrics such as CPU and memory utilization, battery levels and the like using the Fleet Metrics feature of AWS IoT Core. These metrics can be used to proactively identify devices that are degrading or will soon need attention, averting service impact and the need for emergency truck rolls.

Definitions

Sustainability encompasses seven focus areas:

- Region selection
- Alignment to demand
- Software and architecture - optimization
- Software and architecture - cloud
- Data management
- Hardware and services - optimization
- Hardware and services - power management
- Process and culture - user guidance

The following is a list of sustainability-specific definitions in the IoT Lens.

- **Energy efficiency or power efficiency**: Used interchangeably in this document.
- **Truck roll (or site visit):** The act of sending a technician to a site to resolve an issue. This has impacts both from an operational cost perspective (technician labor, fuel costs, maintenance running to hundreds of dollars per device) as well as carbon footprint (emissions, wear and tear). Consequently, designing IoT devices to minimize the need for truck rolls is a key imperative.
- **Battery life:** Refers to how long a device can run on a single charge.
- **Battery lifetime:** Refers to how long the battery can be in operation before needing replacement.

Region selection

There are no best practices specific to region selection in the IoT Lens. Refer to the design principles discussed earlier as well as the guidance provided in the Sustainability Pillar of the AWS Well-Architected Framework.

Alignment to demand

There are no best practices specific to alignment to demand in the IoT Lens. Refer to the guidance provided in the Sustainability Pillar of the AWS Well-Architected Framework.

Software and architecture optimization

IOTSUS01: How do you optimize software and firmware to reduce device's carbon footprint?

Reducing the carbon footprint of IoT devices requires a multi-faceted approach to software optimization throughout the system stack. At the core operating system and firmware level, best practices focus on creating lean, streamlined images that exclude unnecessary bloat. This involves using tools to build custom images containing only the minimum required modules and libraries to support the device's functionality. Techniques like event-driven architectures, efficient programming languages, hardware accelerators, and optimized logging further improve performance and reduce processing overhead. Over-the-air update capabilities allow extending product lifetimes through software updates rather than full device replacements.

Moving up the stack, application-layer best practices concentrate on intelligent edge processing and bandwidth reduction. This includes filtering, aggregating, and enriching data locally to minimize cloud transmission. Adopting efficient data formats, compression, adaptive power management, and message buffering and spooling policies can significantly reduce network traffic. Leveraging AWS IoT services can help with optimizing data ingestion and device management as well as help with implementing sustainability-focused features like carbon footprint monitoring.

IOTSUS01-BP01 Eliminate unnecessary modules, libraries, and processes

Verify that the operating system only runs essential processes that are necessary for the functionality of the IoT device.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

Unnecessary libraries, modules, and processes contribute to a larger device footprint, increase patching requirements, and create a larger attack surface and more processes for the CPU to run.

Choose efficient programming languages that satisfy your business requirements. Programming language choice has an impact on device requirements as well as active and sleep cycles. Programming languages vary in areas such as memory management, typing, and parallelism. It is recommended to design and test as much as possible prior to making a final decision on language.

Produce a more efficient and secure IoT device design by streamlining the operating system and only including essential processes.

Use projects like Yocto or Buildroot to build custom Linux images containing only the necessary modules for device functionality, and build device software like AWS IoT Greengrass or the AWS IoT Device Client into these custom images using layers like the meta-aws Yocto layer.

IOTSUS01-BP02 Use AWS IoT features to optimize network usage and power consumption

Select AWS IoT service features which can help to optimize network and power resources.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

Use AWS IoT Device Shadow services, which are virtual representations of IoT devices in the cloud. Device shadows enable decoupled bi-directional communication between the device and applications running in the cloud. Applications can obtain device state from the shadow rather than the device, reducing traffic between the device and cloud, and allowing the application to continue operation even if the device is disconnected intermittently. When a device comes back online, it can check if there were any changes requested by the application while it was offline, and take action as needed. This allows the device to stay offline, saving power.

Use MQTT retained messages, message expiry, and session expiry features. Retained messages and Device Shadows both retain data from a device but have different capabilities and suitability. MQTT5 message expiry can be used to make sure that devices only receive time-relevant messages, reducing processing load. The session expiry feature can be used by MQTT clients to

set application-specific session expiry limits, making sure that the broker does not need to retain resources beyond what is needed.

IOTSUS01-BP03 Use a hardware watchdog to restart your device automatically

IoT devices should have a hardware watchdog mechanism, which can reduce downtime by automatically restarting the device when it becomes unresponsive. In many cases, restarting the device can put it in a state where it can be remotely managed, minimizing the impact of failures and reducing the need for site visits.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

Choose processors that include a hardware watchdog and that are well supported by vendor software solutions.

IOTSUS01-BP04 Implement resilient and scalable system behavior for clients communicating with the cloud

Clients communicating with the cloud must not only be functionally correct, but also implement resilient and scalable system behavior. Implementing such behavior reduces the work done by each client device and reduces network traffic and doing so can improve device longevity and total lifetime energy consumption.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

Support exponential backoff with jitter when handling connection retries to cloud endpoints.

Minimize the number of connection attempts when dealing with a congested network, reducing the work done by each client and reducing network traffic.

Define a threshold at which point it is more effective to enter low power modes during a backoff period.

Support MQTTv5 reason codes and use that information to determine if and when to reconnect.

Software and architecture – Cloud

IOTSUS02: How do you incorporate optimized cloud services in your architecture to minimize your carbon footprint?

Studies by 451 Research have shown that AWS' infrastructure is 3.6 times more energy efficient than the median of U.S. enterprise data centers surveyed and up to 5 times more energy efficient than the average in Europe. 451 Research also found that AWS can lower customers' workload carbon footprints by nearly 80% compared to surveyed enterprise data centers, and up to 96% once AWS is powered with 100% renewable energy by 2025. While this makes it much easier to lower your carbon footprint for workloads in the cloud, it does not eliminate the need for optimization of cloud workloads.

Use the AWS customer carbon footprint tool to view estimates of the carbon emissions associated with their usage of AWS products and services. This tool provides data visualizations to help customers understand their historical carbon emissions, evaluate emission trends as their use of AWS evolves, approximate the estimated carbon emissions they have avoided by using AWS instead of an on-premises data center, and review forecasted emissions. The forecasted emissions are based on current usage and show how a customer's carbon footprint can change through its actions.

More detailed guidance on sustainability practices in the cloud can be found in documentation relating to those services as well as in the Sustainability Pillar of the AWS Well-Architected Framework.

IOTSUS02-BP01 Use the Basic Ingest feature in AWS IoT Core

With the Basic Ingest feature, you can securely send device data to the AWS services supported by AWS IoT rule actions, without incurring messaging costs. Basic Ingest optimizes data flow by removing the publish/subscribe message broker from the ingestion path, making it more cost-effective and resource-efficient.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

When ingesting data to AWS IoT Core, consider whether to use the Basic Ingest feature or not. Use this approach if your application does not require multiple subscribers for the data being ingested.

For ingestion mechanisms other than Basic Ingest (such as the Amazon Kinesis family of services), refer to the AWS IoT Lens for guidance on which service is appropriate for which use case. At this time, there are no additional sustainability best practices for these services.

IOTSUS02-BP02 Choose an appropriate Quality of Service (QoS) level

Higher QoS levels involve additional network overhead for acknowledgment and retransmission, which can increase power consumption.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

Use MQTT when you have IoT devices or other resource-constrained environments that need to communicate efficiently and reliably with a publish-subscribe messaging pattern. MQTT supports different quality of service (QoS) levels for message delivery. Consider using lower QoS levels (such as QoS 0) if the reliability of message delivery is not critical for your use case to reduce power consumption and network overhead.

Data management

There are no best practices specific to data management in the IoT Lens.

Hardware and services - Hardware optimization

IOTSUS03: How do you pick the right hardware components?

Selecting hardware components is a critical early decision that significantly impacts the overall sustainability of an IoT device throughout its lifecycle. Best practices prioritize right-sizing components by carefully evaluating the actual performance requirements of the intended workload. This includes choosing processors that provide the right balance of compute capabilities and energy efficiency by leveraging benchmarks and real-world test data. Incorporating accelerators and offload engines for specialized tasks like machine learning inference can further improve power utilization.

Beyond CPUs, sustainability-focused hardware choices span memory, storage, sensors, wireless connectivity, power sources, and security modules. Preferred options intelligently manage power states, support energy harvesting, utilize efficient video encoding, and incorporate secure elements to reduce computational overhead. Right-sizing capacities while allowing for future extensibility is key to prolonging product lifetimes. Design decisions should comprehensively evaluate embodied carbon across the manufacturing supply chain in addition to optimizing for the in-use operational phase.

IOTSUS03-BP01 Source sustainable components to help reduce environmental harm and encourage eco-friendly IoT products

Several factors can impact sustainability in various stages of the design process. These include choices related to materials, packaging, and product design, which can significantly influence the carbon footprint of the final product.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance:

Adopt sustainable practices in the design and manufacturing layer to reduce the environmental impact of IoT products. Consider factors that can impact sustainability during various stages of the design process, such as choices related to materials, packaging, and product design, which can significantly influence the carbon footprint of the final product.

Implement sustainable supply-chain practices, such as sourcing from suppliers that demonstrate environmentally responsible practices, using recycled or renewable materials, or selecting products with lower environmental impact throughout their lifecycle.

Use products that are certified as Climate Pledge Friendly, which meet sustainability standards for reducing carbon emissions and promoting a circular economy.

IOTSUS03-BP02 Consider the manufacturing and distribution footprint of your device

Choosing manufacturing facilities with low environmental impacts, optimizing transportation routes to minimize emissions, and utilizing energy-efficient manufacturing processes can all contribute to improved sustainability in the supply-chain.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

1. Choose manufacturing facilities with low environmental impacts, such as those that use energy-efficient processes or renewable energy sources.
2. Optimize transportation routes and modes to minimize emissions from shipping products.
3. Use energy-efficient manufacturing processes, such as using low-temperature solder during the pick-and-place operation to reduce energy consumption for heating solder on printed circuit boards (PCBs).
4. Design IoT devices and packaging in smaller form factors to allow for easier and more efficient shipping in large volumes while consuming fewer raw materials and harmful chemicals.
5. Consider the entire supply chain and make decisions that contribute to positive environmental outcomes through thoughtful product design, material selection, and manufacturing processes.

IOTSUS03-BP03 Use benchmarks to help you make a processor choice

Processor and IoT benchmarks can help you assess and narrow down which processor is appropriate for your use case.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

Choose benchmarks that include workloads that closely mimic the actual workloads IoT devices are expected to handle, such as sensor data processing, edge filtering, and running communication protocols.

Look for benchmarks that provide relevant performance metrics considering the resource constraints of IoT devices, such as low power operation and real-time processing requirements.

Consider benchmarks that include energy efficiency metrics, such as Performance per Watt and Thermal Design Power (TDP), to assess how efficiently CPUs can process workloads while minimizing energy consumption.

Use benchmarks that include test cases to evaluate the real-time processing capabilities of CPUs, including latency and responsiveness, which are important for IoT applications.

Select benchmarks that evaluate the communication and connectivity performance and efficiency of CPUs, as IoT devices require communication capabilities to interact with other devices, cloud services, or data centers.

Consider using benchmarks from the Embedded Microprocessor Benchmark Consortium (EEMBC), such as the EEMBC IoTMark and EEMBC ULPBench, which are specifically designed for evaluating the performance of CPUs in IoT applications and include relevant metrics aligned with sustainability evaluation criteria.

IOTSUS03-BP04 Optimize your device based on real-world testing

Make the final selection of your device hardware based on evaluating one or more hardware choices under close-to-actual operating conditions. Processors, peripherals, and other components must be chosen to optimize power draw during runtime as well as during device idle states. Other criteria as discussed throughout this document can be used to make a final selection based on the results of your testing.

Once the hardware has been finalized, examine whether the observed performance matches the expected performance. Profiling of your code on the target hardware under real workloads can help identify power-hungry sections of the code and help you optimize them for efficiency. Examining application and OS use of the device's power saving features and modes may also be required to achieve optimal efficiency.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

Evaluate one or more hardware choices under close-to-actual operating conditions to make the final selection of device hardware.

Choose processors, peripherals, and other components that optimize power draw during runtime as well as during device idle states.

Use criteria discussed in the sustainability best practices document, such as energy efficiency, real-time processing, and connectivity performance, to make the final hardware selection based on the results of your testing.

Once the hardware has been finalized, examine whether the observed performance matches the expected performance by profiling your code on the target hardware under real workloads.

Identify power-hungry sections of the code through profiling and optimize them for efficiency.

Examine the application and operating system's use of the device's power-saving features and modes, and make necessary adjustments to achieve optimal efficiency.

IOTSUS03-BP05 Use sensors with built-in event detection capabilities

Sensor components are the foundation of IoT, bridging the physical and digital worlds and providing real-time data on environmental conditions. Sensor components should be designed to operate with minimal power consumption by optimizing data transmission. Some sensor components have built-in data processing capabilities to generate events that are directly usable by the host device's application. For example, inertial measurement unit (IMU) sensors can detect fall events by processing acceleration, orientation, and motion data locally, and generating an interrupt to the host processor with an alert when a fall is detected, enabling the host processor to wake up and process the event while conserving power during regular operation.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

Optimize sensor components to have built-in data processing capabilities to generate events that are directly usable by the host device's application, reducing the need for further processing on the host.

Configure sensor sampling rates to balance between capturing enough data for accuracy and conserving power to reduce battery drain, based on the specific use case requirements.

Employ techniques such as adjusting the sampling rate based on sensor data variability, prioritizing critical data over less important data, or using event-triggered or adaptive sampling approaches to reduce unnecessary data collection.

Use sensors that can perform local processing of raw sensor data using embedded algorithms or machine learning models, and generate interrupts to the host processor only when specific events are detected. This allows the host processor to operate in a low-power mode until an interrupt is received.

IOTSUS03-BP06 Use hardware acceleration for video encoding and decoding

Hardware acceleration for video encoding and decoding is crucial for sustainability in IoT devices like security cameras and video doorbells. By offloading intensive video processing to dedicated hardware accelerators, it can reduce power consumption, allowing main processors to operate at lower clock speeds or enter low-power states more frequently. This improved energy efficiency not only decreases the overall energy footprint but also enables more compact and resource-

efficient IoT device designs, aligning with sustainable product principles by minimizing material and resource consumption.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

For IoT devices that require processing and streaming video to the cloud, such as doorbell or security cameras, use video encoding to reduce data transmission and file size.

Adopt the H.265 (HEVC or High Efficiency Video Coding) video encoding standard, which provides better video quality at lower bit rates compared to H.264 (AVC or Advanced Video Coding), resulting in reduced bandwidth requirements, lower power consumption, and lower communication costs during video playback or transmission.

Choose a microcontroller or microprocessor with dedicated video encoding hardware acceleration to improve performance and reduce power consumption in video processing tasks.

Consider system designs that offer a dedicated video encoding co-processor that runs a single video encoding algorithm, allowing the host processor to handle other general-purpose tasks.

With advancements in technology, more efficient video encoding algorithms may be developed in the future. To extend the device's lifespan, choose a hardware accelerator with an FPGA or other updatable logic that can be updated to support more efficient encoding algorithms as they become available.

IOTSUS03-BP07 Use HSMs to accelerate cryptographic operations and save power

Incorporate secure hardware and hardware security modules (HSMs) in IoT device designs to improve security, reduce energy consumption, and enhance sustainability. For example, an HSM typically performs Elliptic Curve Digital Signature Algorithm (ECDSA) signature operations several times faster than software on a general-purpose microcontroller, allowing the host microcontroller to spend more time in a low-power mode while the HSM performs complex cryptographic operations.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

Use secure hardware components such as Trusted Platform Modules (TPMs), hardware security modules (HSMs), secure elements (SEs), and secure enclaves (SEs) like Arm TrustZone in IoT device designs to significantly speed up cryptographic operations, reduce energy consumption, and enhance security.

If the device supports cellular connectivity, use the SIM card as a secure element instead of a dedicated one to reduce the overall bill of materials (BOM).

Use device certificates with long expiration dates designed to be rotated only as needed to maintain the security posture of the device, as rotating device certificates can be computationally expensive and requires additional communication with the cloud.

IOTSUS03-BP08 Use low-power location tracking

Employ low-power tracking solutions for IoT for sustainability and resource efficiency. These solutions extend battery life, minimizing the need for frequent replacements and associated electronic waste. They reduce overall energy consumption and carbon footprints, while enabling reliable operation in remote or off-grid locations without continuous power sources.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

For GPS-based devices, use chipsets that support assisted-GPS (A-GPS), which reduces power consumption by offloading some of the location calculation work to the network.

Consider using location services like AWS IoT Core Device Location, which leverages cloud-based location solvers such as Wi-Fi scan, cellular scan, Global Navigation Satellite System (GNSS) scan, or reverse IP look-up to determine the geo-coordinates of IoT devices. Using cloud-based location services can reduce the device power consumption required to resolve location, as the computationally expensive location calculations are offloaded to the cloud.

Hardware and services - Power management

IOTSUS04: How do you minimize power usage and wastage?

Minimizing power usage and waste for IoT devices is crucial for both environmental sustainability and operational cost reduction. From an environmental perspective, reducing the power

consumption of IoT devices directly translates to a lower carbon footprint. As IoT deployments can consist of millions of devices, any improvements in power efficiency can have a significant cumulative impact on energy conservation and climate change mitigation efforts. Optimizing power usage can also extend the operational lifetime of IoT devices, reducing the need for frequent replacements and minimizing electronic waste. Implementing best practices in energy-efficient hardware selection, software optimization, and intelligent power management strategies can yield benefits in terms of both environmental impact and operational efficiency.

IOTSUS04-BP01 Use energy harvesting technologies to power your device

One approach to improve sustainability is to use energy harvesting technologies to provide some or all of the power needs of a device, reducing reliance on grid-based power sources.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

Incorporate energy harvesting technologies that can capture renewable energy sources such as solar energy, thermal energy, vibration and mechanical energy, radio frequency energy, wind energy, and piezoelectric energy to power IoT devices. Use batteries or supercapacitors to store the captured energy, providing continuous availability of power for the devices.

IOTSUS04-BP02 Implement tickless operation and low-power modes

Implementing *tickless* operation and making full use of low power modes available reduces overall power consumption. Reducing power consumption can, amongst other things, have impacts to how long a device can be deployed and the size battery needed to satisfy the business use case. Doing so improves the overall sustainability of the device.

Level of risk exposed if this best practice is not established: Medium

Prescriptive guidance

1. Use the tickless operation technique in embedded operating systems like FreeRTOS to reduce the frequency of system interrupts or *ticks* while the system is idle, minimizing power consumption. Use the idle hook function in the embedded operating system to place the microcontroller CPU in a low-power mode when the system is idle.

2. For power-critical applications, consider factors such as the latency and power requirements of entering and exiting low-power modes, and choose the low-power mode that provides the best trade-off between power savings and responsiveness.
3. Configure the appropriate wake-up sources or events that will alert the system to exit the low-power mode, further minimizing power consumption by avoiding unnecessary wake-ups.
4. Implement low-power modes for all project areas. For example, it is important to implement low-power modes for the communication stack in a device as well as the sensor portion of the application.

IOTSUS04-BP03 Allow applications or software running on devices to dynamically adjust settings based on requirements and available resources

Implementing dynamic adjustment of hardware settings and power management techniques on devices is important for sustainability. It enables energy efficiency, extends device lifespan, and optimizes resource utilization. By allowing applications to adapt hardware settings based on requirements and available resources, and leveraging dynamic power management techniques, organizations can develop energy-efficient and long-lasting devices that minimize environmental impact.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

1. Enable applications or software on edge devices to make decisions about changing hardware states, such as CPU frequency, voltage, or other hardware settings, based on the specific requirements of the application and the available resources.
2. Implement dynamic power management techniques, where the device adjusts its power consumption in real-time based on the available energy.
3. Use low-power libraries and APIs provided by microcontrollers and processors used in IoT devices, as these offer optimized functions for power management and can help in the realization of dynamic power management.

Process and culture - User guidance

IOTSUS05: How do you educate users to encourage lower carbon footprint of their devices?

Clear documentation and intuitive interfaces are vital for educating users on proper installation, configuration, and maintenance procedures that can maximize energy efficiency and device longevity. Providing step-by-step guidance, power usage monitoring tools, and over-the-air update capabilities can help users avoid wasteful practices that lead to increased energy consumption or premature device replacements.

Organizations can further incentivize sustainable user behavior by offering repair services, trade-in programs, and responsible disposal options. Making it easier and more cost-effective for users to repair, upgrade, or properly recycle devices at end-of-life reduces the likelihood of devices ending up in landfills prematurely.

A holistic approach embeds sustainability into the product lifecycle, from design and manufacturing to user education and end-of-life management. Actively involving users in this process and providing the necessary tools and incentives reduces the carbon footprint of IoT products and services.

IOTSUS05-BP01 Create detailed documentation

Provide user-friendly documentation or mobile applications that give a user detailed step-by-step guidance to educate users on the proper installation and use of devices to avoid errors or misuse that could necessitate a site visit from a technician, leading to additional cost and environmental impact

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

Educate users on the proper installation and use of IoT devices through user-friendly documentation or mobile applications that provide detailed step-by-step guidance.

IOTSUS05-BP02 Promote responsible disposal, repairability, and transfer of ownership for IoT devices to minimize environmental impact

Implementing responsible disposal, repairability, and transfer of ownership practices for devices is crucial for sustainability. It minimizes environmental impact by promoting recycling, proper disposal of hazardous components, and adherence to electronic waste regulations.

Level of risk exposed if this best practice is not established: Low

Prescriptive guidance

Implement environmentally responsible practices for disposing of IoT devices, including recycling electronic components, properly disposing of batteries, and adhering to local regulations and guidelines for electronic waste disposal.

Collaborate among IoT device manufacturers, users, and stakeholders to develop, implement, and improve sustainable and responsible disposal practices to minimize environmental impact.

Promote repairability and support repair and transfer of ownership options for IoT devices to extend their lifespan and reduce waste.

Verify that it is straightforward for users to update, upgrade, and repair devices back to a working state.

Provide users with detailed instructions on how to perform a factory reset, wipe data, and dissociate devices from the current user account before transferring ownership or disposing of the device.

Provide clear instructions to users on how to properly dispose of the device at the end of its life, including guidance on recycling components and isolating any harmful materials.

Consider creating incentives for users to follow proper disposal and recycling practices for IoT devices.

IOTSUS05-BP03 Identify when devices in the field can or should be retired

As circumstances change in your deployed solution (sites shut down, for instance) devices may remain active even though not needed. To minimize the impact of such cases, unused assets should be decommissioned.

Level of risk exposed if this best practice is not established: Low

Key AWS services

- **Ingest IoT data at scale**: Use AWS IoT Core - a managed service - to manage device connectivity, device security to the cloud, message ingestion, message routing, and device state. Use Amazon Kinesis Video Streams, a fully managed AWS service with device side producer SDKs, to stream live video from devices to the AWS Cloud, or build applications for real-time video processing or batch-oriented video analytics. Amazon Kinesis Data Streams can be used to collect and process large streams of data records in real time.
- **Remote device operations:** Use AWS IoT Jobs to instruct devices to download and install applications, run firmware updates, reboot, rotate certificates, or perform remote troubleshooting operations.
- **Seamless remote access:** AWS IoT Secure Tunneling allows you to establish bidirectional communication to remote devices over a secure connection that is managed by AWS IoT, without requiring changes to your existing inbound firewall rules.
- **Monitor and manage devices**: AWS Systems Manager can act as the operations hub for your AWS applications and resources and a secure end-to-end management solution for your devices as well as hybrid and multicloud environments, enabling secure operations at scale.
- **Monitor and mitigate device risks:** AWS IoT Device Defender is a security service that allows you to audit the configuration of your devices, monitor connected devices to detect abnormal behavior, and mitigate security risks. It gives you the ability to enforce consistent security policies across your AWS IoT device fleet and respond quickly when devices are compromised.
- **Develop edge applications:** AWS IoT Greengrass is an open source IoT edge runtime and cloud service that helps you build, deploy and manage IoT applications on your edge devices.

Resources

Refer to the following resources to learn more about our best practices related to sustainability.

- Amazon Sustainability - The Cloud
- Optimizing your AWS Infrastructure for Sustainability
- AWS enables sustainability solutions
- Amazon Sustainability - Driving Climate Solutions
- Greenhouse Gas Protocol
- Deploying and managing an IoT workload on AWS
- How to build smart applications using Protocol Buffers with AWS IoT Core
- Enabling device maintenance across multiple time zones using AWS IoT Jobs
- Connect to remote devices using AWS IoT Secure Tunneling
- How to remote access devices from a web browser using secure tunneling
- Build resilient IoT device applications that remain active using the AWS IoT Device SDKs
- Enhancing IoT device security using Hardware Security Modules and AWS IoT Device SDK
- Enable compliance and mitigate IoT risks with automated incident response
- Introducing new MQTTv5 features for AWS IoT Core to help build flexible architecture patterns
- Schedule remote operations using AWS IoT Device Management Jobs
- Design considerations for cost-effective video surveillance platforms with AWS IoT for Smart Homes
- Build machine learning at the edge applications using Amazon SageMaker AI Edge Manager and AWS IoT Greengrass V2
- Best practices for ingesting data from devices using AWS IoT Core and Amazon Kinesis
- Automate application deployment to IoT devices using AWS IoT Device Management
- Optimize image classification on AWS IoT Greengrass using ONNX Runtime
- Reaching Net-Zero Carbon by 2040: Decarbonizing and Neutralizing the Use Phase of Connected Devices

Conclusion

The AWS Well-Architected Framework provides architectural best practices across the pillars for designing and operating reliable, secure, efficient, and cost-effective systems for IoT applications. The framework provides a set of questions that you can use to review an existing or proposed IoT architecture, and also a set of AWS best practices for each pillar. Using the framework in your architecture helps you produce stable and efficient systems, which allows you to focus on your functional requirements.

Appendix: Best practices by pillar

Operational excellence

IOTOPS01: How do you evaluate governance and compliance requirements?

- IOTOPS01-BP01 Conduct an OT and IT cybersecurity risk assessment using a common framework
- IOTOPS01-BP02 Evaluate if OT and IT teams use separate policies and controls to manage cybersecurity risks or if they use the same policy

IOTOPS02: Is there a central cloud center of excellence (CCoE) with equivalent representation from OT and IT in industrial organizations?

- IOTOPS02-BP01 Consolidate resources into centers of excellence to bring focus to new or transforming enterprises

IOTOPS03: Do you organize the fleet to quickly identify devices?

- IOTOPS03-BP01 Use static and dynamic device hierarchies to support fleet operations
- IOTOPS03-BP02 Use index and search services to enable rapid identification of target devices

IOTOPS04: How do you verify that newly provisioned devices have the required operational prerequisites?

- IOTOPS04-BP01 The device management processes should be automated, data-driven, and based on previous, current, and expected device behavior

IOTOPS05: How do you govern device fleet provisioning process?

- IOTOPS05-BP01 Document how devices join your fleet from manufacturing to provisioning
- IOTOPS05-BP02 Use programmatic techniques to provision devices at scale
- IOTOPS05-BP03 Use device level features to enable re-provisioning

IOTOPS06: How do you implement observability for your IoT system?

- IOTOPS06-BP01 Implement monitoring to capture logs and metrics
- IOTOPS06-BP02 Capture and monitor application performance at the edge
- IOTOPS06-BP03 Monitor the status of your IoT devices
- IOTOPS06-BP04 Use device state management services to detect status and connectivity patterns

IOTOPS07: How do you assess whether your IoT application meets your operational goals?

- IOTOPS07-BP01 Enable appropriate responses to events
- IOTOPS07-BP02 Use data-driven auditing metrics to detect if any of your IoT devices might have been broadly accessed

IOTOPS08: How do you segment your device operations in your IoT application?

- IOTOPS08-BP01 Use static and dynamic device attributes to identify devices with anomalous behavior

IOTOPS09: How do you evolve your IoT application with minimum impact to downstream IoT devices?

- IOTOPS09-BP01 Run ops metrics analysis across business teams, document learnings and define action items for future firmware deployments

IOTOPS10: How do you verify that you are ready to support the operations of devices in your IoT workload?

- IOTOPS10-BP01 Train team members supporting your IoT workloads on the lifecycle of IoT applications and your business objectives

Security

IOTSEC01: How do you associate IoT identities and permissions with your devices?

- IOTSEC01-BP01 Assign unique identities to each IoT device

IOTSEC02: How do you secure your devices and protect device credentials?

- IOTSEC02-BP01 Use a separate hardware or a secure area on your devices to store credentials
- IOTSEC02-BP02 Use a trusted platform module (TPM) to implement cryptographic controls
- IOTSEC02-BP03 Use protected boot and persistent storage encryption

IOTSEC03: How do you authenticate and authorize user access to your IoT application?

- IOTSEC03-BP01 Implement authentication and authorization for users accessing IoT resources
- IOTSEC03-BP02 Decouple access to your IoT infrastructure from the IoT applications

IOTSEC04: How do you apply least privilege to principals that interact with your IoT application?

- IOTSEC04-BP01 Assign least privilege access to devices

IOTSEC05: How do you manage device certificates, including installation, validation, revocation, and rotation?

- IOTSEC05-BP01 Perform certificate lifecycle management

IOTSEC06: How do you analyze application and device logs and metrics to detect security issues?

- IOTSEC06-BP01 Collect and analyze logs and metrics to capture authorization errors and failures to enable appropriate response
- IOTSEC06-BP02 Send alerts when security events, misconfiguration, and behavior violations are detected
- IOTSEC06-BP03 Alert on non-compliant device configurations and remediate using automation

IOTSEC07: What infrastructure protection configuration has been defined for your AWS organization and accounts?

- IOTSEC07-BP01 Configure cloud infrastructure to have secure communications
- IOTSEC07-BP02 Define networking configuration which restricts communications to only those ports and protocols which are required

- IOTSEC07-BP03 Log and monitor network configuration changes and network communication

IOTSEC08: How is the infrastructure into which your IoT devices are deployed managed and maintained?

- IOTSEC08-BP01 Define an automated and monitored mechanism for deploying, managing, and maintaining networks to which IoT devices are connected
- IOTSEC08-BP02 Define an automated and monitored mechanism for deploying, managing, and maintaining network configurations for IoT devices

IOTSEC09: What processes are used to manage and maintain the hardware or software deployed and configured in your IoT devices?

- IOTSEC09-BP01 Manage and maintain IoT Device software using an automated, monitored, and audited mechanism
- IOTSEC09-BP02 Manage IoT device configuration using automated and controlled mechanisms

IOTSEC10: How do you make sure that device data is protected at rest and in transit?

- IOTSEC10-BP01 Use encryption to protect IoT data in transit and at rest
- IOTSEC10-BP02 Use data classification strategies to categorize data access based on levels of sensitivity
- IOTSEC10-BP03 Protect your IoT data in compliance with regulatory requirements

IOTSEC11: How do you plan the security lifecycle of your IoT devices?

- IOTSEC11-BP01 Build incident response mechanisms to address security events at scale
- IOTSEC11-BP02 Require timely vulnerability notifications and software updates from your providers

IOTSEC12: How do you develop, maintain, manage, and deploy application code to your IoT devices and gateways?

- IOTSEC12-BP01 Manage IoT device and gateway source code using source code management tools

- IOTSEC12-BP02 Use static code analysis tools and code scanning to check IoT application code
- IOTSEC12-BP03 Deploy IoT applications using IaC, CI/CD pipelines, and build and deploy automation

IOTSEC13: How do you identify and remediate risks in IoT device firmware, IoT application code, and depended-upon packages or libraries?

- IOTSEC13-BP01 Use code and package scanning tools during development to identify potential risks during development
- IOTSEC13-BP02 Deploy updates to IoT device firmware or software to address identified issues
- IOTSEC13-BP03 Identify IoT devices which require updates and schedule updates to those devices

IOTSEC14: How do you govern the security of your IoT applications?

- IOTSEC14-BP01 Establish a security governance team for your IoT applications or extend the security governance team for the organization
- IOTSEC14-BP02 Define security policy so that it can be written into verifiable checks using policy as code techniques
- IOTSEC14-BP03 Implement a risk assessment and risk management process

IOTSEC15: What regulations apply to your IoT applications and how do you show compliance with these regulations?

- IOTSEC15-BP01 Identify the set of relevant regulations for your IoT applications
- IOTSEC15-BP02 Set up logging and monitoring to support audit checks for compliance
- IOTSEC15-BP03 Implement automated compliance checking using compliance as code

Reliability

IOTREL01: How do you make sure that your device consistently keeps its internal clock accurate?

- IOTREL01-BP01 Use NTP to maintain time synchronization on devices
- IOTREL01-BP02 Provide devices access to NTP servers

IOTREL02: How do you manage service quotas and limits for peaks in your IoT workload?

- IOTREL02-BP01 Manage service quotas and constraints

IOTREL03: How do you design workloads to operate efficiently within network bandwidth and storage constraints?

- IOTREL03-BP01 Down sample data to reduce storage requirements and network utilization

IOTREL04: How do you optimize and control message delivery frequency to IoT devices?

- IOTREL04-BP01 Target messages to relevant devices
- IOTREL04-BP02 Implement retry and back off logic to support throttling by device type

IOTREL05: How do you manage data ingestion and processing throughput for IoT workloads to other applications?

- IOTREL05-BP01 Decouple IoT applications from the Connectivity Layer through an Ingestion Layer

IOTREL06: How do you facilitate reliable processing and delivery of IoT messages across your workload?

- IOTREL06-BP01 Dynamically scale cloud resources based on the utilization

IOTREL07: How do you provision storage strategies for IoT data in the cloud?

- IOTREL07-BP01 Store data before processing
- IOTREL07-BP02 Implement storage redundancy and failover mechanisms for IoT data perpersistence

IOTREL08: How do you update device firmware on your IoT device?

- IOTREL08-BP01 Use a mechanism to deploy and monitor firmware updates
- IOTREL08-BP02 Configure firmware rollback capabilities in devices
- IOTREL08-BP03 Implement support for incremental updates to target device groups

- IOTREL08-BP04 Implement dynamic configuration management for devices

IOTREL09: How do you perform functional testing for your IoT solution?

- IOTREL09-BP01 Implement device simulation to synthesize the entire flow of IoT data

IOTREL10: How do you implement your IoT workload to withstand component and system faults?

- IOTREL10-BP01 Use cloud service capabilities to handle component failures

IOTREL11: How do you verify that your IoT device operates with intermittent connectivity to the cloud?

- IOTREL11-BP01 Implement device logic to automatically reconnect to the cloud
- IOTREL11-BP02 Design devices to use multiple methods of communication
- IOTREL11-BP03 Automate alerting for devices that are unable to reconnect

IOTREL12: How do you verify that required data is transmitted to the cloud after a device has been disconnected?

- IOTREL12-BP01 Provide adequate device storage for offline operations
- IOTREL12-BP02 Synchronize device states upon connection to the cloud

IOTREL13: How do you remotely adjust message frequency to your IoT devices?

- IOTREL13-BP01 Configure cloud services to reliably handle message processing
- IOTREL13-BP02 Send logs directly to the cloud
- IOTREL13-BP03 Design devices to allow for remote configuration of message publication frequency

IOTREL14: How do you plan for disaster recovery in your IoT workloads?

- IOTREL14-BP01 Design server software to initiate communication only with devices that are online

- IOTREL14-BP02 Implement multi-Region support for IoT applications and devices
- IOTREL14-BP03 Use edge devices to store and analyze data

Performance efficiency

IOTPERF01: How do your architectural decisions adapt to device hardware resources?

- IOTPERF01-BP01 Optimize for device hardware resources utilization

IOTPERF02: How do you measure and maintain the performance of your IoT solution?

- IOTPERF02-BP01 Implement comprehensive monitoring solutions to collect performance data from your IoT devices
- IOTPERF02-BP02 Evaluate the runtime performance of your application

IOTPERF03: Does transmitted content include auditable metadata?

- IOTPERF03-BP01 Add timestamps to each published message

IOTPERF04: Is there a mechanism for payload filtering or stream prioritization?

- IOTPERF04-BP01 Have mechanisms to prioritize specific payload types

IOTPERF05: How do you optimize telemetry data ingestion?

- IOTPERF05-BP01 Identify the ingestion mechanisms that best fit your use case
- IOTPERF05-BP02 Optimize data sent from devices to backend services

IOTPERF06: How do you efficiently make sure stored data is usable by business?

- IOTPERF06-BP01 Store data in different tiers following formats, access patterns and methods

IOTPERF07: How do you provide optimal connectivity for edge devices communicating to cloud infrastructure?

- IOTPERF07-BP01 Optimize network topology for distributed devices

- IOTPERF07-BP02 Perform timely connectivity verification for devices

IOTPERF08: How do you make sure application operates within its scaling limits?

- IOTPERF08-BP01 Load test your IoT applications
- IOTPERF08-BP02 Monitor and manage your IoT service quotas using available tools and metrics

IOTPOTPERF09: How do you maintain visibility over the distributed infrastructure deployed?

- IOTPERF09-BP01 Have device inventory in the IoT system that centralizes device configuration and diagnostics

Cost optimization

IOTCOST01: How do you choose cost-efficient tools for data aggregation of your IoT workloads?

- IOTCOST01-BP01 Use a data lake for raw telemetry data
- IOTCOST01-BP02 Provide a self-service interface for end users to search, extract, manage, and update IoT data
- IOTCOST01-BP03 Track and manage the utilization of data sources
- IOTCOST01-BP04 Aggregate data at the edge where possible

IOTCOST02: How do you optimize cost of raw telemetry data?

- IOTCOST02-BP01 Use lifecycle policies to archive your data
- IOTCOST02-BP02 Evaluate storage characteristics for your use case and align with the right services
- IOTCOST02-BP03 Store raw archival data on cost effective services

IOTCOST03: How do you optimize cost of interactions between devices and your IoT cloud solution?

- IOTCOST03-BP01 Select services to optimize cost
- IOTCOST03-BP02 Implement and configure telemetry to reduce data transfer costs

- IOTCOST03-BP03 Use shadow only for slow changing data
- IOTCOST03-BP04 Group and tag IoT devices and messages for cost allocation
- IOTCOST03-BP05 Implement and configure device messaging to reduce data transfer costs

IOTCOST04: How do you optimize cost by matching the supply of resources with device demand?

- IOTCOST04-BP01 Plan expected usage over time

IOTCOST05: How do you optimize payload size between devices and your IoT system to save cost?

- IOTCOST05-BP01 Balance networking throughput against payload size to optimize efficiency

IOTCOST06: How do you optimize the costs of storing the current state of your IoT device?

- IOTCOST06-BP01 Optimize shadow operations

Sustainability

IOTSUS01: How do you optimize software and firmware to reduce device's carbon footprint?

- IOTSUS01-BP01 Eliminate unnecessary modules, libraries, and processes
- IOTSUS01-BP02 Use AWS IoT features to optimize network usage and power consumption
- IOTSUS01-BP03 Use a hardware watchdog to restart your device automatically
- IOTSUS01-BP04 Implement resilient and scalable system behavior for clients communicating with the cloud

IOTSUS02: How do you incorporate optimized cloud services in your architecture to minimize your carbon footprint?

- IOTSUS02-BP01 Use the Basic Ingest feature in AWS IoT Core
- IOTSUS02-BP02 Choose an appropriate Quality of Service(QoS) level

IOTSUS03: How do you pick the right hardware components?

- IOTSUS03-BP01 Source sustainable components to help reduce environmental harm and encourage eco-friendly IoT products
- IOTSUS03-BP02 Consider the manufacturing and distribution footprint of your device
- IOTSUS03-BP03 Use benchmarks to help you make a processor choice
- IOTSUS03-BP04 Optimize your device based on real-world testing
- IOTSUS03-BP05 Use sensors with built-in event detection capabilities
- IOTSUS03-BP06 Use hardware acceleration for video encoding and decoding
- IOTSUS03-BP07 Use HSMs to accelerate cryptographic operations and save power
- IOTSUS03-BP08 Use low-power location tracking

IOTSUS04: How do you minimize power usage and wastage?

- IOTSUS04-BP01 Use energy harvesting technologies to power your device
- IOTSUS04-BP02 Implement tickless operation and low-power modes
- IOTSUS04-BP03 Allow applications or software running on devices to dynamically adjust settings based on requirements and available resources

IOTSUS05: How do you educate users to encourage lower carbon footprint of their devices?

- IOTSUS05-BP01 Create detailed documentation
- IOTSUS05-BP02 Promote responsible disposal, repairability, and transfer of ownership for IoT devices to minimize environmental impact
- IOTSUS05-BP03 Identify when devices in the field can or should be retired

Contributors

The following individuals and organizations contributed to this document:

- Ryan Dsouza, Principal Solutions Architect, Amazon Web Services
- David Malone, WW IoT Tech Leader, Amazon Web Services
- Andre Sa, Senior Solutions Architect, Amazon Web Services
- Chris Simpson, Senior Solutions Architect, Amazon Web Services
- Jordan Alexander, Partner Solutions Architect, Amazon Web Services
- Madhavan Menon, Senior Solutions Architect, Amazon Web Services
- Sunitha Eswaraiah, Senior Solutions Architect, Amazon Web Services
- Tim Hahn, Senior Security Consultant, Amazon Web Services
- Mahmoud Matouk, Principal Security Lead SA, Amazon Web Services
- Steven Arita, Cloud Optimization Success SA, Amazon Web Services
- Jun-Tin Yeh, Cloud Optimization Success SA, Amazon Web Services
- Derek Villavicencio, Cloud Optimization Success SA, Amazon Web Services
- Omkar Mukadam, TAM, Amazon Web Services
- Dinesh Ambu, Solutions Architect, Amazon Web Services
- Bruce Ross, Well-Architected Lens Leader, Amazon Web Services
- Madhuri Srinivasan, Sr. Technical Writer, Amazon Web Services
- Stewart Matzek, Sr. Technical Writer,, Amazon Web Services
- Matthew Wygant, Sr. TPM Guidance, Amazon Web Services

Document revisions

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

Change	Description	Date
Major update	Best practices have been added across the entire whitepaper. Many sections have been updated with new prescriptive guidance.	July 2, 2025
Major update	Updated to include Lens Checklist, industrial IoT (IIoT), and new AWS IoT services and features.	March 31, 2023
Minor update	Updated link.	March 10, 2021
Whitepaper updated	Updated to include additiona l guidance on IoT SDK usage, bootstrapping, device lifecycle management, and IoT	December 23, 2019
Initial publication	IoT Lens first published.	November 1, 2018

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.

AWS Glossary

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