IoT Lens Checklist

AWS Well-Architected Framework

IoT Lens Checklist: AWS Well-Architected Framework

Table of Contents

Overview

This paper describes the AWS Well-Architected Internet of Things (IoT) Lens, a collection of customer-proven best practices for designing well-architected IoT workloads. The IoT Lens contains insights that AWS has gathered from real-world case studies, and you will quickly learn the key design elements of well-architected IoT workloads along with recommendations for improvement. The intended readers of the document are IT architects, developers, and team members who build and operate IoT systems.

How to use the lens checklist

Use this Lens when you want to review the IoT workload. This Lens, however, does not replace the AWS Well-Architected Framework. Instead, it is a supplemental content that clarifies how to interpret and adopt the generic best practices of the Framework into the IoT workload designs. Therefore, we still recommend you to read the Well-Architected Framework whitepaper for more general yet comprehensive guidance for your new architecture designs, or for architecture reviews of existing IoT workload on premises or on AWS.

To conduct a workload review, first of all, read this document. Second, check whether your workload design follows each of the best practices, and lastly, record which best practices your workload should follow for further improvement. After the review, you will have a list of best practices that shows the workload is well-architected, or that it needs improvement. For the well-architected architectural components, share your knowledge among your teams to amplify them. For the best practices that your workload does not follow yet, treat them as technical debt and risks to your business. By adhering to your team's risk management process, you need to acknowledge them as risks, assess the likelihood and size of impact, examine solutions, implement solutions, and monitor the results. Since this Lens does not only provide the best practices but also recommends corresponding solutions, you can use them while exploring risk mitigation solutions.

Workload context checklist

To better understand your business's context, you need to gather the following information.

	ID	Priority	Workload Context
☐	C1	Required	Name of the workload
☐	C2	Required	Description that contains the business purposes, key performance indicators (KPIs), and the intended users of the workload.
☐	C3	Required	Review owner who leads the lens review
☐	C4	Required	Workload owner who is responsible for maintaining the workload.
☐	C5	Required	Business stakeholders who sponsor the workload
☐	C6	Required	Business partners who have a stake in the workload, such as information security, finance, and legal
☐	C7	Recommended	Architecture design document that describes the workload
☐	C8	Recommended	AWS Account IDs associated with the workload
☐	C9	Recommended	Regulatory compliance requirements relevant to the workload (if any)

Well-Architected pillar checklist

This section describes the design principles, best practices, and improvement recommendations that are relevant when designing your workload architecture. For brevity, only questions that are specific to IoT workloads are included in the IoT Lens. Thus, we recommend you also read and apply the guidance found in each Well-Architected pillar, which includes foundational best practices for operational excellence, security, performance efficiency, reliability, and cost optimization that are relevant to all workloads.

Security checklist

The security pillar encompasses the ability to protect data, systems, and assets to take advantage of cloud technologies to improve your security.

1 - Control the identity and access of the IoT devices

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 ensures devices have identity-based permissions for accessing the IoT platform, you establish the greatest control for managing device interactions.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 1.1	Required	Assign unique identities to each IoT device
☐	BP 1.2	Highly Recommended	Assign least privilege access to devices

- Identity and access management for AWS IoT
- AWS IoT Core Developer Guide: Customer authentication
- AWS IoT identity-based policy examples
- AWS IoT Core: X.509 client certificates
- AWS IoT Core Developer Guide: Just-in-time provisioning
- The Internet of Things on AWS – Official Blog: Just-in-Time Registration of Device Certificates on AWS IoT
- AWS IoT Core Developer Guide: Provisioning devices that don't have device certificates using fleet provisioning
- AWS Security Blog: How to Use Your Own Identity and Access Management Systems to Control Access to AWS IoT Resources

Best Practice 1.1 – Assign unique identities to each IoT device

When a device connects to other devices or cloud services, it must establish trust by authenticating using principals such as X.509 certificates, security tokens, or other credentials. 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 it should ensure 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. When multiple devices use the same certificate, this might indicate that a device has been compromised. Its identity might have been cloned to further compromise the system.

Recommendation 1.1.1 – Use X.509 client certificates to authenticate over TLS 1.2

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

Recommendation 1.1.2 - 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 in bulk when they first connect to AWS IoT.

Recommendation 1.1.3 - Use security bearer tokens for authorizing to the IoT broker

If the devices cannot use X509 certificates, or you have an existing fleet of devices with a proprietary access control mechanism, that requires use of bearer tokens such as OAuth over JWT or SAML tokens, use custom auth mechanisms. For example, when a device attempts to connect to AWS IoT, it sends a JWT generated by their identity provider in the HTTP header or query string. The signature is validated by AWS IoT custom authorizer and the connection is established.

Recommendation 1.1.4 - Use a consistent naming convention that maps your device identity to the MQTT topics

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

Best Practice 1.2 – Assign least privilege access to devices

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

See the following for more information:

- AWS IoT Core Developer Guide: Basic AWS IoT Core policy variables
- AWS IoT Core Developer Guide: Example AWS IoT policies

Recommendation 1.2.1 - Grant least privileged access to reduce the scope of impact of the potential events

We recommend using granular device permissions to enable least privileged access, which can help limit the impact of an error or misconfiguration. Define a mechanism so that devices can only communicate with specific authorized resources, such as MQTT topics. If permissions are generated dynamically, ensure 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 on which the action is allowed or denied.

Recommendation 1.2.2. – Consider scaling granular permissions across the IoT fleet

We recommend reusing permissions across principals rather than hardcoding for better manageability as it helps you avoid create redundant permissions per device. For example, an AWS IoT policy allows access based on various thing attributes such as `ThingName`, `ThingTypeName`, `Thing Attributes`, etc. Thus, a device can connect with a client ID (such as foo), only if the device registry contains the matched device (aka `ThingName`), such as `arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}` rather than, `arn:aws:iot:us-east-1:123456789012:client/foo`.

As another example, an AWS IoT policy also allows access based on various certificate attributes such as Subject, Issuer, Subject Alternate Name, Issuer Alternate Name and Others. Thus, a device can only publish to a topic that matches with the Certificate ID associated with the device in the registry like `arn:aws:iot:us-east-1:123456789012:topic/${iot:CertificateId}` Rather than `arn:aws:iot:us-east-1:123456789012:topic/xxxxxxxxxx`

2 – Secure the devices and their credentials

How do you secure your devices and protect device credentials? Your IoT devices and identity principals (certificates, private keys, tokens, etc.) must be secured throughout their lifecycle. To ensure 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 2.1	Required	Use a separate hardware or a secure area on your devices to store credentials
☐	BP 2.2	Highly Recommended	Use a trusted platform module (TPM) to implement cryptographic controls
☐	BP 2.3	Recommended	Use protected boot and persistent storage encryption

For more details, see the following links and information

- AWS Partner Network (APN) Blog: Adding Secure Element Support Using AWS IoT Greengrass Hardware Security Integration (HSI)
- Secure Elements in Amazon FreeRTOS
- AWS Partner Network (APN) Blog: Implementing Secure Authentication with AWS IoT and Microchip's Trust Platform
- AWS re:Invent 2019: Designing secure IoT solutions from the edge to cloud
- AWS Partner Device Catalog: Tamper-Proof Secure Element (SE)
- AWS Partner Device Catalog

- Wikipedia: Trusted Computing Group
- ARM Developer: Arm TrustZone Technology
- GlobalPlatform: TEE System Architecture v1.2
- GOWIN Semiconductor Corp.: GW1NSE SecureFPGA
- AWS Documentation: What is AWS Signer?
- AWS Signer: Integrated Services
- Embedded.com Technical Article: Securing the IoT: Part 2 – Secure boot as root of trust
- Espressif IoT Development Framework (ESP-IDF): Secure Boot
- The Internet of Things on AWS – Official Blog: Unlock the value of embedded security IP to build secure IoT products at scale
- The Internet of Things on AWS – Official Blog: Securing Amazon FreeRTOS devices at scale with Infineon OPTIGA Trust X
- AWS Whitepaper: Device Manufacturing and Provisioning with X.509 Certificates in AWS IoT Core

Best Practice 2.1 – Use a separate hardware or a secure area on your devices to store credentials

A 'secure element' is any hardware feature you can use to protect the device identity at rest. Secure storage at rest helps reduce the risk of unauthorized use of the device identify. Never store or cache device credentials outside of the secure element. Use a secure network, such as using TLS, or a secure manufacturing facility, when transmitting credentials to the secure element. Securely handling a device's identity helps ensure that your hardware and application are resilient to potential security issues that occur in unprotected systems. A secure element provides encryption 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).

Recommendation 2.1.1 - Use tamper-resistant hardware that offloads the security encryption and communication from the IoT application

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

See the following for more information:

- AWS IoT Greengrass: Hardware security integration

Recommendation 2.1.2 - Use cryptographic API operations with the secure element hardware for protecting the secrets on the device

Ensure that any security modules are accessed using the latest security protocols. For example, in Amazon FreeRTOS, use the PKCS#11 libraries for protecting secrets. See the following for more details:

- Amazon FreeRTOS: corePKCS11 library

Recommendation 2.1.3 - Use the AWS Partner Device Catalog to find AWS Partners that offer secure hardware elements or 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 TPM or a secure element, and implement the security features of Amazon FreeRTOS. Use AWS IoT Partners that offer qualified secure elements for storing IoT device identities. Refer to the following for details:

- AWS Partner Device Catalog

Best Practice 2.2 - 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 ensure 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.

See the following for more information:

- The Internet of Things on AWS – Official Blog: Using a Trusted Platform Module for endpoint device security in AWS IoT Greengrass

Recommendation 2.2.2 - Perform cryptographic operations inside the TPM to avoid a third party gaining unauthorized access

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

Recommendation 2.2.3 - 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.

Recommendation 2.2.4 - 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. For example, refer to AWS Partner Device Catalog , that has various device solutions with PUFs such as LPC54018 IoT Solution by NXP.

Best Practice 2.3 - 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 ensure 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 ensures 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 ensures that the storage and cryptographic elements are secured in absence of a TPM or secure element. The disk controller needs to ensure that all read accesses to the disk are transparently decrypted at-runtime.

Recommendation 2.3.1 - 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.

Recommendation 2.3.2 – 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 reduce the surface area of the attack.

Recommendation 2.3.3 – 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.

Recommendation 2.3.4 - Use services that enable you to 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 Amazon FreeRTOS, ensure that the firmware images are signed before deployment.

3 – Manage user access rights for IoT devices

How do you authenticate and authorize user access to your IoT applications? By requiring users to be authenticated and authorized, you can reduce the risk of misuse or downtime of your IoT devices.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 3.1	Required	Implement authentication and authorization for users accessing IoT resources
☐	BP 3.2	Highly Recommended	Decouple access to your IoT infrastructure from your IoT applications

For more details, see the following links and information.

- AWS IoT Core: How AWS IoT works with IAM
- The Internet of Things on AWS – Official Blog: Configuring Cognito User Pools to Communicate with AWS IoT Core
- AWS Security Blog: How to Use New Advanced Security Features for Amazon Cognito User Pools
- AWS Security Blog: How to define least-privileged permissions for actions called by AWS services
- AWS Identity and Access Management: Security best practices in IAM
- AWS Whitepaper: Designing MQTT Topics for AWS IoT Core

Best Practice 3.1 - Implement authentication and authorization for users accessing IoT resources

It enables end users with secure access to connected IoT devices and equipment via different channels such as web or mobile devices. Without valid authentication and authorization, devices can be subjected to compromises or malicious attempts.

Recommendation 3.1.1 - 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 connected devices. 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 or OAuth2.0 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, SAML, etc.) for authenticating users.

Recommendation 3.1.2 - Enable users to be authorized with least privileged access

Authorization is the process of granting permissions to an authenticated identity. You grant permissions to your end users in AWS IoT Core using data plane and control plane IAM policies through the Identity broker. Control plane API allows you to perform administrative tasks like creating or updating certificates, things, rules, and so on. Data plane API allows 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. If you are using a different Identity broker than Amazon Cognito, use AWS IoT custom authorizers to invoke lambda functions that will create the required IAM policies.

Recommendation 3.1.3 – Adopt least privilege when assigning user permissions

Adopt the least privilege principle and assign only the minimum required permissions to user roles. For example, with Amazon Cognito this can be achieved, by setting up role-based access through IAM policies for authenticated (think of consumers, admins) and unauthenticated users. Consumers or unauthenticated users should not be allowed to run destructive actions against IoT services, such as detaching policies, deleting CA, or deleting certificate.

Best Practice 3.2 - Decouple access to your IoT infrastructure from the IoT applications

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

See the following link for more details:

- The Internet of Things on AWS – Official Blog: Provision Devices Globally with AWS IoT

Recommendation 3.2.1 - Use an API layer between the application and IoT layer

Build an application interface layer to reduce the blast radius of 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. This will reduce the blast radius of the IoT data plane from end users.

4 – Analyze application logs to identify security events

How do you analyze application logs and device 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 4.1	Required	Collect and analyze logs and metrics to capture authorization errors and failures to enable appropriate response
☐	BP 4.2	Highly Recommended	Alert when on security events, misconfiguration, and behavior violations are detected
☐	BP 4.3	Recommended	Alert on non-compliant device configurations and remediate using automation

For more details, see the following links and information.

- AWS IoT Core Developer Guide: Monitor AWS IoT using CloudWatch Logs
- AWS IoT Core Developer Guide: AWS IoT metrics and dimensions
- AWS IoT Device Defender: Audit
- AWS IoT Device Defender: Detect

Best Practice 4.1 – 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.

Recommendation 4.1.1 - 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` and `Subscribe.AuthError`.
- Configure CloudWatch alarms for each of the preceding metrics to alarm based on levels higher than normal for your workload.

Best Practice 4.2 – Alert when on security events, misconfiguration, and behavior violations are detected

Audit the configuration of your devices and detect and alert when a device behavior differs from the expected behavior. It provides visibility into operational data that can indicate potential security issues active in the device fleet.

Recommendation 4.2.1 – Enable metrics to detect security events from the data plane

Create a threat model to detect events from security vulnerabilities or device compromises. You can detect events based on configured rules or Machine Learning (ML) models. For example, create a security profile in AWS IoT Device Defender, that detects unusual device behavior that may be indicative of a compromise by continuously monitoring high-value security metrics from 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 datapoint reported for these metrics against user-defined behaviors (rules) and alerts you if an anomaly is detected. When you use ML Detect, the feature sets device behaviors automatically with machine learning to monitor device activities.

Recommendation 4.2.2 – Enable auditing to check misconfigurations

Enable auditing to check for misconfigurations on a regular basis. Audit your device-related resources such as X.509 certificates, permissions, and Client IDs. Additionally, check configurations that are out of compliance with security best practices, such as multiple devices using the same identity, or overly permissive policies that can allow one device to read and update data for many other devices.

Recommendation 4.2.3 – Ensure alerting on a behavior violation

Enable alarming or notifications when the device behavior is anomalous based on configured rules or ML models. For example, AWS IoT Device Defender will alert you with the metric datapoint reported by the device when an ML model flags the datapoint as anomalous. This removes the need for you to define accurate behaviors of your devices and helps you get started with monitoring more quickly and easily.

Best Practice 4.3 - Alert on non-compliant device configurations and remediate using automation

Enable auditing to continuously assess configurations and metrics on the device. security configurations can be impacted by the passage of time and new threats are constantly emerging. For example, cryptographic algorithms once known to provide secure digital signatures for device certificates can be weakened by advances in the computing and cryptoanalysis methods.

Recommendation 4.3.1 – Ensure regular auditing for identifying configuration issues

Audit checks are necessary to determine that device stays configured with required best practices throughout its lifecycle. For instance, its necessary to audit devices regularly on basic checks such as logging, shared certificates and unique device id's. For example, AWS IoT Device Defender 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's

supported natively are `LOGGING_DISABLED_CHECK`, `IOT_POLICY_OVERLY_PERMISSIVE_CHECK`, `DEVICE_CERTIFICATE_SHARED_CHECK`, and `CONFLICTING_CLIENT_IDS_CHECK`.

Recommendation 4.3.2 – 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 such as adding things to a thing group, replacing default policy version and updating device certificate. Or you can enable a mitigation action to re-enable logging and publish the finding to Amazon SNS should the `LOGGING_DISABLED_CHECK` find that logging is not enabled.

5 – Plan the security lifecycle of your IoT devices

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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 5.1	Required	Build an incident response mechanism to address security events at scale
☐	BP 5.2	Highly Recommended	Require timely vulnerability notifications and software updates from your providers

For more details, see the following links and information.

- The Common Criteria for Information Technology Security Evaluation
- ISO/IEC 207035-1:2016 Information technology – Information security incident management – Part 1
- ISO/IEC 207035-2:2016 Information technology – Information security incident management – Part 2
- AWS re:Invent 2019: Designing secure IoT solutions from the edge to cloud

Best Practice 5.1 – 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.

Recommendation 5.1.1 – Ensure 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.

Recommendation 5.1.2 – Quarantine any device that deviates from expected behavior

Inspect the device for potential compromise of the configurations, firmware or applications using device logs or metrics. If a compromise 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 to quarantine the device. When sending code to devices, the best practice is to sign the code file. 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.

Recommendation 5.1.3 – 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 blast radius along with rollout and abort criteria's for a failsafe solution. For example, you can use AWS IoT Jobs for OTA updates of security patch and device configurations in a staged manner with required rollout and abort configurations.

Best Practice 5.2 - Require timely vulnerability notifications and software updates from your providers

Components in a device bill of materials (BOM), such as secure elements for certificate storage or a trusted platform module (TPM), 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.

Recommendation 5.2.1 – Ensure 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, consider using AWS Partner Device Catalog where you can find devices and hardware to help you explore, build, and go to market with your IoT solutions.

6 – Manage device certificate lifecycles

How do you manage device certificates, including installation, validation, revocation, and rotation? To protect and encrypt data in transit from an IoT device to the cloud, most IoT broker supports TLS-based mutual authentication using X.509 certificates. Device makers must provision a unique identity, including a unique private key and X.509 certificate, into each device. Certificates are long-lived credentials and managed using a customer-owned Certificate Authority (CA), a third party CA or AWS IoT CA. Any hosted CA chosen must provide you the ability to validate, activate, deactivate, and rotate certificates.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 6.1	Required	Perform certificate lifecycle management

For more details, see the following links and information.

- AWS IoT Device Defender: Audit
- AWS IoT Device Defender: Detect
- The Internet of Things on AWS – Official Blog: Announcing Mitigation Actions for AWS IoT Device Defender
- The Internet of Things on AWS – Automating Security Remediation Using AWS IoT Device Defender
- The Internet of Things on AWS – Detect anomalies on connected devices using AWS IoT Device Defender
- The Internet of Things on AWS – Using Device Time to Validate AWS IoT Server Certificates
- The Internet of Things on AWS – Ten security golden rules for IoT solutions
- AWS re:Invent 2019: Designing secure IoT solutions from the edge to cloud
- Manage Security of Your IoT Devices with AWS IoT Device Defender - AWS Online Tech Talks

Best Practice 6.1 – Perform certificate lifecycle management

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

Recommendation 6.1.1 – Document your plan for managing certificates

As explained earlier, X509 certificates helps to establish the identity of devices and encrypts the traffic from the edge to cloud. Thus, 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 rotate device certificates, audit cloud or device-related configurations and permissions to ensure 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 rotate the expired or compromised certificates.

Recommendation 6.1.2 – Use certificates signed by your trusted intermediate CA for on-boarding devices

As a best practice, the root CA needs to be locked and protected to secure the chain of trust. The device certificates should be generated from an intermediate CA. So 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 Device defender to resolve the issues.

Recommendation 6.1.3 – Secure provisioning claims private keys and disable the certificate in case of misuse and record the event for further investigation

- Monitor provisioning claims for private keys at all times, including on the device.
- 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.
- For more:
  - https://docs.aws.amazon.com/iot/latest/developerguide/vulnerability-analysis-and-management.html
  - https://aws.amazon.com/blogs/iot/just-in-time-registration-of-device-certificates-on-aws-iot/

7 – Encrypt the device data

How do you ensure that device data is protected at rest and in transit? 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, reduces the risk of unauthorized 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 7.1	Required	Use encryption to protect IoT data in transit and at rest
☐	BP 7.2	Highly Recommended	Use data classification strategies to categorize data access based on levels of sensitivity
☐	BP 7.3	Recommended	Protect your IoT data in compliance with regulatory requirements

For more details, see the following links and information.

- AWS IoT Core Developer Guide: AWS IoT security
- AWS IoT Core Developer Guide: Data protection in AWS IoT Core
- Amazon FreeRTOS User Guide: Transport Layer Security
- AWS IoT Greengrass Developer Guide: Security in AWS IoT Greengrass
- AWS IoT Core Developer Guide: Security best practices in AWS IoT Core
- AWS IoT Greengrass Developer Guide: Security best practices in AWS IoT Greengrass
- The Internet of Things on AWS – Official Blog: Ten security golden rules for IoT solutions
- AWS Training and Certification Free Course: Deep Dive into AWS IoT Authentication and Authorization
- AWS Whitepaper: Securing Internet of Things (IoT) with AWS

Best Practice 7.1 – 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's 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 all data transmitted to and from your devices.

Recommendation 7.1.1 – 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. See AWS IoT Core Developer Guide's Transport security in AWS IoT.

Recommendation 7.1.2 – Encrypt data at rest or secrets on IoT devices

As explained earlier in section 2.3.3, 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, and TPMs. TEEs can add storage protections as well.

Best Practice 7.2 – 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.

See the following for more details:

- AWS IoT Greengrass Developer Guide: Manage data streams on the AWS IoT Greengrass core
- The Internet of Things on AWS – Official Blog: Designing dataflows for multi-schema messages in AWS IoT Analytics

Recommendation 7.2.1 – 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, personally identifiable data, etc.)

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.
- By using AWS IoT Analytics, you can create different workflows for storing classified data. For highly classified data, you can define a separate pipeline and data store.

Best Practice 7.3 – 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.

See the following for more information:

- AWS Cloud Enterprise Strategy Blog: Using a Cloud Center of Excellence (CCOE) to Transform the Entire Enterprise

Recommendation 7.3.1 – 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.

Recommendation 7.3.2 – 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, only services like Secrets Manager can retrieve secrets in an encrypted manner.

Recommendation 7.3.3 – 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.

Reliability checklist

The reliability pillar encompasses the ability of a workload to perform its intended function correctly and consistently when it's expected to. This includes the ability to operate and test the workload through its entire lifecycle.

8 – Design to withstand component and system faults

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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 8.1	Highly Recommended	Use the services provided by your vendors for integration and error handling

For more details, see the following links and information.

- AWS IoT Core Developer Guide: Log AWS IoT API calls using AWS CloudTrail
- AWS IoT Core Developer Guide: Monitor AWS IoT using CloudWatch Logs
- AWS IoT Core Developer Guide: AWS IoT metrics and dimensions
- AWS IoT Core Developer Guide: Monitor AWS IoT alarms and metrics using Amazon CloudWatch
- AWS IoT Core Developer Guide: Rules for AWS IoT
- AWS IoT API Reference: SetLoggingOptions

Best Practice 8.1 – Use the services provided by your vendors for integration and error handling

An IoT design consists of device software, connectivity and control services, and analytics services. Test the entire IoT ecosystem 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.

Recommendation 8.1.1 – Understand and apply the standard libraries available to manage your device firmware

1. Devices can be built on Amazon FreeRTOS, which provides connectivity, messaging, power management and device management libraries that are tested for reliability and designed for ease of use.

Recommendation 8.1.2 – Use log levels appropriate to the lifecycle stage of your workload

1. 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.
2. AWS IoT Secure Tunneling can be used to test and debug devices that are behind a restrictive firewall in the field.

9 – Provision elastic compute capacity for IoT message processing

How do you ensure that all IoT messages are processed? 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 ensure 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 9.1	Required	Dynamically scale cloud resources with utilization

For more details, see the following links and information.

- Rules for AWS IoT
- Amazon SQS

- Building loosely couple applications

Best Practice 9.1 – Dynamically scale cloud resources with 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.

Recommendation 9.1.1 – 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.

10 – Provision reliable storage for IoT data that has been sent to the cloud

How do you ensure that data will reach a reliable resting point before compute operations are performed? 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 10.1	Required	Store data before processing
☐	BP 10.2	Highly Recommended	Have mechanisms in place to compensate when the primary storage location is unavailable

For more details, see the following links and information.

- Amazon S3 Intelligent-Tiering
- AWS Storage Optimization
- AWS Personal Health Dashboard Status
- Track and respond to changes to your AWS resources
- Amazon S3

- AWS Cloud Storage Services
- Multi-Tiered Storage
- AWS Health
- AWS Personal Health Dashboard
- Amazon CloudWatch Logs
- AWS IoT Rules Engine
- Amazon CloudWatch Events
- Track and respond to changes to your AWS resources
- Creating a CloudWatch Events Rule That Triggers on an AWS API Call Using AWS CloudTrail
- Create a Custom Event Pattern for a CloudWatch Event Rule

Best Practice 10.1 – Store data before processing

Ensure 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.

Recommendation 10.1.1 – Use IoT Core Rules Engine to send data to Kinesis Data Firehose to batch and store data on Amazon Simple Storage Service (Amazon S3)

- IoT Rules Engine can send data to Kinesis Data Firehose to batch and store data on Amazon Simple Storage Service (Amazon S3). Intelligent tiering can be enabled on 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 any scale.

Best Practice 10.2 – Have mechanisms in place to compensate when the primary storage location is unavailable

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.

Recommendation 10.2.1 – Know how to monitor and take action on cloud storage failures for IoT data

- AWS Personal 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.

11 – Design to withstand connectivity failures

How do you ensure that your IoT device operates with intermittent connectivity to the cloud?
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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 11.1	Required	Synchronize device states upon connection to the cloud
☐	BP 11.2	Highly Recommended	Use device hardware with sufficient capacity to meet your data retention requirements while disconnected
☐	BP 11.3	Highly Recommended	Down sample data to reduce storage requirements and network utilization
☐	BP 11.4	Recommended	Use an exponential backoff with jitter and retry logic to connect remote devices to the cloud

For more details, see the following links and information.

- Device Shadow Service for AWS IoT
- AWS SDK handles the exponential back off

Best Practice 11.1 – 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.

Recommendation 11.1.1 – Utilize a digital device state digital representation to synchronize device state

- 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 Amazon 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 Amazon FreeRTOS device shadow API operations define functions to create, update, and delete AWS IoT Device Shadows.

Best Practice 11.2 – Use device hardware with sufficient capacity to meet your data retention requirements while disconnected

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 of time to prevent any loss of information.

Recommendation 11.2.1 – Leverage device edge software capabilities for storing data locally

- Using AWS IoT Greengrass for device software 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, the 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 on AWS IoT Greengrass Core for preprocessing
- The 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.
  - ETL with AWS IoT Greengrass solution accelerator
- Consider using AWS IoT SiteWise for data coming from disparate industrial equipment
  - The AWS IoT SiteWise connector sends local equipment data in AWS IoT SiteWise. You can use this connector to collect data from multiple OPC Unified Architecture (UA) servers and publish it to AWS IoT SiteWise.
  - AWS IoT SiteWise connector with AWS IoT Greengrass can 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.
  - AWS IoT Greengrass: AWS IoT SiteWise connectors

Best Practice 11.3 – 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.

Recommendation 11.3.1 – Leverage device edge software capabilities for downsampling

- Using AWS IoT Greengrass for device software to down sample data.
  - Local Lambda functions 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.

Best Practice 11.4 – 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 prevent them from creating unnecessary peak traffic.

Recommendation 11.4.1 – Implement logic in the cloud to notify the device operator if a device has not connected for an extended period of time

- AWS IoT Events can be used to monitor devices remotely.
- Remote monitoring using AWS IoT Events

Recommendation 11.4.2 – Use device edge software and the SDK to leverage built in exponential back off 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 Amazon FreeRTOS.
- AWS SDK handles the exponential back off
- AWS IoT Device SDK for C uses "IOT_MQTT_RETRY_MS_CEILING" for setting maximum retry interval limit.

Recommendation 11.4.3 – Establish alternate network channels to meet requirements

- Have a separate failover network channel to deliver critical messages to AWS IoT. Failover channels can include WiFi, cellular netoworks, 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.

12 – Design devices to have an accurate time

How do you ensure that your device accurately determines UTC? 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 12.1	Required	Use NTP to maintain time synchronization on devices
☐	BP 12.2	Highly Recommended	Provide devices access to NTP servers

For more details, see the following links and information.

- The Internet of Things on AWS – Official Blog: Using Device Time to Validate AWS IoT Server Certificates

- AWS News Blog: Keeping Time With Amazon Time Sync Service
- Setting time for your instance

Best Practice 12.1 – 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 prevent it from being able to connect to the cloud.

Recommendation 12.1.1 – 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.

Recommendation 12.1.2 – 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's likely to include the ntpd daemon
- You can import an NTP client to your platform if using Amazon 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 prevent the device from deviating from UTC
- https://www.pool.ntp.org/en/vendors.htmlChrony is a different implementation of NTP than what ntpd uses and it's able to synchronize the system clock faster and with better accuracy than ntpd. Chrony can be set up as a client and server.
  - https://chrony.tuxfamily.org/

Best Practice 12.2 – 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.

Recommendation 12.2.1 – 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's able to synchronize the system clock faster and with better accuracy than ntpd. Chrony can be set up as a server and client.
  - https://chrony.tuxfamily.org/

13 – Control the frequency of message delivery to the device

How do you implement cloud-side mechanisms to control and modify message frequency to the device? 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 13.1	Required	Target messages to relevant devices
☐	BP 13.2	Highly Recommended	Implement retry and backoff logic to support throttling by device type

- Using AWS Lambda with AWS IoT Events
- Amazon DynamoDB with IoT
- Amazon SNS
- Amazon SQS

Best Practice 13.1 – 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.

Recommendation 13.2.1 – 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

Best Practice 13.2 – Implement retry and backoff logic to support throttling by device type

Retry and backoff 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.

Recommendation 13.3.1 – 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 (Amazon RDS), 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 RDS.

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

14 – Design to reliably update device firmware

How do you ensure that you can reliably update device firmware from your IoT application? 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 14.1	Required	Use a mechanism to deploy and monitor firmware updates
☐	BP 14.2	Highly Recommended	Implement firmware rollback capabilities in devices
☐	BP 14.3	Recommended	Implement support for incremental updates to target device groups
☐	BP 14.4	Recommended	Implement dynamic configuration management for devices

For more details, see the following links and information.

- AWS IoT connected device OTA using IoT Jobs
- AWS IoT Jobs remote operations
- AWS IoT Jobs management and executions

Best Practice 14.1 – Use a mechanism to deploy and monitor firmware updates

When performing over-the-air (OTA) updates to remote devices' firmware, we should always ensure 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.

Recommendation 14.1.1 – Use a cloud-based update orchestrator

- 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 Amazon FreeRTOS OTA using AWS IoT Jobs: By using AWS IoT Jobs for Amazon 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.

Best Practice 14.2 – Implement 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.

Recommendation 14.2.1 – Leverage a RTOS with functionality to roll back device firmware

By combining OTA agents provided by Amazon 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.

Best Practice 14.3 – Implement support for incremental updates to target device groups

It's a good practice to test new firmware on a small group of devices. Using a smaller group of devices for firmware updates helps ensure that the firmware as well as the upgrade process is well tested before the entire fleet is updated.

Recommendation 14.3.1 – Leverage 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.

Best Practice 14.4 – 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 reduces the amount of bandwidth compared to firmware updates.

Recommendation 14.4.1 Utilize 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 Defender and AWS IoT Device Management jobs to rotate expiring device certificates instead of pushing a new image with updated certificates.
- Secure Tunneling

- Device Shadows
- Device Defender

15 – Plan for disaster recovery (DR) of IoT workloads

How do you plan for disaster recovery (DR) in your IoT workloads? Though disasters are rare, you need comprehensive pre-planning and testing to mount an effective response when they do occur. Device data storage and processing should continue to support business outcomes even when a network is down due to large-scale events.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 15.1	Required	Design server software to initiate communication only with devices that are online
☐	BP 15.2	Highly Recommended	Implement multi-region support for IoT applications and devices
☐	BP 15.3	Recommended	Use edge devices to store and analyze data

For more details, see the following links and information.

- AWS IoT Greengrass Features
- AWS IoT Core endpoints and quotas
- Amazon Route 53
- AWS IoT device registration
- Device Shadow Service for AWS IoT
- Managing devices with AWS IoT
- AWS Infrastructure
- AWS Regional Table

Best Practice 15.1 – 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.

Recommendation 15.1.1 – Design client software to accept commands from the server

- Amazon 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.

Best Practice 15.2 – Implement multi-region support for IoT applications and devices

Cloud service providers have the same service in multiple regions. This architecture enables you 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.

Recommendation 15.2.1 – 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

Recommendation 15.2.2 – 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

Recommendation 15.2.3 – Use device services in all the 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

Best Practice 15.3 – 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.

Recommendation 15.3.1 – 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

Performance efficiency checklist

The performance efficiency pillar focuses on the efficient use of computing resources to meet requirements and the maintenance of that efficiency as demand changes and technologies evolve.

16 – Measure the performance of the IoT applications

How do you ensure your IoT application's performance? Defining and analyzing key performance metrics for your IoT applications helps you to understand the performance characteristics for your

application. Logging and end-to-end application monitoring are key to measuring, evaluating, and optimizing the performance of your IoT applications.

Follow the best practices and check if your workload is well-designed.

	ID	Priority	Best Practice
☐	BP 16.1	Highly Recommended	Analyze the runtime performances of your application
☐	BP 16.2	Recommended	Add timestamps to each message published
☐	BP 16.3	Recommended	Load test your IoT applications

For more details, see the following links and information.

- IoT Device Tester
- IoT Device Simulator
- Real time device monitoring with Kinesis Data Analytics
- Building an end-to-end industrial IoT solution with AWS IoT

Best Practice 16.1 – Analyze the runtime performances of your application

Application performances at runtime are different from what you can observe in a controlled test environment. Actively analyzing the performances 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.

Recommendation 16.1.1 – Analyzing anomaly as well as setting up a security profile will allow detection of anomalies along with having granular control of the device should changes be necessary

- Measurement of changes in connection patterns of devices may indicate some devices having a jittery network connection
- Measurement of device-side timestamps from multiple devices in comparison to arrival times on the cloud-side may indicate local network latency or additional hops in device path
- AWS IoT Device Defender Detect
- https://www.youtube.com/watch Anomaly detection using AWS IoT

Best Practice 16.2 – Add timestamps to each message published

Timestamps helps in determining delays that might incur during the transmission of a message from the device to the application. Timestamps can be associated to the message and to fields contained in the message. If a timestamp is included, the sent timestamp, along with the sensor or event data, is recorded on the cloud-side.

Recommendation 16.2.1 – Add timestamp 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 corresponding to the reception of the message.
- For example, AWS IoT Rules SQL language provides a `timestamp()` function to generate a timestamp at the reception of the message.

Recommendation 16.2.2 – 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 to obtain a reliable time when connected
  - Real Time Clock devices can be used to maintain an accurate time while the device lacks 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.
- Developer Guide - timestamp()
- Time series compression algorithms
- Delta encoding

Best Practice 16.3 – Load test your IoT applications

Applications can be complex and have multiple dependencies. Testing the application under load help identify problems before going into production. Load testing your IoT applications ensures 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 any performance bottlenecks, and apply mitigating strategies prior to releasing changes to your production systems.

Recommendation 16.3.1 – Simulate the real device behavior

- A device simulator should implement the device behavior as close as possible. Do not only test message publishing, but also: connections, reconnections, subscriptions, enrollment and other environmental disruptive events. Start testing at a lower load, progressively increasing to over 100%.
- For example:
  - 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.
- IoT Device Simulator
- From testing to scaling

17 – Measure the performance of the IoT devices

How do you ensure your IoT device's performance? Defining and analyzing key performance metrics for your hardware devices helps you to understand the performance characteristics of the devices and how they relate to the application performances. Capturing device logs and device metrics are key to measuring, evaluating, and optimizing the performance of your IoT devices.

Follow the best practices and check if your workload is well-designed.

	ID	Priority	Best Practice
☐	BP 17.1	Highly Recommended	Capture device diagnostic data into the IoT platform
☐	BP 17.2	Recommended	Measure, evaluate, and optimize firmware updates
☐	BP 17.3	Required	Limit the number of messages that devices receive and filter out

For more details, see the following links and information.

- AWS Solutions Library - Real-Time IoT Device Monitoring with Kinesis Data Analytics
- The Internet of Things on AWS – Official Blog – Using AWS IoT Services for Asset Condition Monitoring

Best Practice 17.1 – Capture device diagnostic data into the IoT platform

As the number of devices increases watch out for performance bottlenecks when all the devices connect to the cloud-side. These devices will be generating a large aggregate amount of data, and obtaining device diagnostics is critical for ensuring the understanding of the area of improvement. Using different types of device diagnostics, the immediate health of a device and likely those in proximity to that device can be obtained.

Recommendation 17.1.1 – 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. Below are some sample metrics:
  - TCP Connections
    - TCP_connections
    - Connections
    - Local_interface
  - Listening TCP/UDP Ports
    - Listening_TCP/UDP_ports
    - Interface
  - Network Statistics
    - Bytes_in/out
    - Packets_in/out
    - Network_statistics

- Sending Metrics from Devices

Best Practice 17.2 – Measure, evaluate, and optimize firmware updates

Firmware updates are critical to ensure that the IoT devices remain performant over time, but might sometime not have the expected impact. As you deploy firmware updates to your devices, monitoring your KPIs will ensure that the updates do not have any unintended impacts to the performance of your hardware devices or to your IoT applications.

Recommendation 17.2.1 – Implement canary deployment for device firmware

- Deploy new firmware to a limited set of devices and monitor the impact on performance before rolling out to the entire fleet. Abort deployment if degradation is detected.
- Example:
  - Use AWS IoT Jobs to manage 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 of firmware release appear to have been impacted, use AWS IoT Secure Tunneling, a feature of AWS IoT Device Management, to remotely troubleshoot the device.
  - Release firmware updates to remediate identified issues.
- For more:
  - The Internet of Things on AWS – Official Blog – Using Continuous Jobs with AWS IoT Device Management
  - The Internet of Things on AWS – Official Blog – Using Device Jobs for Over-the-Air Updates
  - The Internet of Things on AWS – Official Blog – Introducing Secure Tunneling for AWS IoT Device Management, a new secure way to troubleshoot IoT devices

Best Practice 17.3 – Limit the number of messages that devices receive and filter out

Devices might have constrained resources, and filtering messages at the edge might subject them to unnecessary load. This could be counterproductive from a power and memory consumption perspective. Sending only messages that the device make use of, reduces the load on the resources and ensures better performances.

Recommendation 17.3.1 – Structure the topics using the scope/verb approach

- In this way, you can subscribe to all messages for a given scope (for example, a device) or refine the subscription on a given scope and verb.
- Designing MQTT Topics for AWS IoT Core

18 – Operate applications within the limits of platform

How do you ensure that your application operates within the limits set by the platform? Being aware of the soft and hard limits of the platform and continuously monitoring the key performance indicators enables you to anticipate when actions must be taken to request increases in the limits and re-evaluate your architecture. Ensuring that your application operates within the limits of the services that you are building on is key to providing the optimal performance to your users.

Follow the best practices and check if your workload is well-designed.

	ID	Priority	Best Practice
☐	BP 18.1	Recommended	Monitor and manage your IoT service quotas using available tools and metrics

For more details, see the following links and information.

- AWS Limit Monitor solution
- AWS Service Quotas

Best Practice 18.1 – Monitor and manage your IoT service quotas using available tools and metrics

Monitoring enables you to be aware of which service limits you might be hitting, allowing you to engineer your application to cope with the hard limits or to request the increase of a soft limit with enough lead time.

Recommendation 18.1.1 – Familiarize yourself with the service limits of the different IoT service

- Pay attention to which limits are soft limits and which are hard limits as they require different approaches.
- For example:
  - A hard limit, such a control plane request rate, would require changes in the application behavior to avoid the event repeating too often. Workarounds for hard limits might require different design decisions, such as using multiple accounts. It's good to know the hard limits in advance so that you can make these design decisions as early as possible in the development process.
  - Soft limits 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 more:
  - AWS IoT Core
  - AWS IoT Device Defender
  - AWS IoT Device Management
  - AWS IoT Events
  - AWS IoT Greengrass
  - AWS IoT SiteWise
  - AWS IoT Things Graph
  - AWS IoT Analytics
  - AWS IoT 1-Click

- For example:
  - For AWS IoT Core alert on `RulesMessageThrottles`, `Connect.ClientIDThrottle`, `Connect.Throttle`, `PublishIn.Throttle`, `Subscribe.Throttle`, `Unsubscribe.Throttle`
  - For AWS IoT Analytics alert on `ActionExecutionThrottled`, `PipelineConcurrentExecutionCount`
  - For AWS IoT Device Management monitor Active continuous jobs, Active snapshot jobs in Service Quotas
  - For AWS IoT SiteWise Monitor the quotas in Service Quotas

- For more:
  - AWS IoT Analytics CloudWatch
  - AWS IoT Core monitoring with Amazon CloudWatch
  - Service Quotas for AWS IoT SiteWise
  - Service Quotas for AWS IoT Device Management
  - Amazon CloudWatch dashboards
  - AWS IoT Monitoring tools
  - Logging AWS IoT API calls with AWS CloudTrails

19 – Optimize the ingestion of telemetry data

How do you optimize the ingestion of telemetry data? Evaluating and optimizing your IoT application for its specific needs, whether telemetry data ingestion or controlling devices in the field, ensures that you get the best outcomes in balancing performances and cost. Separating the way that your application handles data collected through sensors or device probes from command-and-control flows helps achieve better performance.

Follow the best practices and check if your workload is well-designed.

	ID	Priority	Best Practice
☐	BP 19.1	Highly Recommended	Identify the ingestion mechanisms that best fit your use case
☐	BP 19.2	Required	Evaluate network connectivity and data freshness requirements
☐	BP 19.3	Recommended	Optimize data sent from devices to backend services

For more details, see the following links and information.

- Integrate your IoT data with your data lake
- Ingest data in AWS IoT SiteWise
- Industrial data ingestion and processing solutions

Best Practice 19.1 – 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/operational complexity tradeoff. Multiple mechanisms might be needed. It provides the optimal ingestion path for the data generated by your devices to obtain the best performance and costs trade-offs.

Recommendation 19.1.1 – Evaluate ingestion mechanism for telemetry data

- Determine if the communication pattern is uni-directional (device to backend) or bi-directional. For example:

  - HTTPS should be considered if your device is acting as an aggregator and needs to send more than 100 messages per second instead of opening multiple MQTT connections. 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 Analytics provides an HTTP API that is capable of batching several messages and is suitable for high rate data ingestion when the data is consumed in near-real-time fashion and a service-integrated data storage, data processing and data retention and replay are desired.
  - AWS IoT SiteWise provides an HTTP API to ingest operational data from industrial applications which needs to be stored for a limited period of time 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 out-of-the-box exporters to Amazon Kinesis Data Streams and AWS IoT Analytics (for example, industrial gateways).
- 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.
- For more:
  - AWS IoT Core supported protocols
  - AWS IoT Analytics
  - AWS IoT SiteWise
  - Amazon Kinesis Data Streams
  - Amazon Kinesis Video Streams
  - Amazon S3
  - Amazon S3 presigned URLs
  - AWS IoT Greengrass stream manager

Best Practice 19.2 – Evaluate network connectivity and data freshness requirements

It enables you to make the right assumptions on the local data storage and data transmission needed to satisfy the requirements of your workload. It also provides a clear understanding of the requirements of the workload and allows you to determine the hardware and software needs of the devices and the platform.

Recommendation 19.2.1 - 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.

Recommendation 19.2.2 – Right size the offline persistent storage to ensure your application objective can be obtained without wasting resources

- The AWS IoT Greengrass message spooler can be configured with an offline message queue for messages that need to be sent to the AWS IoT Core. The size and type of storage should be configured according to the needs of the workload.
- For more:
  - MQTT QoS
  - Publish/subscribe AWS IoT Core MQTT messages

Best Practice 19.3 – Optimize data sent from devices to backend services

Optimizing the amount of data sent by the devices at the edge allows the backend to more easily 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.

Recommendation 19.3.1 – Aggregate or compress data at the edge

- You can aggregate data points at the edge before sending it to the cloud, such as performing statistical aggregation, frequency histograms, signal processing.
- For example, if you are using AWS IoT Greengrass you can implement data processing at the edge with a combination of streams and Lambda functions.
- For more:
  - Run Lambda functions on the AWS IoT Greengrass core
  - Industrial OEE workshop
  - AWS IoT Greengrass stream manager

Cost optimization checklist

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 very first proof of concept to the ongoing operation of production workloads, adopting the practices in this document can enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, thus allowing your business to maximize its return on investment.

20 – Choose cost-efficient tools for data aggregation of your IoT workload

How do you deliver, enrich, and aggregate data across your IoT workload? 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 20.1	Required	Use a data lake for raw telemetry data
☐	BP 20.2	Highly Recommended	Provide a self-service interface for end users to search, extract, manage, and update IoT data
☐	BP 20.3	Highly Recommended	Track and manage the utilization of data sources
☐	BP 20.4	Highly Recommended	Aggregate data at the edge where possible

For more details, see the following links and information.

- AWS IoT Events
- AWS IoT Analytics - Analytics for IoT devices
- AWS Samples on GitHub - AWS IoT Analytics Workshop
- AWS Pricing Calculator

Best Practice 20.1 – 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.

Recommendation 20.1.1 – Categorize telemetry types and map to storage capabilities

- 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 more:
  - AWS storage types
  - AWS re:Invent 2018: Building with AWS Databases: Match Your Workload to the Right Database (DAT301)

Best Practice 20.2 – Provide a self-service interface for end users to search, extract, manage, and update IoT data

With inexpensive 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 will encourage experimentation and speed up change by removing barriers for teams to gain access to the data they need to make decisions.

Recommendation 20.2.1 – Use an architecture that allows various end users to easily find, obtain, enhance, and share data

Recommendation 20.2.2 – 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

For more:

- 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

Best Practice 20.3 – Track and manage the utilization of data sources

Applications and users evolve over time, and IoT solutions can generate very large volumes of data very 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.

Recommendation 20.3.1 Track and manage the utilization of data sources to identify hot and cold spots to assess value of data

- 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.
- For more:
  - Monitoring Amazon S3 metrics with Amazon CloudWatch
  - Find cost of your S3 buckets using AWS Cost Explorer

Best Practice 20.4 – 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.

Recommendation 20.4.1 – Examine device telemetry for opportunities to batch and aggregate data

- 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, 10K of device telemetry data might be packaged as one 10-K message, two 5-K messages, or 10 1-K messages. Each packaging format has implications outside of cost such as network traffic (10 1-K messages will each add their own header messaging as opposed to a single 10-K 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.

Recommendation 20.4.2 – Use cost calculators to model different approaches for message size and count

- The AWS Pricing Calculator can estimate IoT costs for specific message sizes, traffic, and operations.

21 – Optimize cost of interactions between devices and IoT platform

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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 21.1	Highly Recommended	Select services to optimize cost
☐	BP 21.2	Highly Recommended	Implement and configure telemetry to reduce data
☐	BP 21.3	Highly Recommended	Use shadow only for slowly changing data
☐	BP 21.4	Highly Recommended	Group and tag IoT devices and messages for cost allocation
☐	BP 21.5	Highly Recommended	Implement and configure device messaging to reduce data transfer costs

For more details, see the following links and information.

- AWS IoT Pricing

Best Practice 21.1 – 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.

Recommendation 21.1.1 Select services to optimize cost

- Review your IoT architecture to find communication patterns and scenarios that could map to service discount features.
- With AWS IoT Core Rules Engine Basic Ingest, you can publish directly to a rule without messaging charges.
- Use your registry of things only for primarily immutable data such as `serialNumber`.

- For your device's shadow, minimize the frequency of reads and writes to reduce the total metered operation and your operating costs.
- For more:
  - AWS IoT Rules Engine Basic Ingest
  - AWS IoT Pricing

Best Practice 21.2 – 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.

Recommendation 21.2.1 – 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.

Best Practice 21.3 – Use shadow only for slowly 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/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.

Recommendation 21.3.1 – 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 understand cost implications.
- For more:
  - Last Will and Testament (LWT) Lifecycle Event

Best Practice 21.4 – Group and tag IoT devices and messages for cost allocation

An IoT billing group enables you to tag devices by categories related to your IoT application. You should 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.

Recommendation 21.4.1 Use AWS IoT Core Billing Groups to tag IoT devices for cost allocation

1. Add tracking elements to messages and devices to help trace source such as product model and serial number.
2. Ensure that your system can group and organize data by both technical and business entity.

3. For more:
   a. Tagging IoT Billing Groups

Best Practice 21.5 – 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.

Recommendation 21.5.1 – 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.

Recommendation 21.5.2 – Evaluate cost of streaming services versus IoT messaging services

- 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.

22 – Optimize cost of raw telemetry data

How do you manage 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 22.1	Required	Use lifecycle policies to archive your data
☐	BP 22.2	Highly Recommended	Evaluate storage characteristics for your use case and align with the right services
☐	BP 22.3	Recommended	Store raw archival data on cost effective services

For more details, see the following links and information.

- S3 Life Cycle

Best Practice 22.1 – 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—going to market quickly, shipping new features, or meeting a deadline—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.

Recommendation 22.1.1 – Evaluate your organization's data retention and handling requirements and configure your AWS services to support them

- 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.

Best Practice 22.2 – 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.

Recommendation 22.2.1 – Evaluate velocity and the volume of data coming from IoT devices when selecting storage services

- For data at high scale of devices/time/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/other characteristics—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.

Best Practice 22.3 – Store raw archival data on cost effective services

Using the right storage solution for a specific data type will align costs with usage.

Recommendation 22.3.1 Use an object store for archival storage

- 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.

Operational excellence checklist

The operational excellence pillar includes the ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures.

23 – Evaluate IoT applications against operational goals

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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 23.1	Required	Enable appropriate responses to events

For more details, see the following links and information.

- Design with Ops in Mind
- Build a Monitoring Plan

Best Practice 23.1 – 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 prevent or reduce significant business interruption. By defining a minimum set of logs, metrics, and alarms, your operations team can provide a first line of defense against significant business interruption.

Recommendation 23.1.1 – Configure logging to capture and store at least error-level events

- Use AWS IoT service logging options to capture error events in CloudWatch Logs

Recommendation 23.1.2 – 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.
- For example:
  - Create a dashboard in CloudWatch with a widget that runs a CloudWatch Logs Insights query
  - For AWS IoT Core logs, set the log group to `AWSIotLogsV2` and the query to `filter logLevel="ERROR" | stats count(*) as errorCount by bin(5m) | sort errorCount desc`
- For more:

- Analyzing Log Data with CloudWatch Logs Insights

Recommendation 23.1.3 – Review 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 sensors across manufacturing plants 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.ServerError`, `Connect.Throttle`. Activity in any of 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 any of the new CloudWatch alarms changes status.
- For more:
  - Monitor AWS IoT alarms and metrics using Amazon CloudWatch

Recommendation 23.1.4 – Configure an automated monitoring and alerting tool to detect common symptoms and warnings of operational impact

- 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:
  - AWS IoT Device Defender Audit

24 – Educate people to operate IoT workload at scale

How do you ensure 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 ensure 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 24.1	Required	Train team members supporting your IoT workloads on the lifecycle of IoT applications and your business objectives

For more details, see the following links and information.

- AWS IoT Foundation Series
- AWS IoT Security Series
- IoT Atlas

Best Practice 24.1 – 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.

Recommendation 24.1.1 – 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.

Recommendation 24.1.2 – 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.
- For more:
  - Runbook
  - AWS IoT Secure Tunneling

25 – Govern device fleet provisioning processes

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. Defining how devices are provisioned must include how the devices are manufactured and how they are registered. 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 25.1	Required	Document how devices join your fleet from manufacturing to provisioning
☐	BP 25.2	Recommended	Use programmatic techniques to provision devices at scale

	ID	Priority	Best Practice
☐	BP 25.3	Highly Recommended	Use device level features to enable re-provisioning
☐	BP 25.4	Recommended	Use data-driven auditing metrics to detect if any of your IoT devices might have been compromised
☐	BP 25.5	Highly Recommended	Bootstrap devices to use the endpoint with the least latency

For more details, see the following links and information.

- Provisioning devices that don't have device certificates using fleet provisioning
- Just-in-time provisioning
- Bulk registration
- Provisioning devices that have device certificates
- Security best practices in AWS IoT Core
- Provisioning with a bootstrap certificate in AWS IoT Core

Best Practice 25.1 – 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.

Recommendation 25.1.1 – 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 ensure that newly provisioned devices are yours.

Recommendation 25.1.2 – Assign device metadata to enable easy 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 all 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.

Best Practice 25.2 – 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 ensures a well-documented, reliable, and programmatic provisioning mechanism that is consistent across all devices devoid of any human errors.

Recommendation 25.2.1 – 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.

Recommendation 25.2.2 – Use programmatic bootstrapping mechanisms, if you are bringing your own certificates

- Determine if you will or won't have device information beforehand
- If you don't 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.
  - o 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.

Best Practice 25.3 – 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 prevent its misuse. Only after a provisioning process is invoked and approved, should the device be allowed to operate on the system as designed.

Recommendation 25.3.1 – 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.

Recommendation 25.3.2 – 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.
- Ensure that this list is securely transferred from the manufacturer to you, is encrypted, and is not publicly accessible.
- Ensure that any bootstrap certificate used is signed by a certificate authority (CA) you own or trust.

Best Practice 25.4 – 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.

Recommendation 25.4.1 – Validate and secure the manufacturer-provided list of allowed devices

- Validate the list of devices that the manufacturer shared to ensure it has not been tampered with. Ensure 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.

Recommendation 25.4.2 – 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 prevent the device from any future use.
- Example:
  - 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 platform.
  - Use AWS IoT Device Defender to identify security issues and deviations from best practices.
- For more:
  - AWS IoT Core - Activate or deactivate a client
  - AWS IoT Core - Security best practices in AWS IoT
  - AWS IoT Core - Vulnerability analysis and management in AWS IoT Cored

Best Practice 25.5 – Bootstrap devices to use the endpoint with the least latency

In IoT, bootstrapping refers to the process of assigning identity to the device and enabling communications with an endpoint. Devices in a global fleet should be provisioned in the regional data center nearest to its physical location for minimum latency. Each device should get its regional endpoint and certificate no later than the time of bootstrapping. Each device is provisioned in the nearest to its physical location and gets the certificate and IoT endpoint at the time of bootstrapping. This ensures best possible latency for bidirectional communications.

Recommendation 25.5.1 - Obtain key metadata and regional endpoint for the device at the time of bootstrapping

- The device signs its thing name with a private key and sends a provisioning request to a pre-defined cloud endpoint. If the device uses its own private key, it provides a certificate signing request (CSR) in the provisioning request. If a CSR is not present in the request, AWS IoT creates the private key.
- IoT services in the cloud receive and validate the request and thereafter, provision the device.

Recommendation 25.5.2 – Use automated mechanisms to audit the configuration of your devices, monitor connected devices to detect abnormal behavior, and mitigate security risks

- For example, use AWS IoT Device Defender to continually audit your IoT configurations to make sure that they aren't deviating from security best practices.

Recommendation 25.5.3 – Use temporary, limited-privilege security tokens to communicate with cloud services

- For example, AWS IoT provides the credentials provider feature that allows a caller to authenticate AWS requests by having an X.509 certificate. The credentials provider authenticates a caller using an X.509 certificate, and vends a temporary, limited-privilege security token. The token can be used to sign and authenticate any AWS request.
- For more:
  - The Internet of Things on AWS – Official Blog: Provision Devices Globally with AWS IoT
  - AWS Security Blog: How to Eliminate the Need for Hardcoded AWS Credentials in Devices by Using the AWS IoT Credentials Provider
  - AWS IoT Core - Authorizing direct calls to AWS

26 – 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 26.1	Required	Provision devices into predefined hierarchies
☐	BP 26.2	Recommended	Use static and dynamic device hierarchies to support fleet operations
☐	BP 26.3	Recommended	Use index and search services to enable rapid identification of target devices

For more details, see the following links and information.

- AWS IoT Core - Thing types

- AWS IoT Core - Static thing groups
- AWS IoT Core - Dynamic thing groups
- AWS IoT Core - Fleet indexing service

Best Practice 26.1 – Provision devices into predefined hierarchies

Categorizing your fleet proactively into groups and hierarchies is imperative to a good organizational and search framework. All new devices are associated with the appropriate predefined hierarchy branch, making it simpler and efficient to inherit policies. Well-defined and structured organization of devices makes it easier to find devices using indexed searches.

Recommendation 26.1.1 – Identify common attributes of devices and specify them as static types

- Define device types to classify description and configuration information that are common to all devices of the same type.
- For example, define a `LightBulb` type with manufacturer, wattage (power), and other attributes common to a light bulb. Document which of your devices should have the type `LightBulb` and specify values for each of the attributes defined.

Recommendation 26.1.2 – Create a plan for devices to be filed into the group hierarchy as they are provisioned during onboarding

- Create a hierarchy for your groups, such as grouping multiple sensors in a single vehicle and grouping multiple vehicles in a fleet. That way, your devices can inherit access policies based on the group hierarchy.
- For example:
  - A static group for devices of same generation or manufacturer—Associate IAM policies to the group that can be inherited by all the devices under that group.
  - A dynamic group for all devices with more than 90% battery—Push firmware updates only to devices with high battery levels.

Recommendation 26.1.3 – Categorize how devices are organized depending on their connectivity, metadata, or runtime data

- 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.

Best Practice 26.2 – 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.

Recommendation 26.2.1 – Assign static types to devices with common attributes

- Define device types to classify description and configuration information that are common to all devices of the same type.

- Use provisioning templates to assign this static type to devices as they are provisioned for the first time.
- For example, define a `LightBulb` thing type in AWS IoT Core with manufacturer, wattage (power) and other attributes common to a light bulb. Associate all the devices in the thing registry of the type `LightBulb` to this type and specify values for each of the attribute defined.

Recommendation 26.2.2 – 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.

Recommendation 26.2.3 – Build a device index to efficiently search for devices, and aggregate registry data, runtime data, and device connectivity data

- Use a fleet indexing service 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.

Best Practice 26.3 – 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 easy to index and organize the device data, and search for any device by any attribute. Ingesting device data to a search service, for example, Amazon OpenSearch Service (OpenSearch Service), makes it easy to use powerful search, visualization, and analytics capabilities of Amazon ES to organize and search for devices. You can ingest your device data and the state to Amazon ES seamlessly.

Recommendation 26.3.1 – 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 Kinesis Data Firehose through the AWS IoT Core rules engine.
- Ingest the data from Kinesis Data Firehose to Amazon OpenSearch Service (OpenSearch Service) through built-in destination options for Amazon ES.
- Configure search and visualizations on the data directly or through the OpenSearch Dashboards console.
- For more:
  - AWS IoT Core - Fleet indexing service

    AWS IoT Core - AWS IoT Device Shadow service
  - What is Amazon OpenSearch Service?
  - The Internet of Things on AWS – Official Blog: Archive AWS IoT Device Shadows in Amazon OpenSearch Service
  - Analyze device-generated data with AWS IoT and Amazon OpenSearch Service

27 – Monitor the status of IoT devices

How do you monitor the status of your devices in your IoT application? 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.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 27.1	Required	Collect lifecycle events from the device fleet
☐	BP 27.2	Highly Recommended	Configure your devices to communicate their status periodically
☐	BP 27.3	Recommended	Use device state management services to detect status and connectivity patterns

For more details, see the following links and information.

- AWS IoT Core - Lifecycle events
- AWS IoT Core - Monitoring AWS IoT
- AWS IoT Greengrass - CloudWatch Metrics connector
- What is AWS IoT Events?
- AWS IoT Core - Querying for aggregate data

Best Practice 27.1 – Collect lifecycle events from the device fleet

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.

Recommendation 27.1.1 – Subscribe to lifecycle events

- 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.

Recommendation 27.1.2 – Send periodic keep-alive messages from the devices

- 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.
- Ensure 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.

Best Practice 27.2 – Configure your devices to communicate their status periodically

Implement Last Will and Testament (LWT) messages and periodic device keep-alive messages.

Recommendation 27.2.1 – Implement a well-designed device connectivity state machine

- Ensure 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.

Recommendation 27.2.3 – Use device connection and disconnection status for anomaly detection

- Use connectivity data patterns from devices to detect anomalous behavior in their communication patterns.
- For more:
  - AWS IoT Now Supports WebSockets, Custom Keepalive Intervals, and Enhanced Console
  - AWS IoT Device Defender Now Provides Statistical Anomaly Detection and Data Visualization
  - AWS Solutions Library: Real-Time IoT Device Monitoring with Kinesis Data Analytics

Best Practice 27.3 – 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.

Recommendation 27.3.1 - 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.
- For more:
  - AWS IoT Core - Managing thing indexing

28 – Segment the device operations of the IoT workload

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 suboptimally and minimize response time to security events.

Follow the best practices and check if your workload is well-architected.

	ID	Priority	Best Practice
☐	BP 28.1	Recommended	Use static and dynamic device attributes to identify devices with anomalous behavior

For more details, see the following links and information.

- AWS IoT Core - Static thing groups
- AWS IoT Core - Managing thing group indexing
- AWS IoT Device Defender Now Provides Statistical Anomaly Detection and Data Visualization
- What is AWS IoT Events?
- AWS IoT Core - Querying for aggregate data

Best Practice 28.1 – 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.

Recommendation 28.1.1 – 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, message transmission, etc.
- 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.
  - AWS IoT Core - Authorization
  - AWS IoT - Device Defender

Conclusion

This lens provides architectural guidance for designing and building reliable, secure, efficient, and cost-effective IoT workloads in the cloud. We captured common architectures and overarching IoT design tenets. The document also discussed the Well-Architected pillars through the IoT Lens, providing you with a set of questions to consider for new or existing IoT architectures. Applying the framework to your architecture helps you build robust, stable, and efficient systems, leaving you to focus on running IoT workloads and pushing the boundaries of the field to which you're committed. The IoT landscape is continuing to evolve as the ecosystem of tooling and processes grows and matures. As this evolution occurs, we will continue to update this paper to help you ensure that your IoT applications are well-architected.

Contributors

The following individuals and organizations contributed to this document:

- Ryan Burke: Worldwide Tech Leader, IoT
- Olawale Oladehin: Worldwide SA Leader, IoT
- Neel Mitra: Principal Specialist Solutions Architect
- Arun Viswanathan: Senior Specialist Solutions Architect
- Massimiliano Angelino: Principal Specialist Solutions Architect
- Matthew Stanlake: Americas SA Leader, IoT
- Lon Miller: Senior Solutions Architect
- David Malone: Senior Specialist Solutions Architect
- Chris Snowden: Senior Technical Account Manager
- Ravikant Gupta: Senior Solutions Architect
- Syed Rehan: Senior Specialist Solutions Architect
- Rajeev Muralidhar: Principal IoT Specialist SA
- Albert Lee: Senior IoT Specialist SA
- David Malone: Senior IoT Specialist SA
- Neel Sendas: Senior Technical Account Manager
- Mike Ruiz: Principal Solutions Architect, Well-Architected (Technical Reviewer)
- Jongnam Lee: Senior Solutions Architect, Well-Architected (Program Manager)

Document history

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

Change	Description	Date
Minor update (p. 40)	BP 21.1 changed from Required to Highly Recommended.	December 1, 2022
Minor update (p. 58)	Fixed broken link.	June 30, 2022
Initial publication (p. 58)	IoT Lens Checklist first published.	May 28, 2021

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

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.