IOTSEC 8. How are you planning the security lifecycle of your IoT devices?

IOTSEC 9. How are you ensuring timely notification of security bugs in your third-party firmware and software components?

Although there is no cloud infrastructure to manage when using AWS IoT services, there are integration points where AWS IoT Core interacts on your behalf with other AWS services. For example, the AWS IoT rules engine consists of rules that are analyzed that can trigger downstream actions to other AWS services based on the MQTT topic stream. Since AWS IoT communicates to your other AWS resources, you must ensure that the right service role permissions are configured for your application.

Data Protection

Before architecting an IoT application, data classification, governance, and controls must be designed and documented to reflect how the data can be persisted in the cloud, and how data should be encrypted, whether on a device or between the devices and the cloud. Unlike traditional cloud applications, data sensitivity and governance extend to the IoT devices that are deployed in remote locations outside of your network boundary. These techniques are important because they support protecting personally identifiable data transmitted from devices and complying with regulatory obligations.

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

In addition to reducing data storage locally, there are other mitigations that must be implemented at the end of life of a device. First, the device should offer a reset option which can reset the hardware and firmware to a default factory version. Second, your IoT application can run periodic scans for the last logon time of every device. Devices that have been offline for too long a period of time, or are associated with inactive customer accounts, can be revoked. Third, encrypt sensitive data that must be persisted on the device using a key that is unique to that particular device.

IOTSEC 10: How do you classify, manage, and protect your data in transit and at rest?

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

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

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

Incident Response

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

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

- IoT devices are organized in different groups based on device attributes such as location and hardware version.
- IoT devices are searchable by dynamic attributes, such as connectivity status, firmware version, application status, and device health.

- OTA updates can be staged for devices and deployed over a period of time. Deployment rollouts are monitored and can be automatically aborted if devices fail to maintain the appropriate KPIs.
- Any update process is resilient to errors, and devices can recover and roll back from a failed software update.
- Detailed logging, metrics, and device telemetry are available that contain contextual information about how a device is currently performing and has performed over a period of time.
- Fleet-wide metrics monitor the overall health of your fleet and alert when operational KPIs are not met for a period of time.
- Any individual device that deviates from expected behavior can be quarantined, inspected, and analyzed for potential compromise of the firmware and applications.

IOTSEC 11: How do you prepare to respond to an incident that impacts a single device or a fleet of devices?

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

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

Key AWS Services

The essential AWS security services in IoT are the AWS IoT registry, AWS IoT Device Defender, AWS Identity and Access Management (IAM), and Amazon Cognito. In combination, these services allow you to securely control access to IoT devices, AWS

services, and resources for your users. The following services and features support the five areas of security:

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

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

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

Infrastructure protection: AWS IoT Core is a cloud service that lets connected devices easily and securely interact with cloud applications and other devices. The AWS IoT rules engine in AWS IoT Core uses IAM permissions to communicate with other downstream AWS services.

Data protection: AWS IoT includes encryption capabilities for devices over TLS to protect your data in transit. AWS IoT integrates directly with services, such as Amazon S3 and Amazon DynamoDB, which support encryption at rest. In addition, AWS Key Management Service (AWS KMS) supports the ability for you to create and control keys used for encryption. On devices, you can use AWS edge offerings such as Amazon FreeRTOS, AWS IoT Greengrass, or the AWS IoT Embedded C SDK to support secure communication.

Incident response: AWS IoT Device Defender allows you to create security profiles that can be used to detect deviations from normal device behavior and trigger automated responses including AWS Lambda. AWS IoT Device Management should be

used to group devices that need remediation and then using AWS IoT Jobs to deploy fixes to devices.

Resources

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

Documentation and Blogs

- IoT Security Identity
- AWS IoT Device Defender
- IoT Authentication Model
- MQTT on port 443
- Detect Anomalies with Device Defender

Whitepapers

- MQTT Topic Design

Reliability Pillar

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

Design Principles

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

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

Definition

There are three best practice areas for reliability in the cloud:

1. Foundations
2. Change management
3. Failure management

To achieve reliability, a system must have a well-planned foundation and monitoring in place, with mechanisms for handling changes in demand, requirements, or potentially defending an unauthorized denial of service attack. The system should be designed to detect the failure and automatically heal itself.

Best Practices

Foundations

IoT devices must continue to operate in some capacity in the face of network or cloud errors. Design device firmware to handle intermittent connectivity or loss in connectivity in a way that is sensitive to memory and power constraints. IoT cloud applications must

also be designed to handle remote devices that frequently transition between being online and offline to maintain data coherency and scale horizontally over time. Monitor overall IoT utilization and create a mechanism to automatically increase capacity to ensure that your application can manage peak IoT traffic.

To prevent devices from creating unnecessary peak traffic, device firmware must be implemented that prevents the entire fleet of devices from attempting the same operations at the same time. For example, if an IoT application is composed of alarm systems and all the alarm systems send an activation event at 9am local time, the IoT application is inundated with an immediate spike from your entire fleet. Instead, you should incorporate a randomization factor into those scheduled activities, such as timed events and exponential back off, to permit the IoT devices to more evenly distribute their peak traffic within a window of time.

The following questions focus on the considerations for reliability.

IOTREL 1. How do you handle AWS service limits for peaks in your IoT application?

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

The most important part of your IoT scaling approach is to ensure that you architect around any hard limits because exceeding limits that are not adjustable results in application errors, such as throttling and client errors. Hard limits are related to throughput on a single IoT connection. If you find your application exceeds a hard limit, we recommend redesigning your application to avoid those scenarios. This can be done in several ways, such as restructuring your MQTT topics, or implementing cloud-side logic to aggregate or filter messages before delivering the messages to the interested devices.

Soft limits in AWS IoT traditionally correlate to account-level limits that are independent of a single device. For any account-level limits, you should calculate your IoT usage for a single device and then multiply that usage by the number of devices to determine the base IoT limits that your application will require for your initial product launch. AWS

recommends that you have a ramp-up period where your limit increases align closely to your current production peak usage with an additional buffer. To ensure that the IoT application is not under provisioned:

- Consult published AWS IoT CloudWatch metrics for all of the limits.
- Monitor CloudWatch metrics in AWS IoT Core.
- Alert on CloudWatch throttle metrics, which would signal if you need a limit increase.
- Set alarms for all thresholds in IoT, including MQTT connect, publish, subscribe, receive, and rule engine actions.
- Ensure that you request a limit increase in a timely fashion, before reaching 100% capacity.

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

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

IOTREL 2. What is your strategy for managing ingestion and processing throughput of IoT data to other applications?

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

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

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

IOTREL 3. How do you handle device reliability when communicating with the cloud?

IoT solution reliability must also encompass the device itself. Devices are deployed in remote locations and deal with intermittent connectivity, or loss in connectivity, due to a variety of external factors that are out of your IoT application's control. For example, if an ISP is interrupted for several hours, how will the device behave and respond to these long periods of potential network outage? Implement a minimum set of embedded operations on the device to make it more resilient to the nuances of managing connectivity and communication to AWS IoT Core.

Your IoT device must be able to operate without internet connectivity. You must implement robust operations in your firmware provide the following capabilities:

- Store important messages durably offline and, once reconnected, send those messages to AWS IoT Core.
- Implement exponential retry and back-off logic when connection attempts fail.
- If necessary, have a separate failover network channel to deliver critical messages to AWS IoT. This can include failing over from Wi-Fi to standby cellular network, or failing over to a wireless personal area network protocol (such as Bluetooth LE) to send messages to a connected device or gateway.
- Have a method to set the current time using an NTP client or low-drift real-time clock. A device should wait until it has synchronized its time before attempting a connection with AWS IoT Core. If this isn't possible, the system provides a way for a user to set the device's time so that subsequent connections can succeed.

- Send error codes and overall diagnostics messages to AWS IoT Core.

Change Management

IOTREL 4. How do you roll out and roll back changes to your IoT application?

It is important to implement the capability to revert to a previous version of your device firmware or your cloud application in the event of a failed rollout. If your application is well-architected, you will capture metrics from the device, as well as metrics generated by AWS IoT Core and AWS IoT Device Defender. You will also be alerted when your device canaries deviate from expected behavior after any cloud-side changes. Based on any deviations in your operational metrics, you need the ability to:

- Version all of the device firmware using Amazon S3.
- Version the manifest or execution steps for your device firmware.
- Implement a known-safe default firmware version for your devices to fall back to in the event of an error.
- Implement an update strategy using cryptographic code-signing, version checking, and multiple non-volatile storage partitions, to deploy software images and rollback.
- Version all IoT rules engine configurations in CloudFormation.
- Version all downstream AWS Cloud resources using CloudFormation.
- Implement a rollback strategy for reverting cloud side changes using CloudFormation and other infrastructure as code tools.

Treating your infrastructure as code on AWS allows you to automate monitoring and change management for your IoT application. Version all of the device firmware artifacts and ensure that updates can be verified, installed, or rolled back when necessary.

Failure Management

IOTREL 5. How does your IoT application withstand failure?

Because IoT is an event-driven workload, your application code must be resilient to handling known and unknown errors that can occur as events are permeated through your application. A well-architected IoT application has the ability to log and retry errors in data processing. An IoT application will archive all data in its raw format. By archiving

all data, valid and invalid, an architecture can more accurately restore data to a given point in time.

With the IoT rules engine, an application can enable an IoT error action. If a problem occurs when invoking an action, the rules engine will invoke the error action. This allows you to capture, monitor, alert, and eventually retry messages that could not be delivered to their primary IoT action. We recommend that an IoT error action is configured with a different AWS service from the primary action. Use durable storage for error actions such as Amazon SQS or Amazon Kinesis.

Beginning with the rules engine, your application logic should initially process messages from a queue and validate that the schema of that message is correct. Your application logic should catch and log any known errors and optionally move those messages to their own DLQ for further analysis. Have a catch-all IoT rule that uses Amazon Kinesis Data Firehose and AWS IoT Analytics channels to transfer all raw and unformatted messages into long-term storage in Amazon S3, AWS IoT Analytics data stores, and Amazon Redshift for data warehousing.

IOTREL 6. How do you verify different levels of hardware failure modes for your physical assets?

IoT implementations must allow for multiple types of failure at the device level. Failures can be due to hardware, software, connectivity, or unexpected adverse conditions. One way to plan for thing failure is to deploy devices in pairs, if possible, or to deploy dual sensors across a fleet of devices deployed over the same coverage area (meshing).

Regardless of the underlying cause for device failures, if the device can communicate to your cloud application, it should send diagnostic information about the hardware failure to AWS IoT Core using a diagnostics topic. If the device loses connectivity because of the hardware failure, use Fleet Indexing with connectivity status to track the change in connectivity status. If the device is offline for extended periods of time, trigger an alert that the device may require remediation.

Key AWS Services

Use Amazon CloudWatch to monitor runtime metrics and ensure reliability. Other services and features that support the three areas of reliability are as follows:

Foundations: AWS IoT Core enables you to scale your IoT application without having to manage the underlying infrastructure. You can scale AWS IoT Core by requesting account level limit increases.

Change management: AWS IoT Device Management enables you to update devices in the field while using Amazon S3 to version all firmware, software, and update manifests for devices. AWS CloudFormation lets you document your IoT infrastructure as code and provision cloud resources using a CloudFormation template.

Failure management: Amazon S3 allows you to durably archive telemetry from devices. The AWS IoT rules engine Error action enables you to fall back to other AWS services when a primary AWS service is returning errors.

Resources

Refer to the following resources to learn more about our best practices related to reliability:

Documentation and Blogs

- Using Device Time to Validate AWS IoT Server Certificates
- AWS IoT Core Limits
- IoT Error Action
- Fleet Indexing
- IoT Atlas

Performance Efficiency Pillar

The **Performance Efficiency** pillar focuses on using computing resources efficiently. Key topics include selecting the right resource types and sizes based on workload requirements, monitoring performance, and making informed decisions to maintain efficiency as business and technology needs evolve. The performance efficiency pillar focuses on the efficient use of computing resources to meet the requirements and the maintenance of that efficiency as demand changes and technologies evolve.

Design Principles

In addition to the overall Well-Architected Framework performance efficiency design principles, there are three design principles for performance efficiency for IoT in the cloud:

- **Use managed services**: AWS provides several managed services across databases, compute, and storage which can assist your architecture in increasing the overall reliability and performance.
- **Process data in batches**: Decouple the connectivity portion of IoT applications from the ingestion and processing portion in IoT. By decoupling the ingestion layer, your IoT application can handle data in aggregate and can scale more seamlessly by processing multiple IoT messages at once.
- **Use event driven architectures**: IoT systems publish events from devices and permeate those events to other subsystems in your IoT application. Design mechanisms that cater to event-driven architectures, such as leveraging queues, message handling, idempotency, dead-letter queues, and state machines.

Definition

There are four best practice areas for Performance Efficiency in the cloud:

1. Selection
2. Review
3. Monitoring
4. Tradeoffs

Use a data-driven approach when selecting a high-performance architecture. Gather data on all aspects of the architecture, from the high-level design to the selection and configuration of resource types. By reviewing your choices on a cyclical basis, you will ensure that you are taking advantage of the continually evolving AWS platform. Monitoring ensures that you are aware of any deviation from expected performance and allows you to act. Your architecture can make tradeoffs to improve performance, such as using compression or caching, or relaxing consistency requirements.

Best Practices

Selection

Well-Architected IoT solutions are made up of multiple systems and components such as devices, connectivity, databases, data processing, and analytics. In AWS, there are several IoT services, database offerings, and analytics solutions that enable you to quickly build solutions that are well-architected while allowing you to focus on business objectives. AWS recommends that you leverage a mix of managed AWS services that

best fit your workload. The following questions focus on these considerations for performance efficiency.

IOTPERF 1. How do you select the best performing IoT architecture?

When you select the implementation for your architecture, use a data-driven approach based on the long-term view of your operation. IoT applications align naturally to event driven architectures. Your architecture will combine services that integrate with event-driven patterns such as notifications, publishing and subscribing to data, stream processing, and event-driven compute. In the following sections we look at the five main IoT resource types that you should consider (devices, connectivity, databases, compute, and analytics).

Devices

The optimal embedded software for a particular system will vary based on the hardware footprint of the device. For example, network security protocols, while necessary for preserving data privacy and integrity, can have a relatively large RAM footprint. For intranet and internet connections, use TLS with a combination of a strong cipher suite and minimal footprint. AWS IoT supports Elliptic Curve Cryptography (ECC) for devices connecting to AWS IoT using TLS. A secure software and hardware platform on device should take precedence during the selection criteria for your devices. AWS also has a number of IoT partners that provide hardware solutions that can securely integrate to AWS IoT.

In addition to selecting the right hardware partner, you may choose to use a number of software components to run your application logic on the device, including Amazon FreeRTOS and AWS IoT Greengrass.

IOTPERF 2. How do you select your hardware and operating system for IoT devices?

IoT Connectivity

Before firmware is developed to communicate to the cloud, implement a secure, scalable connectivity platform to support the long-term growth of your devices over time. Based on the anticipated volume of devices, an IoT platform must be able to scale the communication workflows between devices and the cloud, whether that is simple ingestion of telemetry or command and response communication between devices.

You can build your IoT application using AWS services such as EC2, but you take on the undifferentiated heavy lifting for building unique value into your IoT offering. Therefore, AWS recommends that you use AWS IoT Core for your IoT platform.

AWS IoT Core supports HTTP, WebSockets, and MQTT, a lightweight communication protocol designed to tolerate intermittent connections, minimize the code footprint on devices, and reduce network bandwidth requirements.

IOTPERF 3. How do you select your primary IoT platform?

Databases

You will have multiple databases in your IoT application, each selected for attributes such as the write frequency of data to the database, the read frequency of data from the database, and how the data is structured and queried. There are other criteria to consider when selecting a database offering:

- Volume of data and retention period.
- Intrinsic data organization and structure.
- Users and applications consuming the data (either raw or processed) and their geographical location/dispersion.
- Advanced analytics needs, such as machine learning or real-time visualizations.
- Data synchronization across other teams, organizations, and business units.
- Security of the data at the row, table, and database levels.
- Interactions with other related data-driven events such as enterprise applications, drill-through dashboards, or systems of interaction.

AWS has several database offerings that support IoT solutions. For structured data, you should use Amazon Aurora, a highly scalable relational interface to organizational data. For semi structured data that requires low latency for queries and will be used by multiple consumers, use DynamoDB, a fully managed, multi-region, multi-master database that provides consistent single-digit millisecond latency, and offers built-in security, backup and restore, and in-memory caching.

For storing raw, unformatted event data, use AWS IoT Analytics. AWS IoT Analytics filters, transforms, and enriches IoT data before storing it in a time series data store for analysis. Use Amazon SageMaker to build, train, and deploy machine learning models,

based off of your IoT data, in the cloud and on the edge using AWS IoT Services such as Greengrass Machine Learning Inference. Consider storing your raw formatted time series data in a data warehouse solution such as Amazon Redshift. Unformatted data can be imported to Amazon Redshift via Amazon S3 and Amazon Kinesis Data Firehose. By archiving unformatted data in a scalable, managed data storage solution, you can begin to gain business insights, explore your data, and identify trends and patterns over time.

In addition to storing and leveraging the historical trends of your IoT data, you must have a system that stores the current state of the device and provides the ability to query against the current state of all of your devices. This supports internal analytics and customer facing views into your IoT data.

The AWS IoT Shadow service is an effective mechanism to store a virtual representation of your device in the cloud. AWS IoT device shadow is best suited for managing the current state of each device. In addition, for internal teams that need to query against the shadow for operational needs, leverage the managed capabilities of Fleet Indexing, which provides a searchable index incorporating your IoT registry and shadow metadata. If there is a need to provide index based searching or filtering capability to a large number of external users, such as for a consumer application, dynamically archive the shadow state using a combination of the IoT rules engine, Kinesis Data Firehose, and Amazon ElasticSearch Service to store your data in a format that allows fine grained query access for external users.

IOTPERF 4. How do you select the database for your IoT device state?

Compute

IoT applications lend themselves to a high flow of ingestion that requires continuous processing over the stream of messages. Therefore, an architecture must choose compute services that support the steady enrichment of stream processing and the execution of business applications during and prior to data storage.

The most common compute service used in IoT is AWS Lambda, which allows actions to be invoked when telemetry data reaches AWS IoT Core or AWS IoT Greengrass. AWS Lambda can be used at different points throughout IoT. The location where you elect to trigger your business logic with AWS Lambda is influenced by the time that you want to process a specific data event.

Amazon EC2 instances can also be used for a variety of IoT use cases. They can be used for managed relational databases systems and for a variety of applications, such as web, reporting, or to host existing on-premises solutions.

IOTPERF 5. How do you select your compute solutions for processing AWS IoT events?

Analytics

The primary business case for implementing IoT solutions is to respond more quickly to how devices are performing and being used in the field. By acting directly on incoming telemetry, businesses can make more informed decisions about which new products or features to prioritize, or how to more efficiently operate workflows within their organization. Analytics services must be selected in such a way that gives you varying views on your data based on the type of analysis you are performing. AWS provides several services that align with different analytics workflows including time-series analytics, real-time metrics, and archival and data lake use cases.

With IoT data, your application can generate time-series analytics on top of the steaming data messages. You can calculate metrics over time windows and then stream values to other AWS services.

In addition, IoT applications that use AWS IoT Analytics can implement a managed AWS Data Pipeline consisting of data transformation, enrichment, and filtering before storing data in a time series data store. Additionally, with AWS IoT Analytics, visualizations and analytics can be performed natively using QuickSight and Jupyter Notebooks.

Review

IOTPERF 6. How do you evolve your architecture based on the historical analysis of your IoT application?

When building complex IoT solutions, you can devote a large percentage of time on efforts that do not directly impact your business outcome. For example, managing IoT protocols, securing device identities, and transferring telemetry between devices and the cloud. Although these aspects of IoT are important, they do not directly lead to differentiating value. The pace of innovation in IoT can also be a challenge.

AWS regularly releases new features and services based on the common challenges of IoT. Perform a regular review of your data to see if new AWS IoT services can solve a current IoT gap in your architecture, or if they can replace components of your architecture that are not core business differentiators. Leverage services built to aggregate your IoT data, store your data, and then later visualize your data to implement historical analysis. You can leverage a combination of sending timestamp information from your IoT device with leveraging services like AWS IoT Analytics and time-based indexing to archive your data with associated timestamp information. Data in AWS IoT Analytics can be stored in your own Amazon S3 bucket along with additional IT or OT operational and efficiency logs from your devices. By combining this archival state of data in IoT with visualization tools, you can make data driven decisions about how new AWS services can provide additional value and measure how the services improve efficiency across your fleet.

Monitoring

IOTPERF 7. How are you running end to end simulation tests of your IoT application?

IoT applications can be simulated using production devices set up as test devices (with a specific test MQTT namespace), or by using simulated devices. All incoming data captured using the IoT rules engine is processed using the same workflows that are used for production.

The frequency of end-to-end simulations must be driven by your specific release cycle or device adoption. You should test failure pathways (code that is only executed during a failure) to ensure that the solution is resilient to errors. You should also continuously run device canaries against your production and pre-production accounts. The device canaries act as key indicators of the system performance during simulation tests. Outputs of the tests should be documented and remediation plans should be drafted. User acceptance tests should be performed.

IOTPERF 8. How are you using performance monitoring in your IoT implementation?

There are several key types of performance monitoring related to IoT deployments including device, cloud performance, and storage/analytics. Create appropriate

performance metrics using data collected from logs with telemetry and command data. Start with basic performance tracking and build on the metrics as your business core competencies expand.

Leverage CloudWatch Logs metric filters to transform your IoT application standard output into custom metrics through regex (regular expressions) pattern matching. Create CloudWatch alarms based on your application's custom metrics to gain quick insight into your IoT application's behavior.

Set up fine-grained logs to track specific thing groups. During IoT solution development, enable DEBUG logging for a clear understanding of the progress of events about each IoT message as it passes from your devices through the message broker and the rules engine. In production, change the logging to ERROR and WARN.

In addition to cloud instrumentation, you must run instrumentation on devices prior to deployment to ensure that devices make the most efficient use of their local resources, and that firmware code does not lead to unwanted scenarios like memory leaks. Deploy code that is highly optimized for constrained devices and monitor the health of your devices using device diagnostic messages published to AWS IoT from your embedded application.

Tradeoffs

IoT solutions drive rich analytics capabilities across vast areas of crucial enterprise functions, such as operations, customer care, finance, sales, and marketing. At the same time, they can be used as efficient egress points for edge gateways. Careful consideration must be given to architecting highly efficient IoT implementations where data and analytics are pushed to the cloud by devices, and where machine learning algorithms are pulled down on the device gateways from the cloud.

Individual devices will be constrained by the throughput supported over a given network. The frequency with which data is exchanged must be balanced with the transport layer and the ability of the device to optionally store, aggregate, and then send data to the cloud. Send data from devices to the cloud at timing intervals that align to the time required by backend applications to process and take action on the data. For example, if you need to see data at a one-second increment, your device must send data at a more frequent time interval than one second. Conversely, if your application only reads data at an hourly rate, you can make a trade-off in performance by aggregating data points at the edge and sending the data every half hour.

IOTPERF 9. How are you ensuring that data from your IoT devices is ready to be consumed by business and operational systems?

IOTPERF 10. How frequently is data transmitted from devices to your IoT application?

The speed with which enterprise applications, business, and operations need to gain visibility into IoT telemetry data determines the most efficient point to process IoT data. In network constrained environments where the hardware is not limited, use edge solutions such as AWS IoT Greengrass to operate and process data offline from the cloud. In cases where both the network and hardware are constrained, look for opportunities to compress message payloads by using binary formatting and grouping similar messages together into a single request.

For visualizations, Amazon Kinesis Data Analytics enables you to quickly author SQL code that continuously reads, processes, and stores data in near-real-time. Using standard SQL queries on the streaming data allows you to construct applications that transform and provide insights into your data. With Kinesis Data Analytics, you can expose IoT data for streaming analytics.

Key AWS Services

The key AWS service for performance efficiency is Amazon CloudWatch, which integrates with several IoT services including AWS IoT Core, AWS IoT Device Defender, AWS IoT Device Management, AWS Lambda, and DynamoDB. Amazon CloudWatch provides visibility into your application's overall performance and operational health. The following services also support performance efficiency:

Selection

Devices: AWS hardware partners provide production ready IoT devices that can be used as part of you IoT application. Amazon FreeRTOS is an operating system with software libraries for microcontrollers. AWS IoT Greengrass allows you to run local compute, messaging, data caching, sync, and ML at the edge.

Connectivity: AWS IoT Core is a managed IoT platform that supports MQTT, a lightweight publish and subscribe protocol for device communication.

Database: Amazon DynamoDB is a fully managed NoSQL datastore that supports single digit millisecond latency requests to support quick retrieval of different views of your IoT data.

Compute: AWS Lambda is an event driven compute service that lets you run application code without provisioning servers. Lambda integrates natively with IoT events triggered from AWS IoT Core or upstream services such as Amazon Kinesis and Amazon SQS.

Analytics: AWS IoT Analytics is a managed service that operationalizes device level analytics while providing a time series data store for your IoT telemetry.

Review: The AWS IoT Blog section on the AWS website is a resource for learning about what is newly launched as part of AWS IoT.

Monitoring: Amazon CloudWatch Metrics and Amazon CloudWatch Logs provide metrics, logs, filters, alarms, and notifications that you can integrate with your existing monitoring solution. These metrics can be augmented with device telemetry to monitor your application.

Tradeoff: AWS IoT Greengrass and Amazon Kinesis are services that allow you to aggregate and batch data at different locations of your IoT application, providing you more efficient compute performance.

Resources

Refer to the following resources to learn more about our best practices related to performance efficiency:

Documentation and Blogs

- AWS Lambda Getting Started
- DynamoDB Getting Started
- AWS IoT Analytics User Guide
- Amazon FreeRTOS Getting Started
- AWS IoT Greengrass Getting Started
- AWS IoT Blog

Cost Optimization Pillar

The **Cost Optimization** pillar includes the continual process of refinement and improvement of a system over its entire lifecycle. From the initial design of your first proof of concept to the ongoing operation of production workloads, adopting the

practices in this paper will enable you to build and operate cost-aware systems that achieve business outcomes and minimize costs, allowing your business to maximize its return on investment.

Design Principles

In addition to the overall Well-Architected Framework cost optimization design principles, there is one design principle for cost optimization for IoT in the cloud:

- **Manage manufacturing cost tradeoffs**: Business partnering criteria, hardware component selection, firmware complexity, and distribution requirements all play a role in manufacturing cost. Minimizing that cost helps determine whether a product can be brought to market successfully over multiple product generations. However, taking shortcuts in the selection of your components and manufacturer can increase downstream costs. For example, partnering with a reputable manufacturer helps minimize downstream hardware failure and customer support cost. Selecting a dedicated crypto component can increase bill of material (BOM) cost, but reduce downstream manufacturing and provisioning complexity, since the part may already come with an onboard private key and certificate.

Definition

There are four best practice areas for Cost Optimization in the cloud:

1. Cost-effective resources
2. Matching supply and demand
3. Expenditure awareness
4. Optimizing over time

There are tradeoffs to consider. For example, do you want to optimize for speed to market or cost? In some cases, it's best to optimize for speed—going to market quickly, shipping new features, or meeting a deadline—rather than investing in upfront cost optimization. Design decisions are sometimes guided by haste as opposed to empirical data, as the temptation always exists to overcompensate rather than spend time benchmarking for a cost-optimal deployment. This leads to over-provisioned and under-optimized deployments. The following sections provide techniques and strategic guidance for your deployment’s initial and ongoing cost optimization.

Best Practices

Cost-Effective Resources

Given the scale of devices and data that can be generated by an IoT application, using the appropriate AWS services for your system is key to cost savings. In addition to the overall cost for your IoT solution, IoT architects often look at connectivity through the lens of BOM costs. For BOM calculations, you must predict and monitor what the long-term costs will be for managing the connectivity to your IoT application throughout the lifetime of that device. Leveraging AWS services will help you calculate initial BOM costs, make use of cost-effective services that are event driven, and update your architecture to continue to lower your overall lifetime cost for connectivity.

The most straightforward way to increase the cost-effectiveness of your resources is to group IoT events into batches and process data collectively. By processing events in groups, you are able to lower the overall compute time for each individual message. Aggregation can help you save on compute resources and enable solutions when data is compressed and archived before being persisted. This strategy decreases the overall storage footprint without losing data or compromising the query ability of the data.

COST 1. How do you select an approach for batch, enriched, and aggregate data delivered from your IoT platform to other services?

AWS IoT is best suited for streaming data for either immediate consumption or historical analyses. There are several ways to batch data from AWS IoT Core to other AWS services and the differentiating factor is driven by batching raw data (as is) or enriching the data and then batching it. Enriching, transforming, and filtering IoT telemetry data during (or immediately after) ingestion is best performed by creating an AWS IoT rule that sends the data to Kinesis Data Streams, Kinesis Data Firehose, AWS IoT Analytics, or Amazon SQS. These services allow you to process multiple data events at once.

When dealing with raw device data from this batch pipeline, you can use AWS IoT Analytics and Amazon Kinesis Data Firehose to transfer data to S3 buckets and Amazon Redshift. To lower storage costs in Amazon S3, an application can leverage lifecycle policies that archive data to lower cost storage, such as Amazon S3 Glacier.

Matching Supply and Demand

Optimally matching supply to demand delivers the lowest cost for a system. However, given the susceptibility of IoT workloads to data bursts, solutions must be dynamically

scalable and consider peak capacity when provisioning resources. With the event driven flow of data, you can choose to automatically provision your AWS resources to match your peak capacity and then scale up and down during known low periods of traffic.

The following questions focus on the considerations for cost optimization:

COST 2. How do you match the supply of resources with device demand?

Serverless technologies, such as AWS Lambda and API Gateway, help you create a more scalable and resilient architecture, and you only pay for when your application utilizes those services. AWS IoT Core, AWS IoT Device Management, AWS IoT Device Defender, AWS IoT Greengrass, and AWS IoT Analytics are also managed services that are pay per usage and do not charge you for idle compute capacity. The benefit of managed services is that AWS manages the automatic provisioning of your resources. If you utilize managed services, you are responsible for monitoring and setting alerts for limit increases for AWS services.

When architecting to match supply against demand, proactively plan your expected usage over time, and the limits that you are most likely to exceed. Factor those limit increases into your future planning.

Optimizing Over Time

Evaluating new AWS features allows you to optimize cost by analyzing how your devices are performing and make changes to how your devices communicate with your IoT.

To optimize the cost of your solution through changes to device firmware, you should review the pricing components of AWS services, such as AWS IoT, determine where you are below billing metering thresholds for a given service, and then weigh the trade-offs between cost and performance.

COST 3. How do you optimize payload size between devices and your IoT platform?

IoT applications must balance the networking throughput that can be realized by end devices with the most efficient way that data should be processed by your IoT

application. We recommend that IoT deployments initially optimize data transfer based on the device constraints. Begin by sending discrete data events from the device to the cloud, making minimal use of batching multiple events in a single message. Later, if necessary, you can use serialization frameworks to compress the messages prior to sending it to your IoT platform.

From a cost perspective, the MQTT payload size is a critical cost optimization element for AWS IoT Core. An IoT message is billed in 5-KB increments, up to 128 KB. For this reason, each MQTT payload size should be as close to possible to any 5 KB. For example, a payload that is currently sized at 6 KB is not as cost efficient as a payload that is 10 KB because the overall costs of publishing that message is identical despite one message being larger than the other.

In order to take advantage of the payload size, look for opportunities to either compress data or aggregate data into messages:

- You should shorten values while keeping them legible. If 5 digits of precision are sufficient then you should not use 12 digits in the payload.
- If you do not require IoT rules engine payload inspection, you can use serialization frameworks to compress payloads to smaller sizes.
- You can send data less frequently and aggregate messages together within the billable increments. For example, sending a single 2-KB message every second can be achieved at a lower IoT message cost by sending two 2-KB messages every other second.

This approach has tradeoffs that should be considered before implementation. Adding complexity or delay in your devices may unexpectedly increase processing costs. A cost optimization exercise for IoT payloads should only happen after your solution has been in production and you can use a data-driven approach to determine the cost impact of changing the way data is sent to AWS IoT Core.

COST 4. How do you optimize the costs of storing the current state of your IoT device?

Well-Architected IoT applications have a virtual representation of the device in the cloud. This virtual representation is composed of a managed data store or specialized IoT application data store. In both cases, your end devices must be programmed in a way that efficiently transmits device state changes to your IoT application. For example,

your device should only send its full device state if your firmware logic dictates that the full device state may be out of sync and would be best reconciled by sending all current settings. As individual state changes occur, the device should optimize the frequency it transmits those changes to the cloud.

In AWS IoT, device shadow and registry operations are metered in 1 KB increments and billing is per million access/modify operations. The shadow stores the desired or actual state of each device and the registry is used to name and manage devices.

Cost optimization processes for device shadows and registry focus on managing how many operations are performed and the size of each operation. If your operation is cost sensitive to shadow and registry operations, you should look for ways to optimize shadow operations. For example, for the shadow you could aggregate several reported fields together into one shadow message update instead of sending each reported change independently. Grouping shadow updates together reduces the overall cost of the shadow by consolidating updates to the service.

Key AWS Services

The key AWS feature supporting cost optimization is cost allocation tags, which help you to understand the costs of a system. The following services and features are important in the three areas of cost optimization:

- **Cost-effective resources**: Amazon Kinesis, AWS IoT Analytics, and Amazon S3 are AWS services that enable you to process multiple IoT messages in a single request in order to improve the cost effectiveness of compute resources.
- **Matching supply and demand**: AWS IoT Core is a managed IoT platform for managing connectivity, device security to the cloud, messaging routing, and device state.
- **Optimizing over time**: The AWS IoT Blog section on the AWS website is a resource for learning about what is newly launched as part of AWS IoT.

Resources

Refer to the following resources to learn more about AWS best practices for cost optimization.

Documentation and Blogs

- AWS IoT Blogs

Conclusion

The AWS Well-Architected Framework provides architectural best practices across the pillars for designing and operating reliable, secure, efficient, and cost-effective systems in the cloud for IoT applications. The framework provides a set of questions that you can use to review an existing or proposed IoT architecture, and also a set of AWS best practices for each pillar. Using the framework in your architecture helps you produce stable and efficient systems, which allows you to focus on your functional requirements.

Contributors

The following individuals and organizations contributed to this document:

- Olawale Oladehin, Solutions Architect Specialist, IoT
- Dan Griffin, Software Development Engineer, IoT
- Catalin Vieru, Solutions Architect Specialist, IoT
- Brett Francis, Product Solutions Architect, IoT
- Craig Williams, Partner Solutions Architect, IoT
- Philip Fitzsimons, Sr. Manager Well-Architected, Amazon Web Services

Document Revisions

Date	Description
December 2019	Updated to include additional guidance on IoT SDK usage, bootstrapping, device lifecycle management, and IoT
November 2018	First publication