



AWS Whitepaper

Implementing Low-Power Wide-Area Network (LPWAN) Solutions with AWS IoT



Implementing Low-Power Wide-Area Network (LPWAN) Solutions with AWS IoT: AWS Whitepaper

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

.....	vi
Abstract and introduction	i
Introduction	1
Choosing the connectivity technology	3
Assessing use case requirements	3
Evaluating technical fit	4
Deployment area requirements	4
Range, indoor, and underground coverage requirements	4
Battery life and power consumption requirements	5
Data rate and communication interval requirements	5
Device mobility requirements	6
Comparing LPWAN connectivity technologies	8
Scenarios for implementing an IoT solution	10
Integration with AWS Partners	10
Implementing LPWAN IoT solutions on AWS using NB-IoT and LTE-M	11
Implementing LPWAN IoT solutions with UDP	11
Pattern 1: Telco-provided fully integrated solution	13
Pattern 2: Customer-operated solution secured by Telco-provided VPN	15
Pattern 3: Customer-operated solution secured by DTLS	17
Implementing LPWAN IoT solutions with CoAP	18
Introduction to CoAP	18
CoAP client and server	18
CoAP resource model	19
Confirmable and non-confirmable requests	19
Block-wise transfer mode	21
Encryption in transit and device authentication	21
Architectural considerations	21
Implementation patterns	22
Implementing LPWAN IoT solutions with MQTT	26
Device telemetry ingestion	26
Device commands	27
Firmware updates	27
Implementing device provisioning	27
Managing cellular connectivity	27

Implementing LPWAN IoT solutions on AWS using LoRaWAN	29
LoRaWAN network architecture	30
LoRaWAN network server	31
LoRaWAN application server	32
LoRaWAN join server	32
LoRaWAN networks	32
Public LoRaWAN networks	32
Private LoRaWAN networks	33
Building a private LoRaWAN network	33
Managed LoRaWAN server with AWS IoT Core for LoRaWAN	34
LoRaWAN server as an AWS Partner software as a service (SaaS) solution	37
Customer-operated LoRaWAN server on the edge device	38
Public LoRaWAN networks	39
Implementing LPWAN IoT solutions on AWS using Sigfox	40
Implementing uplink from Sigfox devices to AWS Cloud	40
Implementing downlink from AWS Cloud to Sigfox devices	41
Conclusion	42
Contributors	43
Further reading	44
Document history	45
Appendix: Characteristics of LPWAN technologies	46
NB-IoT	46
Range and coverage	46
Data rate	47
Mobility	47
Spectrum licensing	48
Payload size	48
Further considerations	48
LTE-M	48
LTE-M compared to NB-IoT	48
Range and coverage	49
Data rate	49
Mobility	49
Battery life	49
Latency	49
Spectrum licensing	49

Payload size	50
Further considerations	50
LoRaWAN	50
Range and coverage	50
Data rate	51
Mobility	51
Battery life	51
Spectrum licensing	51
Application payload size	51
Latency and reachability	52
Further considerations	52
Sigfox	52
Mobility	52
Battery life	52
Spectrum licensing	53
Payload size	53
Latency	53
Notices	54
AWS Glossary	55

This whitepaper is for historical reference only. Some content might be outdated and some links might not be available.

Implementing Low-Power Wide-Area Network (LPWAN) Solutions with AWS IoT

Publication date: **December 17, 2021** ([Document history](#))

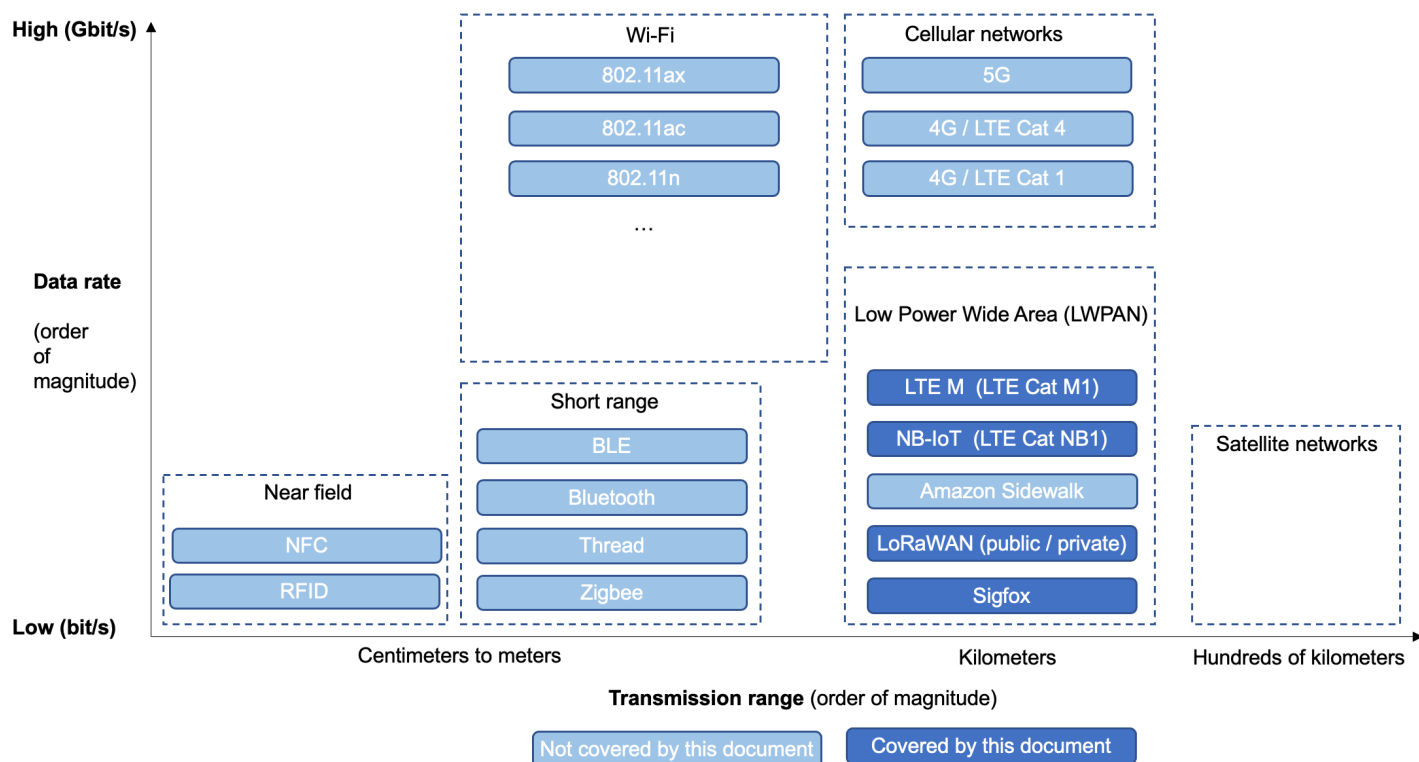
This whitepaper helps Amazon Web Services (AWS) customers to implement their Internet of Things (IoT) solutions with low-power wide-area network (LPWAN) connectivity technologies. First, this paper provides customers with a decision framework, to help them decide if LPWAN is the right choice for their IoT use case. Then, it provides an overview of LPWAN connectivity technologies and their capabilities, as well as implementation guidelines for implementing Narrowband Internet of Things (NB IoT), [LTE-M](#), Long Range Wide Area Network (LoRaWAN), and [Sigfox](#) connectivity for the IoT applications. Finally, a detailed description of NB-IoT, LTE-M, LoRaWAN, and Sigfox technologies is provided in the appendix.

This whitepaper is intended for technical architects, IoT cloud engineers, IoT security architects, and embedded engineers.

Introduction

When building Internet of Things (IoT) solutions on AWS, you have a broad choice of wired and wireless connectivity options. For example, in the area of wireless connectivity, you can choose from short range connectivity (for example, Bluetooth Low Energy (BLE), Zigbee, or Z-Wave), a low-power wide-area network (LPWAN) (for example, LoRaWAN, Narrowband Internet of Things (NB-IoT), LTE-M, or Sigfox), cellular (for example, LTE Cat 4), and satellite. This broad range of choices allows you to address specific requirements of individual use cases. It also introduces a certain complexity in the selection process. Because choosing a specific connectivity option can be costly to reverse (due to the impact on hardware design and procurement), a special diligence is required when selecting connectivity technology.

This whitepaper focuses on LPWAN connectivity technologies. First, it will help you to decide if LPWAN connectivity technologies are appropriate for a specific use case. Next, it provides you with information about the differences between individual LPWAN technologies, allowing you to make better trade-off decisions. Finally, it provides you with guidelines for implementing individual LPWAN technologies using AWS. The scope of this whitepaper is illustrated in the following diagram:



Connectivity technologies overview

When building IoT solutions, a decision for any connectivity technology shall be based on the requirements of the use cases and applications. The next section explores a framework you can use to evaluate these requirements.

Choosing the connectivity technology

A key aspect in making the correct decisions regarding your choice of connectivity technology is to understand connectivity requirements for your use case. After the requirements are defined, these requirements can be mapped to the characteristics of the available connectivity technologies. By performing this mapping, you can create a short list of connectivity technologies for your use case. Based on that short list you can decide if LPWAN connectivity technologies are a good fit for your use case.

Topics

- [Assessing use case requirements](#)
- [Evaluating technical fit](#)
- [Comparing LPWAN connectivity technologies](#)

Assessing use case requirements

When deciding for a connectivity technology for your use case, AWS suggests you consider a technical fit, operational impact, and commercial impact as evaluation criteria.

When evaluating a **technical fit**, AWS recommends analyzing the requirements of your use case regarding:

- Device deployment locations, range, and coverage
- Battery life and power consumption
- Data rate and communicational interval
- Device mobility, latency, and reachability
- Security and regulatory requirements

For **operational impact**, key considerations include:

- Ease of deployment
- Complexity of maintenance

For **commercial impact**, key considerations include:

- Costs of devices
- Recurring subscription fees
- Operational efforts

This whitepaper focuses on the analysis of the technical impact. Operational impact and commercial impact are out of scope for this whitepaper. The following sections describe the technical fit requirements in detail.

Evaluating technical fit

The purpose of technical fit evaluation is to ensure that a connectivity technology of your choice fulfills technical requirements of your use-case. AWS recommends you consider both imminent use-cases and also foreseen use-cases, which you expect to arise as your IoT solution evolves. This foresight of future use-cases is important, because choices of connectivity technology typically impact the IoT device hardware design, that may be costly to reverse. This section provides an overview of technical fit requirements.

Deployment area requirements

Availability of LPWAN connectivity options can vary significantly depending on the intended locations of your IoT device. If available in the intended location, public LPWAN networks (for example, NB-IoT/LTE-M networks, public LoRaWAN network, or Sigfox) can be a feasible choice. In the areas where public LPWAN networks are not available or are not economically feasible, an option to operate private LPWAN network can be considered, for example using LoRaWAN technology. You will find more information about building private LoRaWAN networks in the *Implementing LPWAN IoT solutions on AWS with LoRaWAN* section of this document.

Range, indoor, and underground coverage requirements

Whatever LPWAN connectivity technology you choose, your IoT device will need an intermediary to exchange data with the AWS Cloud or your edge device. For LPWAN technologies, examples of such intermediaries include a network operator cell tower or LoRaWAN gateway. For example, an asset tracking use case might require IoT devices to send and receive data when kilometers away from a cell tower or a gateway. Another example is an indoor IoT solution for food safety, where a few hundred meters of distance to a cell tower or LoRaWAN gateway are sufficient for the purpose of the use case.

For LPWAN, **range requirements** refer to the ability of your IoT device to send and receive data with an expected quality of service with the specified distance to the cell tower or a gateway. If your use case requires IoT devices to send and receive data with buildings or underground, then **indoor and underground coverage requirements** need to be considered. They refer to the expected ability of your IoT device to send and receive data, when there are obstacles such as walls between device and cell tower, or gateway.

LPWAN technologies are optimized for long-range transmission and indoor coverage (for example, inside buildings or underground).

Wi-Fi technology provides standards that support significant transmission ranges and indoor coverage. However, the power consumption of Wi-Fi devices can be higher than power consumption of LPWAN devices. If you consider Wi-Fi for your IoT solution, AWS recommends that you exercise special diligence in the evaluation of battery life and power consumption requirements described in the next section.

Battery life and power consumption requirements

Battery life requirements apply to IoT use cases for device that operate from batteries. For such use cases, it's important to define a needed duration of operation before battery replacement or recharge is required. The duration of battery-based operation is determined by the device's power consumption and the battery capacity.

The device's power consumption depends on used radio transmission technology and device activity patterns. For example, devices that use LPWAN technologies such as LoRaWAN, NB-IoT, LTE-M, and Sigfox can operate for years on the same low-capacity battery (for example, AA/AAA or CR2032), assuming they only need to send and receive data infrequently and can spend most of the time in an energy-efficient sleep mode.

However, if your use case requires devices to constantly listen for incoming messages, that would prevent devices from changing into the power saving model. That would significantly reduce the battery life regardless of the used wireless connectivity technology. For example, LoRaWAN Class C devices can constantly listen for incoming and more frequently send outgoing messages, at a price of having an external power source.

Data rate and communication interval requirements

Data rate describes the amount of data per time period an IoT device will need to exchange with your application to fulfill the requirements of your use case. When evaluating data rate, both uplink and downlink data rate requirements are considered. The uplink data rate relates to the data

your application is sending to the device for further processing. Examples of uplink data are sensor telemetry or video stream content from a security camera. The downlink data rate relates to the data your application is sending to the device. Examples of downlink data are device commands, device configuration updates, and firmware updates.

The maximum data rate differs for each LPWAN. Examples include Sigfox with up to 0.1 kilobit/second, LoRaWAN with up to 5,47 kilobit/second, NB-IoT with up to 26 kilobit/second (downlink), and LTE-M (LTE Cat M1) with up to one megabit/second. Consider that use of higher data rates tends to result in higher power consumption, and shorter battery life.

Communication interval requirements describe how frequently your use case requires the IoT device to send uplink or receive downlink. LPWAN technologies are used in use cases with infrequent communication (for example, few times a day to few times a week), while still maintaining context in the network for reachability. Using less frequent communication improves energy efficiency, allowing low-power operation, and longer battery life.

Device mobility requirements

Device mobility requirements describe if your use case requires your IoT devices to be stationary or moving. In the case of IoT devices that are moving, the important consideration is the area in which the devices are moving. If the IoT devices are sending and receiving data during the movement, velocity of movement is also considered.

In general, LPWAN technologies support both stationary and mobile devices. However, device mobility requirements can have a significant influence to the selection of specific LPWAN technology for your use case. Consult the descriptions of individual LPWAN technologies in [Appendix: Characteristics of LPWAN technologies](#) to learn more about their support for device mobility.

Not only will device mobility impact the specific LPWAN technology used, but it will also influence the decision to use private or public networks, because private networks are usually constrained to relatively small, fixed areas. If an IoT device has high-mobility requirements and will travel outside of a private network's coverage area, it might need to join a public network.

Finally, especially for use cases with cross-country mobility requirements, roaming capabilities of the network operator shall be considered. For example, by using roaming features of LTE-M and NB-IoT networks, IoT devices might establish connectivity in cross-country mobility scenarios. Similar concepts are also available for LoRaWAN and Sigfox.

Latency and reachability requirements

Latency can be defined as an amount of time it takes for a data sent by a device to arrive in a cloud application. For example, after a sensor measurement is performed by an IoT device, it should take no more than 10 seconds until this data can be processed by the cloud application. For LPWAN technologies, latency varies between milliseconds and a few seconds, depending on the selected LPWAN technology.

Reachability can be defined as the amount of time it takes until data sent by a cloud application arrives at a device. For an example of reachability requirement, assume a use case where an IoT device has an attached actuator (a device that is capable of converting electrical signals into physical action, such as a relay for operating a power switch). The cloud application will be used to activate the actuator (for example, a power switch) on that IoT device. In this example, reachability is a duration until the command from the cloud application arrives at the device and the power switch state is changed. A possible requirement for reachability can be: **the switch change command from the cloud application shall arrive on the IoT device in less than 10 minutes.**

If LPWAN connectivity is used on a battery-operated device, there is a trade-off between reachability duration and battery lifetime. This tradeoff is caused by the mechanism of energy-efficient power saving mode most LPWAN technologies offer. During power saving mode, power consumption is reduced, but devices are not capable of receiving downlink data. The longer time IoT devices are in listening mode, the lower the reachability and the higher the energy consumption.

Cost requirements

Cost considerations for individual connectivity technologies are out of scope for this whitepaper. However, AWS suggests that you consider the following costs when evaluating connectivity technologies:

- Device costs for IoT devices, and possibly necessary gateways
- Subscription and connectivity costs for network operators
- Operating costs for software components

Security and regulatory

AWS recommends that you define security requirements for your use case and evaluate the connectivity technologies, in question, based on these requirements.

Note

Security evaluation of individual connectivity technologies is out of scope for this whitepaper.

Comparing LPWAN connectivity technologies

The following table provides a short overview of the key characteristics of LPWAN technologies covered in this whitepaper. Refer to [Appendix: Characteristics of LPWAN technologies](#) for a detailed description of each mentioned technology.

Table 1 – LPWAN connectivity technologies

Technology	LoRaWAN	NB-IoT (LTE Cat NB1)	LTE-M (LTE Cat M1)	Sigfox
Range and coverage	~5 km urban ~20 km rural	~1 km urban ~10 km rural	~1 km urban ~10 km rural	~10 km urban ~40 km rural
Support for device mobility	Yes	Limited	Yes	Yes
Uplink latency (order of magnitude)	Seconds	1.2–100 s	< 60 ms	Seconds
Data rate	0,3 (SF12)–5,47 (SF7) kilobit/s	< 66 kilobit/s (uplink) < 26 kilobit/s (downlink)	1 megabit/s	0.1 kilobit/s
Allow private networks	Yes	No	No	No (however, customers can deploy their own gateways)

Technology	LoRaWAN	NB-IoT (LTE Cat NB1)	LTE-M (LTE Cat M1)	Sigfox
Battery life and power consumption	Years (when operating as a Class A device)	Years (when using eDRX and PSM features)	Years (when using eDRX and PSM features)	Years
Maximum payload size	11–242 bytes, depending on regional regulations and spread factor	1,280 bytes recommended	1,280 bytes recommended	12 bytes (uplink) 8 bytes (downlink)
Support for indoor and underground coverage	Yes	Yes	Yes	Yes
Reachability	Depends on device class (high for Class A, low for Class C)	Depends on power saving model configuration (PSM/eDRX)	Depends on power saving model configuration (PSM/eDRX)	High (downlink transmission allowed only during 30 seconds after uplink)
Licensed spectrum	No	Yes	Yes	No
Maximum messages per day	Depends on regional regulations of duty cycle	Unlimited	Unlimited	140 per day (uplink) 4 per day (downlink)

Scenarios for implementing an IoT solution

A wireless connectivity technology is not a purpose by itself, but a means for achieving a business goal by implementing an IoT solution. When architecting and implementing an IoT solution, AWS suggests that you consider the following scenarios: device provisioning, device authentication, telemetry ingestion, device commands, firmware updates, and connectivity management. In this section, these scenarios will be defined, whereas they will be used in the following sections to structure the implementation guidance for individual LPWAN technologies.

- **Device provisioning** – This scenario refers to a mechanism that provides unique device identities and configuration data to a device. Device provisioning is also involved with ongoing maintenance, and eventual decommissioning, of devices over time.
- **Device authentication** – This scenario refers to mechanisms used to reliably verify the identity of an IoT device, because IoT devices communicate with your cloud application.
- **Telemetry ingestion** – An IoT device collects telemetry (for example, read-only data) and transmits this data to an IoT application for further processing. Examples of such data include sensor measurements or device health metrics.
- **Device commands** – An IoT application in the AWS Cloud must be capable of sending commands to an IoT device remotely. Examples of such commands can be a trigger for an actor connected to the device, a configuration update for the device or a trigger to initiate a firmware update.
- **Firmware updates** – A capability to remotely update firmware on IoT devices is critical in ensuring the security and adding new capabilities to your IoT solution.
- **Connectivity management** – This scenario refers to the capability of your device software to configure, establish, and manages a wireless connection.

Integration with AWS Partners

If a connection between the IoT device and [AWS IoT Core](#) endpoint is not secured, for example, through TLS, Datagram Transport Layer Security (DTLS), or virtual private network (VPN), an additional security consideration is necessary. For example, such a situation could arise if a software component in customer's AWS account acts as an intermediary, receives data from the IoT device, and forwards data to AWS IoT Core. To ingest messages between software components in customer's AWS account to AWS IoT Core, AWS recommends always using [AWS IoT Data Plane application programming interfaces \(APIs\)](#) authorized by IAM mechanisms.

Implementing LPWAN IoT solutions on AWS using NB-IoT and LTE-M

IoT solutions using NB-IoT and LTE-M technologies often use resource-constrained battery-operated devices. For such devices, use of connection-based protocols such as MQTT and HTTP might unnecessarily decrease battery life or increase implementation complexity.

As opposed to MQTT and HTTP, connectionless protocols such as Constrained Application Protocol (CoAP) or User Datagram Protocol (UDP) offer higher energy efficiency and can reduce implementation complexity on the IoT device. Because of these benefits, telemetry ingestion and device commands for resource-constrained battery-operated NB-IoT and LTE-M devices are often implemented using UDP and CoAP protocols. You will find a detailed description for various implementation approaches in the [Implementing LPWAN IoT solutions with UDP](#) and [Implementing LPWAN IoT solutions with CoAP](#) sections of this document.

When using LTE-M with a non-constrained device or a device without long battery lifetime requirements, also MQTT and HTTP protocols shall be considered. You will find further information about using MQTT and HTTP protocols in the [Implementing LPWAN IoT solutions with MQTT](#) section of this document.

The following sections introduce User Datagram Protocol (UDP) and Constrained Application Protocol (CoAP) and then provide guidelines for their implementation with AWS IoT.

Topics

- [Implementing LPWAN IoT solutions with UDP](#)
- [Implementing LPWAN IoT solutions with CoAP](#)
- [Implementing LPWAN IoT solutions with MQTT](#)
- [Managing cellular connectivity](#)

Implementing LPWAN IoT solutions with UDP

User Datagram Protocol (UDP) is a lightweight protocol based on a principle of a connectionless communication. UDP is based on the Internet Protocol (IP). UDP/IP is a protocol stack allowing an optimization of power consumption (for example, for use cases requiring longer device battery duration).

UDP, by itself, does not ensure confidentiality, integrity, or authenticity of information exchange. To secure communication with UDP, additional measures need to be taken. An example of such a measure is to use transport layer security protocols such as DTLS.

When implementing IoT solutions using UDP, further characteristics need to be considered. A transmission using UDP is not reliable, because UDP does not include acknowledgement or retransmission mechanisms. In rare cases, UDP messages can be duplicated (for example, by network hardware components such as routers).

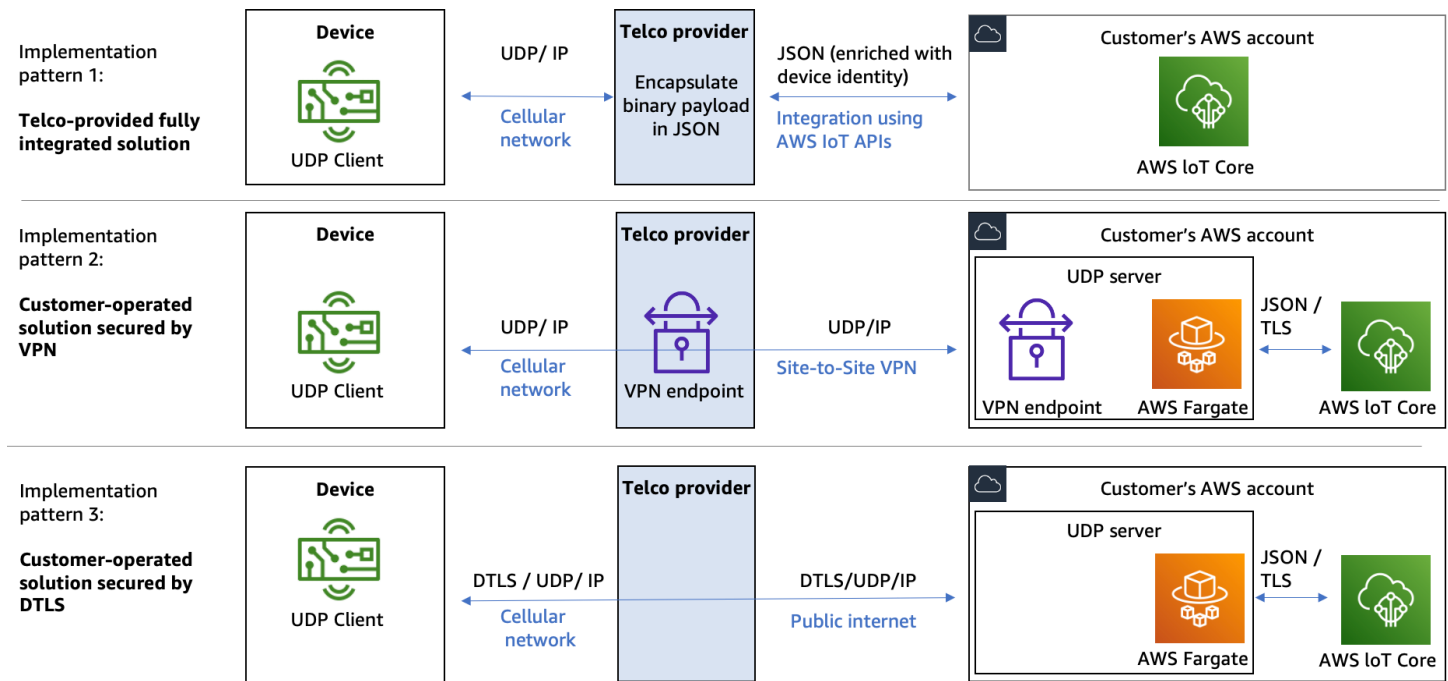
Transmission using UDP is not ordered. If several messages are sent to a recipient, no specific order of arrival can be guaranteed. If your IoT solutions requires guaranteed transmission of messages, deduplication, or message ordering when using UDP, appropriate mechanisms need to be developed on the application layer.

This whitepaper describes an approach on how to ingest UDP communication from a microcontroller unit (MCU) device to an AWS IoT Core MQTT topic. An important consideration when using UDP or CoAP/UDP is securing a communication in terms of implementing reliable authentication of the device in the cloud, confidentiality of communication between the device and the cloud, and integrity of exchanged data. Another consideration is to decide which capabilities to build and operate (such as, make approach) and which capabilities to source (such as, buy approach”).

There are three implementation patterns which provide different approaches regarding these considerations:

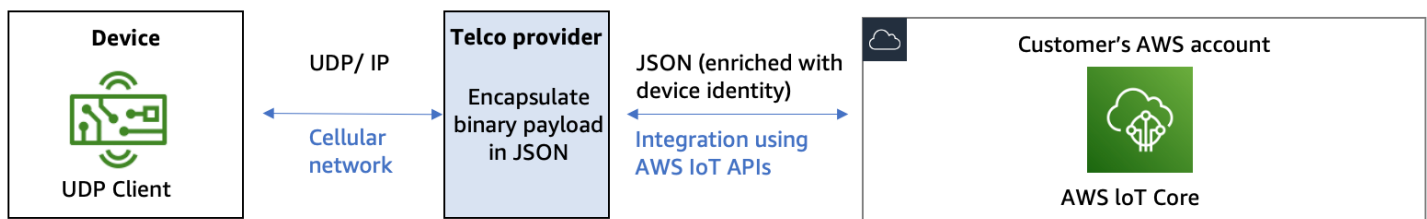
- Telecommunications (telco)-provided fully integrated solution
- Customer-operated solution secured by Telco-provided VPN
- Customer-operated solution secured by DTLS

The following figure provides an overview of these patterns:



Patterns for implementing IoT solutions with AWS using UDP protocol

Pattern 1: Telco-provided fully integrated solution



Architecture of a fully integrated solution provided by the telco

In this pattern, the telco provider is operating a UDP endpoint. The customer's devices target this endpoint for ingesting the payloads using UDP. Telco's software component then encapsulates the binary payload from the UDP message into a JSON format. Finally, the telco will ingest the payload into the customer's AWS account. A deployment of an [AWS CloudFormation](#) template into the customer's account is typically necessary for the integration between telco's infrastructure and customer's AWS account.

Device-side considerations

For the purpose of device authentication, international mobile equipment identity (IMEI) or international mobile subscriber identity (IMSI) numbers are typically used. The telecommunication

provider will offer an option to enrich this authentication data when ingesting messages into the customer's account.

For the purpose of telemetry ingestion using UDP, customers can use a UDP library available on their MCU. When using FreeRTOS, customers can refer to [Sending UDP Data examples](#). Before ingesting UDP data, customers need to configure a UDP endpoint and port supplied by their telecommunication provider.

To send data to the device for the purpose of command processing, the implementation varies depending on the telecommunication provider. When using FreeRTOS, customers can refer to [Receiving UDP Data examples](#).

Cloud-side considerations

Key considerations for an implementation on the cloud side are integration between telco and customer's account, payload binary decoding, and integration of decoded payload with other AWS services.

For an integration between telco and customer's account, telco will ingest the incoming payload from the device into customer's account AWS IoT Core. A possible implementation pattern for the ingestion is an invocation of an AWS IoT Core service API (for example, [Publish action of AWS IoT data API](#)) in the customer's account by the telco provider. To give telco provider access to AWS resources in your account, AWS recommends using IAM roles with external IDs to delegate access to your AWS resources to the telco's AWS account.

The payload binary decoding is necessary, because the telecommunication provider will encapsulate the device's binary UDP payload into a JSON message, without performing binary decoding. After the telco ingests the JSON message into the customer's account, binary decoding is performed. To implement binary decoding, customers can use the [AWS IoT SQL `aws\_lambda` function](#) in [AWS IoT rules](#).

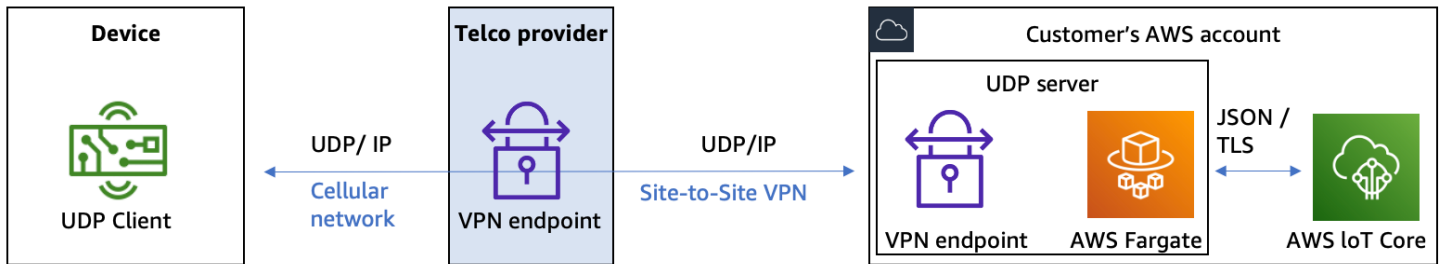
For integration of decoded payload with other AWS services, the recommended approach is to use [AWS IoT rules](#) and [AWS IoT rule actions](#).

Useful resources

Refer to the following resource for further details:

- [Automated Device Provisioning to AWS IoT Core Using 1NCE Global SIM](#)

Pattern 2: Customer-operated solution secured by Telco-provided VPN



Architecture of a customer-operated solution secured by VPN

In this pattern, the customer is operating a solution for ingesting UDP payloads for the customer's IoT devices. The solution can be built by the customer, the AWS Partner, or—if available—sourced from the telecommunication provider.

To secure confidentiality and integrity of the communication between the telecommunication provider and the customer's account, some telecommunication provider, a Site-to-Site VPN between the telecommunication provider and the customer's AWS account is established.

Cloud-side considerations

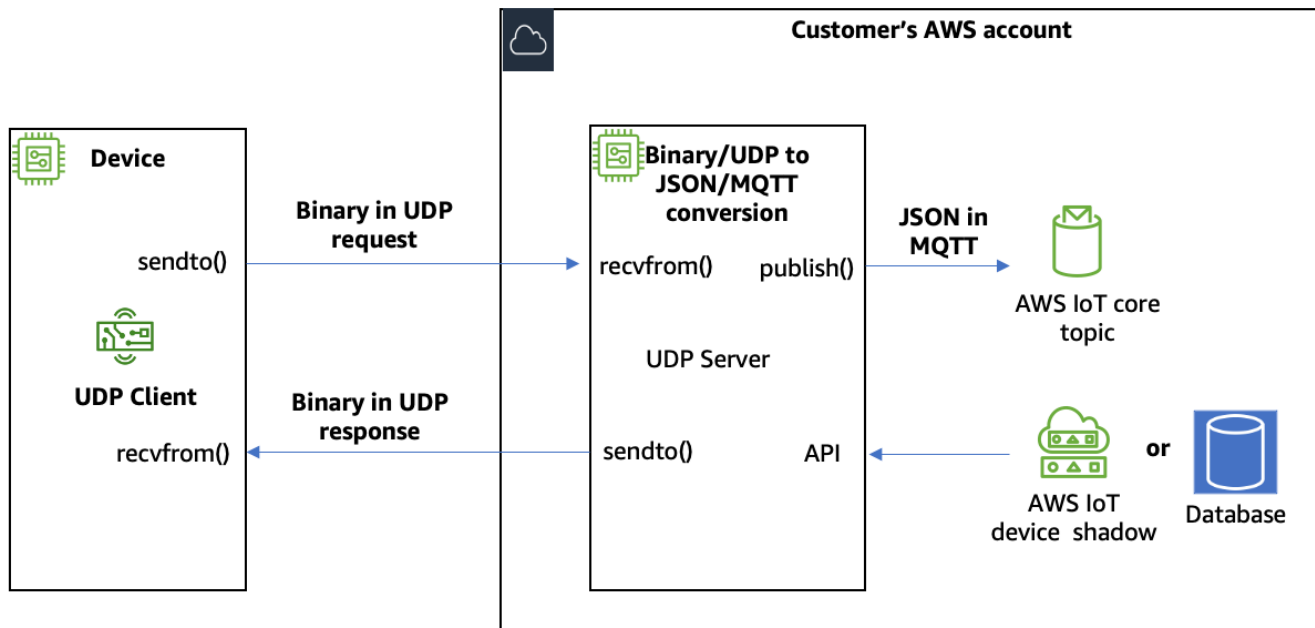
In this pattern, the customer is operating a solution for ingesting UDP payloads for the customer's IoT devices. The solution can be built by the customer, the AWS Partner or—if available—sourced from the telecommunication provider.

Key features of this solution are payload binary decoding, ingestion to AWS IoT Core, and integration with other AWS services. The payload binary decoding is necessary, because the devices will place a binary payload into UDP for energy efficiency reasons. During payload binary decoding, a transformation from a binary representation of payload into, for example, JSON takes place. For further processing, an ingestion to IoT Core [MQTT topic](#) or [AWS IoT rule](#) can be performed. After the payload arrives on IoT Core MQTT topic or AWS IoT rule, it can be simply integrated with more than [20 AWS services supported by AWS IoT rules](#).

To ingest messages to AWS IoT Core, the recommended approach is to use [AWS IoT Data plane APIs](#) authorized by [IAM](#) mechanisms.

The same software component can also support downlink communication to the devices. For example, this software component can be designed to attach downlink payload data to the UDP response to uplink message (piggybacking mechanism). The source of the downlink data can be

[AWS IoT Device Shadow](#) or a database such as [Amazon DynamoDB](#). The following figure illustrates this approach:



Example architecture for processing UDP datagrams with AWS

Device-side considerations

For device authentication, the customer can use DTLS protocol (outlined in [Pattern 3](#)).

For the purpose of telemetry ingestion using UDP, customers can use a UDP library available on their MCU. When using FreeRTOS, customers can refer to [Sending UDP Data examples](#). Before ingesting UDP data, customers need to configure a UDP endpoint and port pointing to the DNS record or [Elastic IP address](#) of the UDP server in their AWS account.

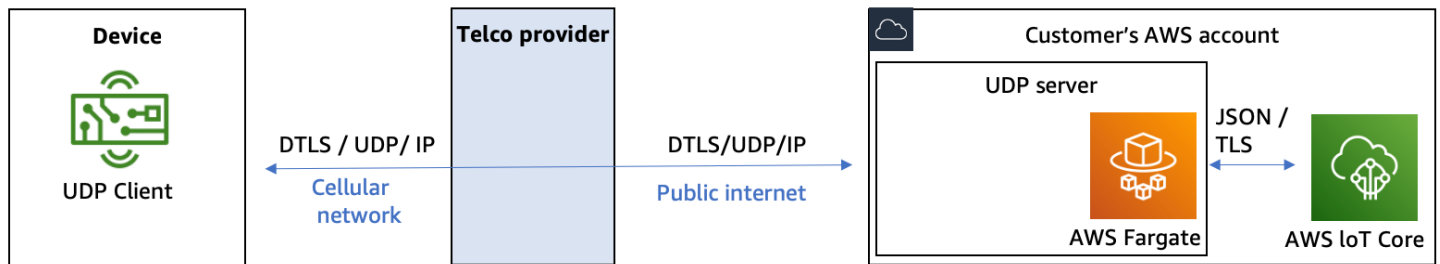
To send data to the device for the purpose of command processing, customers have multiple options. One possible option is to use the piggybacking mechanism as described in the *Cloud-side considerations* section of this document. That mechanism improves energy efficiency, but might introduce longer reachability times. Another option is to send data to device by addressing the devices by their internal IP addresses in the telecommunication provider's network. When using this option, devices should operate a UDP server to be capable to process incoming UDP. This approach reduces reachability times, but using this approach can increase power consumption on the device.

Useful resources

Refer to the following resource for further details:

- [Using a Telefonica Data Bridge to Connect Narrow Band IoT Devices to AWS IoT Core](#)

Pattern 3: Customer-operated solution secured by DTLS



Architecture of a customer-operated solution secured by DTLS

DTLS is a protocol which can be used to secure UDP-based communication in terms of device authentication, confidentiality, and integrity. The current version of DTLS 1.2 for use with UDP is defined in [RFC6347](#). When using DTLS, the payload of underlying UDP messages remains unchanged. Consequently, DTLS-based communication shows the same characteristics as UDP: best effort delivery without built-in acknowledgement and retry mechanisms, no guarantee for ordering of messages, and the possibility for duplicated messages. DTLS supports pre-shared secret, certificates and raw public keys as credentials. When using DTLS, a secure storage of pre-shared secret and certificates is an important precondition to avoid impersonation attacks.

Use of DTLS might have several benefits. First, the communication from the IoT devices to the customer's AWS account can be secured in terms of confidentiality and integrity without needing to rely on the telecommunication provider's VPC peering or VPN solutions described in [Pattern 2](#). This might simplify switching telecommunication providers or using multiple telecommunication providers. Second, DTLS provides an effective way of securing the device authentication. By associating the pre-shared secret or certificate of an IoT devices with the device identifier, the cloud application is able to verify the identity of the device.

However, when using DTLS, the aspects of increased energy consumption and credential management need to be considered. When using DTLS, additional operations need to be performed on the IoT device to encrypt the outgoing messages and decrypt the incoming messages. These operations will result in an increase of energy consumption compared to sending unencrypted UDP payloads as described in the other discussed patterns. Increased energy

consumption can reduce the battery lifetime of the IoT devices. Another aspect to consider is credential management (for example, an approach for provisioning, storing, and rotating the DTLS credentials on the IoT devices).

Device-side considerations

For device authentication, you can use DTLS.

Cloud-side considerations

In this pattern, the customer is operating a solution for ingesting DTLS-encrypted UDP payloads for the customer's IoT devices. The solution can be built by the customer or AWS Partner. A key component of this solution is software which receives incoming UDP messages, authenticates the device payload, performs decryption and decoding of message payload, and forwards the payload for further processing to AWS IoT Core [MQTT topic](#) or [AWS IoT rule](#). To ingest messages to AWS IoT Core we recommend to use [AWS IoT Data plane APIs](#) authorized by [IAM](#) mechanisms. After the payload arrives on IoT Core MQTT topic or AWS IoT rule, it can be simply integrated with more than [20 AWS services supported by AWS IoT rules](#).

Implementing LPWAN IoT solutions with CoAP

This section first provides an overview of the architectural considerations for an implementation of a CoAP-based IoT solution. It then describes typical implementation patterns that AWS customers can use to implement CoAP-based IoT solutions.

Introduction to CoAP

CoAP is an application layer protocol that was purpose built for constrained devices. It has low overhead and allows simple implementation on IoT devices. For example, the smallest size of the CoAP message is four bytes, if omitting tokens, options, and payload. Using CoAP, you can implement HTTP-like request/response patterns. Because CoAP is based on UDP, implementation using CoAP is more energy-efficient than when using MQTT or HTTP. You will find a specification of CoAP in [RFC 7252](#).

CoAP client and server

Depending on use case requirements, an IoT device can act as a CoAP client, a CoAP server, or both. Battery-operated LPWAN devices will act as a CoAP client for the purpose of reduction of energy consumption. A mainline-operated LPWAN device can also act as a CoAP server.

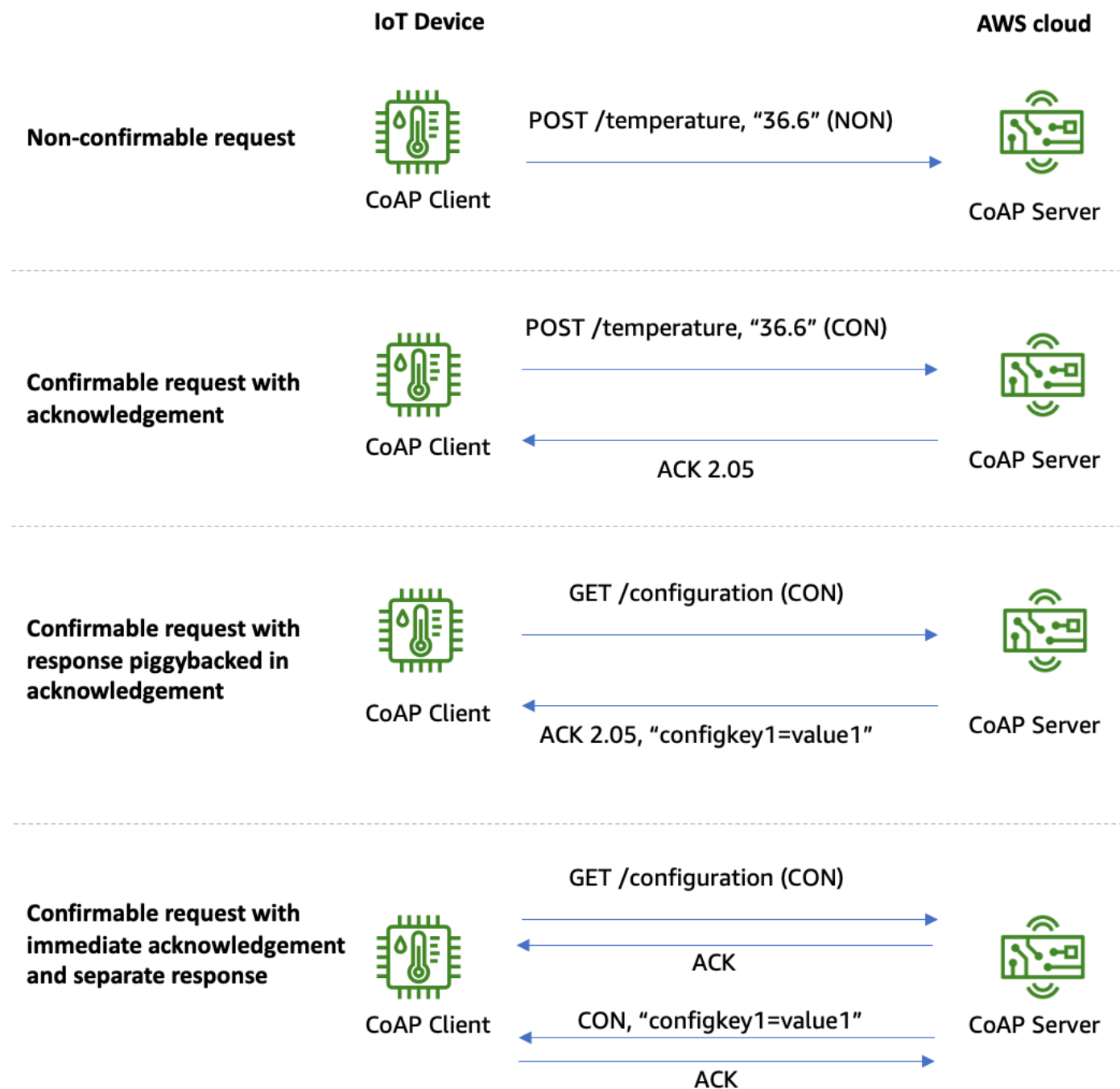
CoAP resource model

CoAP is based on a resource model. A *resource*, in the context of CoAP, is a logical container on the CoAP server. Each CoAP resource is mapped to a Unique Resource Identifier (URI). A CoAP resource URI consists of host, port, path, and query. An example CoAP URI is `coaps://host:port/path/to/resource?key=value`. To interact with resources, CoAP supports methods such as GET, PUT, POST, and DELETE with semantics and return codes similar (but not identical) to HTTP.

Confirmable and non-confirmable requests

CoAP supports both confirmable and non-confirmable requests. When using confirmable requests, the CoAP server must acknowledge the request with an acknowledgement in response. The response acknowledgement can optionally contain the payload sent by the server. Clients can use the confirmable messages to deal with the packet loss. If CoAP client does not receive an acknowledgement from the server, the CoAP client can perform a retry.

The following figure shows examples for common patterns of using both confirmable and non-confirmable requests. For the requests from the CoAP client to the CoAP server, the figure specifies a CoAP method (for example, POST), URI-path (for example, /temperature), type of message (CONFIRMABLE or NON-CONFIRMABLE), and—where applicable—payload (for example, "36.6"). For the responses from the CoAP server to the CoAP client, the figure specifies the response code (for example, 2.05 which means "Content") and—where applicable—payload.



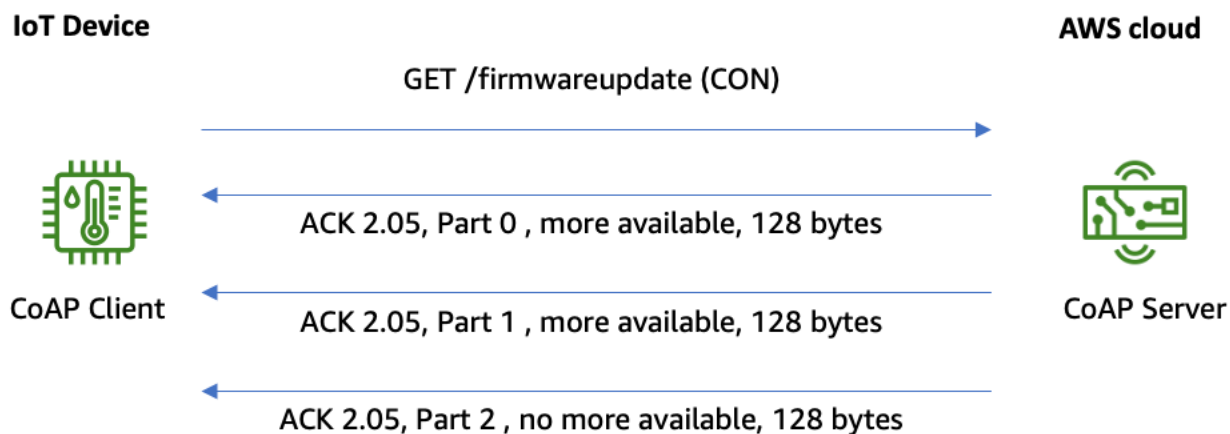
Types of CoAP requests

As mentioned previously, an IoT device can act as a CoAP client, CoAP server, or both. For the use case of telemetry ingestion from battery-operated LPWAN devices, the IoT device will act as a CoAP client.

Block-wise transfer mode

Normal CoAP messages are restricted in size and typically used for small payloads, such as measurements from the sensors or commands to the actuators. The size restriction results from factors such as maximum UDP datagram size or maximum MTU size enforced by cellular network operators. To enable applications to transfer larger amounts of data (for example, for the purpose of firmware updates), CoAP supports block-wise transfer mode as specified in [RFC 7959](#). Block-wise transfer mode is applicable to both CoAP requests and CoAP responses.

In the following example, a payload with the size of 384 bytes needs to be sent from the server to the client, over a connection limiting the maximum payload size per UDP datagram to 128 bytes. CoAP client performs a GET request, and CoAP server sends three blocks of 128 bytes each.



Example of CoAP block-wise transfer

Encryption in transit and device authentication

CoAP provides neither capability for encryption in transit, no dedicated features for device authentication. For the purpose of encryption in transit and device authentication, DTLS can be used. You will find more details on DTLS in [RFC6347](#).

Architectural considerations

When architecting an IoT solution based on CoAP, AWS recommends several fundamental considerations. First, consider whether your IoT device should take on the role of CoAP client, CoAP server, or both. Battery-powered IoT devices that use LPWAN connectivity should act as CoAP clients for energy efficiency. However, depending on the use case, your IoT device might also

need to operate as a CoAP server. In this case, it is important to note that not all implementation patterns described in the following section support IoT devices acting as CoAP servers.

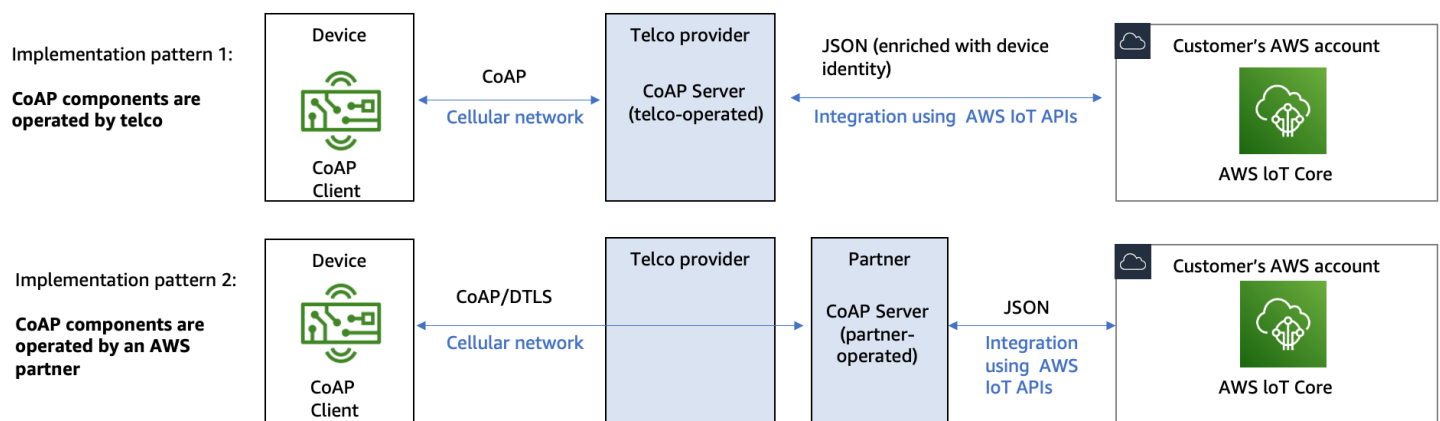
The second consideration is a definition of the CoAP features required for an implementation of your use case. AWS recommends that you consider both CoAP features which are required in the short term and CoAP features which may be required at a later point of time as your IoT solution evolves. This step is important because individual implementation approaches may differ depending on the subset of CoAP features they support. Examples of such CoAP features can be support for [GET/PUT/POST/DELETE methods](#), [block-wise transfer](#), [confirmable messages](#), [piggybacked response](#), and [separate response](#). Also, support for individual [CoAP options](#) such as [ETag](#) may vary depending on the chosen solution.

If you plan to use DTLS as a security layer for CoAP, an analysis of required DTLS features is also recommended. Examples for such features are support for authentication methods (for example, [No-Sec](#), [Pre-shared keys \(PSK\)](#), [Raw Public Key Certificates](#), [X.509 certificates](#)) and [Session resumption](#) with session ID and connection ID.

Implementation patterns

When implementing data ingestion and device commands with CoAP, there are three implementation patterns to consider. These patterns differ depending which entity operates the CoAP software components, such as CoAP server or CoAP client. These patterns are: CoAP software components are operated by telco, CoAP software components are operated by an AWS Partner, and CoAP software components are operated by the customers. In the latter case, two different variations shall be distinguished, depending on how the data in transit are secured. The data in transit can be secured by a VPN connection between telco and customer's account, or by using a communication protocol such as DTLS.

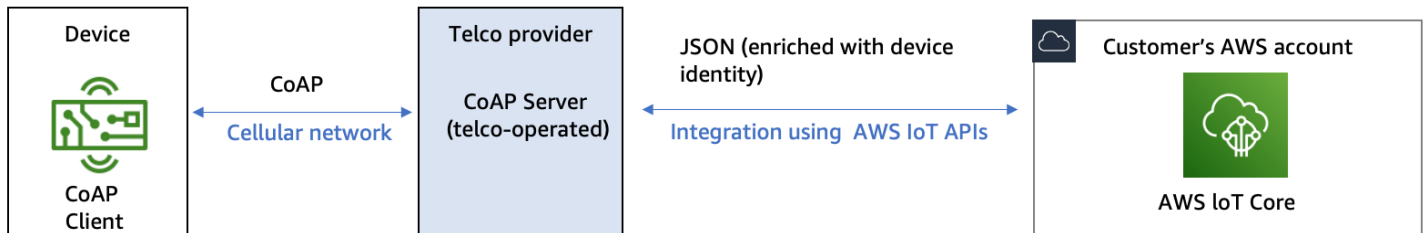
The following diagram illustrates three patterns, including two variations of the latter one:



Patterns for implementing IoT solutions with AWS using CoAP

The following sections contain a detailed description of each of these patterns.

Pattern 1: CoAP components are operated by telco



Architecture for CoAP server operated by telco

In this pattern, the telecom provider operates a CoAP endpoint, which acts as a CoAP server. Customers can configure the CoAP endpoint to integrate with the customer's AWS account. Each time the CoAP client on the IoT device sends a message to the CoAP server, the CoAP server will perform the steps of encapsulation, enrichment, and ingestion into the customer's account.

In the encapsulation and enrichment step, the CoAP server first encapsulates the binary payload into a JSON message, and then enriches this JSON message with additional useful information. An example of additional information is the International Mobile Subscriber Identity (IMSI) of the IoT device.

Note

The binary payload remains unchanged in this step. If necessary, binary decoding should be performed in the customer's AWS account.

In the ingestion step, the CoAP server forwards the JSON message to the AWS IoT Core service in the customer's AWS account. After the data arrive in AWS IoT Core message broker or AWS IoT Core rule engine, it can be further processed by a broad range of AWS services.

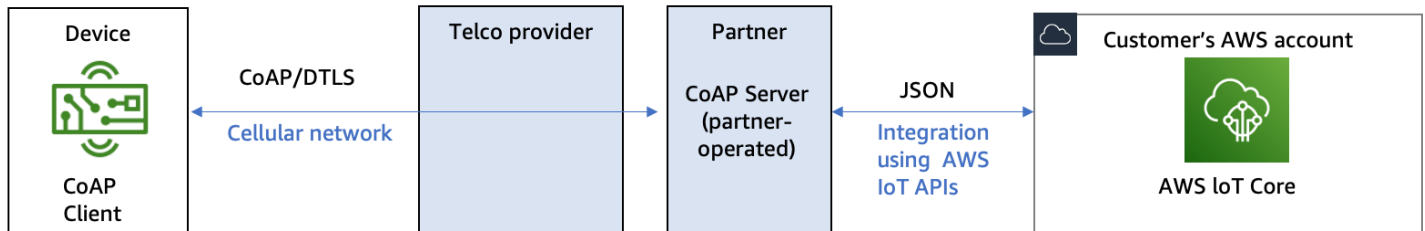
If you want to give telco provider access to AWS resources in your account, AWS recommends using IAM roles with external IDs to delegate access of your AWS resources to the telco AWS account.

Refer to the following resource for examples of AWS Partners supporting this pattern:

- [Automated Device Provisioning to AWS IoT Core Using 1NCE Global SIM](#)

- [Ericsson Cloud Connect: Making it easy for enterprises to securely connect cellular devices to Amazon Web Services](#)

Pattern 2: CoAP components are operated by an AWS Partner



Architecture for CoAP server operated by an AWS Partner

In this pattern, an AWS Partner operates a CoAP endpoint, which acts as a CoAP server. Customers can configure the CoAP endpoint to integrate with a customer's AWS account. Each time the CoAP client on the IoT device sends a message to the CoAP server, it will perform the steps of encapsulation and ingestion into the customer's account.

1. **Encapsulation step** – The CoAP server encapsulates the binary payload into a JSON message. Note that the binary payload remains unchanged in this step. If necessary, binary decoding should be performed in the customer's AWS account.
2. **Ingestion step** – The CoAP server forwards the JSON message to the AWS IoT Core service in the customer's AWS account. After the data arrive in the AWS IoT Core message broker or AWS IoT Core rule engine, the data can be further processed by a broad range of AWS services.

If you want to give an AWS Partner access to AWS resources in your account, AWS recommends using IAM roles with external IDs to delegate access from your AWS resources to the AWS Partner account.

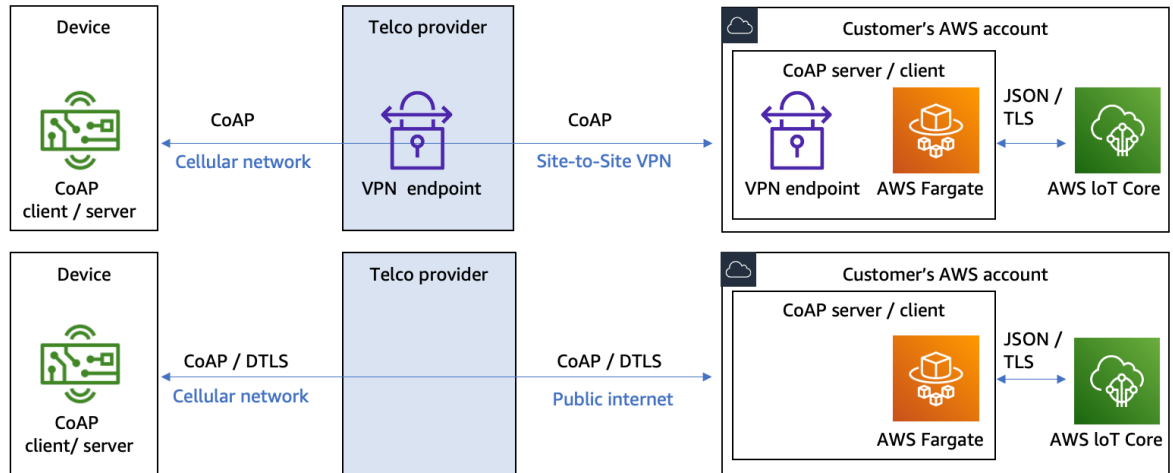
Refer to the following resources for examples of AWS Partners supporting this pattern:

- [IoTerop Nebraska in AWS Marketplace](#)
- [Ericsson Cloud Connect: Making it easy for enterprises to securely connect cellular devices to Amazon Web Services](#)

Pattern 3: CoAP components are operated by the customer

Implementation pattern 3:
CoAP server/client
operated by the customer

**Variant A: secured via
VPN or VPC peering**

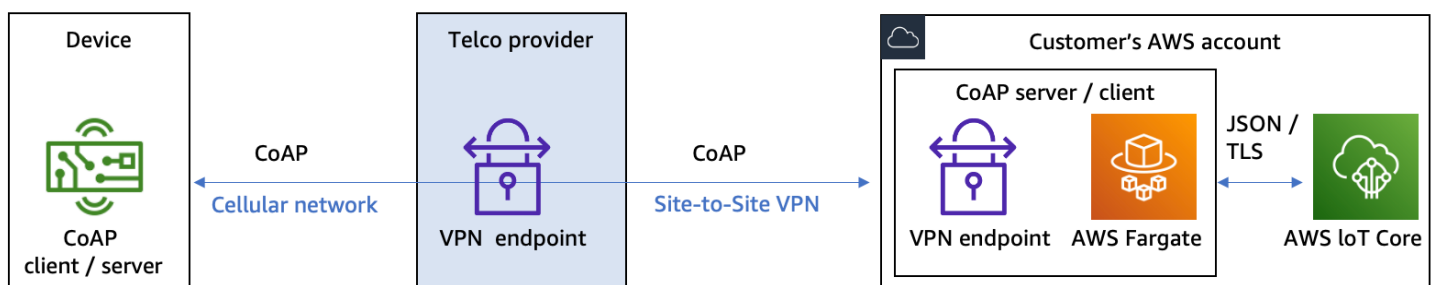


Architecture for CoAP server and client operated by customer

In this pattern, the AWS customer operates a CoAP endpoint in their AWS account. Depending on the use case requirements, the CoAP endpoint can operate as CoAP server, CoAP client, or both. The CoAP endpoint can handle forwarding the incoming CoAP messages to AWS IoT Core, and sending CoAP messages to the IoT devices. To ingest messages to AWS IoT Core, AWS recommends using [AWS IoT data plane APIs](#) authorized by IAM mechanisms.

When using this pattern, a mechanism used for securing data in transit needs to be defined. This mechanism will protect the complete path of communication from the IoT device to the customer's AWS account. The first consideration is to protect the data in transit between the IoT device and the telco provider's infrastructure. The second consideration is to protect the data in transit between the telco provider's infrastructure and the customer's AWS account. There are two possible variants for securing data in transit:

Securing data in transit with a VPN connection between telco and customer's AWS account

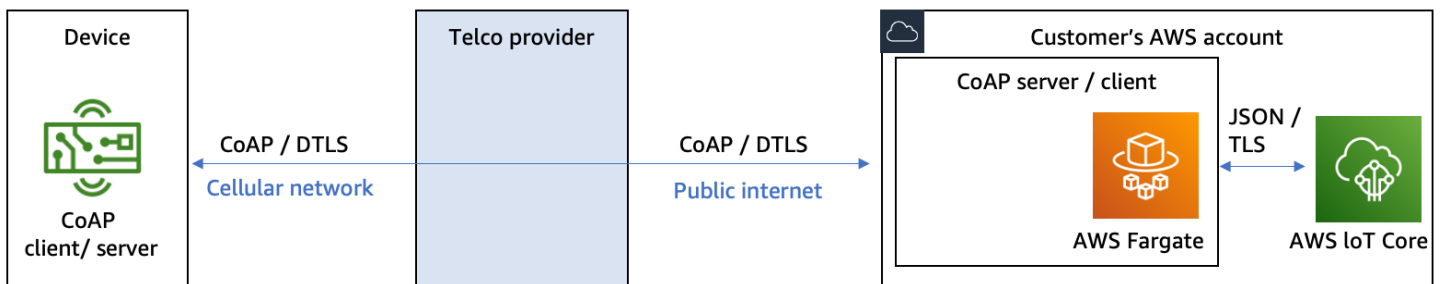


Using VPN to secure data in transit

In this variant, customers rely on the telco provider's infrastructure for protection data in transit between the IoT device and the telco provider's infrastructure. No protocol-level security is used, and CoAP is used in NoSec mode. To protect data in transit between telco provider's infrastructure and the customer's AWS account, a VPN connection is established between the telco provider and the customer's AWS account.

For further information on using VPN, review the [Site-to-Site VPN](#) documentation. For examples of AWS Partners supporting this pattern, refer to [Using a Telefonica Data Bridge to Connect Narrow Band IoT Devices to AWS IoT Core](#).

Securing data in transit with protocols as DTLS



Using DTLS to secure data in transit

In this variant, customers use DTLS protocol as a mechanism for infrastructure for protection data in transit between the IoT device and telco provider's infrastructure, and between telco provider's infrastructure and the customer's AWS account.

Implementing LPWAN IoT solutions with MQTT

Using UDP and CoAP on MCU devices is recommended to improve power efficiency and ensure longer battery life. However, for use cases with a non-constrained devices or devices without long battery lifetime requirements, also MQTT and HTTP protocols should be considered, as described in the following sections.

Device telemetry ingestion

To ingest telemetry with MQTT and HTTP, customers can use FreeRTOS libraries and OS-independent AWS IoT Device SDK for Embedded C. For customers who use FreeRTOS, AWS recommends using FreeRTOS libraries ingest telemetry through JSON (using [coreJSON](#) library),

HTTP(S) (using [coreHTTP](#) library), MQTT (using [coreMQTT](#) library), and TCP sockets (using [Secure Sockets](#) library). If using other operating system than FreeRTOS, customers can also use [AWS IoT Device SDK for Embedded C](#), which also provides [coreJSON](#), [coreHTTP](#), and [coreMQTT](#) libraries.

Device commands

To send commands to the device, customers can use [AWS IoT Device Shadow](#). Interaction with AWS IoT Device Shadow is supported in FreeRTOS (refer to [AWS IoT Device Shadow library](#)) and OS-independent [AWS IoT Device SDK for Embedded C](#).

Firmware updates

Device provisioning relates to the application workflow that provides unique identity and configuration data to the device. The provisioning layer is also involved with ongoing maintenance and eventual decommissioning of devices over time.

Firmware and over-the-air (OTA) updates without human intervention is critical for security, scalability, and delivering new capabilities to the IoT devices.

Customers using FreeRTOS can use [OTA Agent library](#) to perform firmware upgrades on the FreeRTOS devices. If using operating systems other than FreeRTOS, customers can use [AWS IoT Over-the-air Update library](#) included in the [AWS IoT Device SDK for Embedded C](#).

Implementing device provisioning

When you use AWS IoT fleet provisioning, AWS IoT can generate and securely deliver device certificates and private keys to your devices when they connect to AWS IoT for the first time. You can find the guidelines for implementing fleet provisioning on your MCU device in the [Fleet Provisioning Documentation](#).

Managing cellular connectivity

Customers can use FreeRTOS connectivity libraries for Wi-Fi and cellular to enable their applications to interact with the communication interfaces on the MCUs. The [FreeRTOS Cellular library](#) exposes the capability of a few popular cellular modems through a uniform API. That API hides the complexity of AT commands, and exposes a socket-like interface to C programmers. The [FreeRTOS Wi-Fi library](#) abstracts port-specific Wi-Fi implementations into a common API. Using this API, applications can communicate with their lower-level wireless stack through a common interface.

 Note

Applications are typically not using these libraries directly, but are using higher level libraries (for example, [coreMQTT](#) for ingesting MQTT payloads).

Implementing LPWAN IoT solutions on AWS using LoRaWAN

LoRa Alliance specifies the LoRaWAN protocol in documents called LoRaWAN specifications. Although LoRaWAN operates in the unlicensed radio spectrum, manufacturers and operators of LoRaWAN devices still have to fulfill various country-specific regulations. To account for this, LoRa Alliance also publishes regional parameters documents specify a common denominator per country as a recommendation (but not specification) for device manufacturers and operators.

Table 2 – LoRaWAN specifications

Version	Year	Major changes		Download specification		
1.0	2015	Initial release				
1.0.1	2016	Clarifications / corrections				
1.0.2	2016	Separation of regional parameters				
1.1	2017	Roaming added, security features added			1.1	
1.0.3	2018	Class B added as normative			1.0.3	

Version	Year	Major changes		Download specification		
1.0.4	2020	Class B improved, security improved, clarifications			1.0.4	

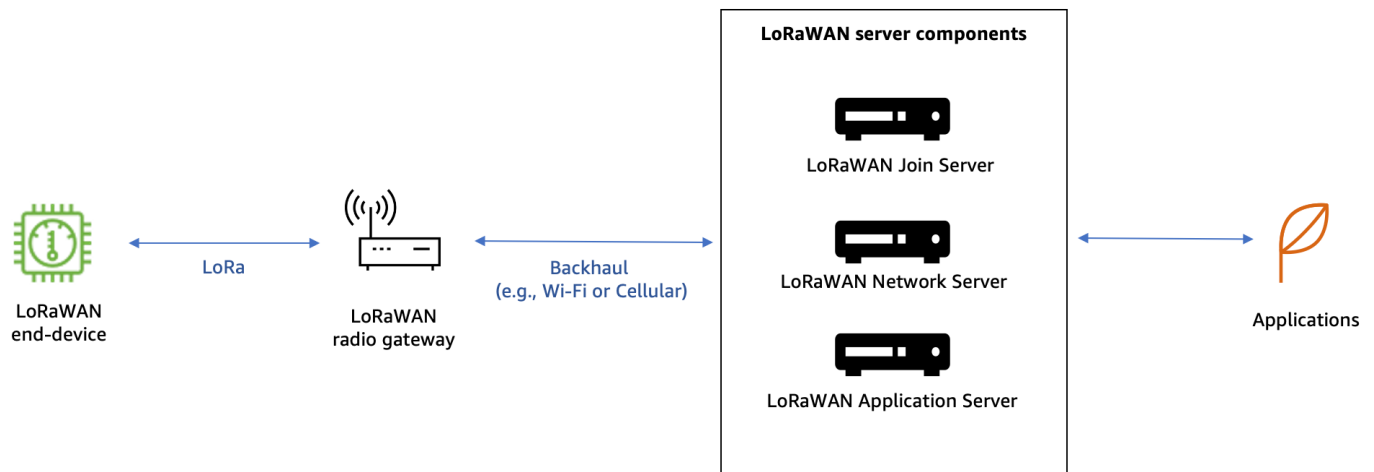
The LoRaWAN Regional Parameters document describes the recommended configurations and parameters for different regions worldwide. They are separated from the protocol specification to allow the addition of new regions without impacting the LoRaWAN specification. You can download the most recent version (RP2-1.0.2) of the [LoRaWAN Regional Parameters](#).

Topics

- [LoRaWAN network architecture](#)
- [LoRaWAN networks](#)
- [Building a private LoRaWAN network](#)
- [Public LoRaWAN networks](#)

LoRaWAN network architecture

The following figure shows a simplified representation of a LoRaWAN network architecture:



LoRaWAN network architecture (simplified)

LoRaWAN radio gateway

The radio gateway forwards all received LoRaWAN radio packets to the network server that is connected through an IP backbone. Its role is to decode uplink radio packets from the air and forward them unprocessed to the network server. For downlinks, the radio gateway forwards the packet transmission requests coming from the LoRaWAN network server without any interpretation of the packet payload.

LoRaWAN network server

The network server shuts down the LoRaWAN layer for the LoRaWAN end devices connected to the network. Some functions of the network server include:

- Deduplicating uplink messages from the devices. Deduplication is necessary in case several gateways within reach of the device receive and forward the message to the LoRaWAN network server.
- Forwarding uplink application payloads to the appropriate application servers.
- Queueing of downlink payloads.
- Interacting with the join server during the join procedure.

LoRaWAN application server

The application server handles all of the application layer payloads from the end devices. It also generates all the application layer downlink payloads towards the connected end devices.

LoRaWAN join server

The join server manages the OTA end device activation process.

LoRaWAN networks

When architecting an AWS solution based on LoRaWAN connectivity, AWS customers can implement their IoT solutions using public or private LoRaWAN networks.

Public LoRaWAN networks

A public LoRaWAN network is used by several customers sharing the same network infrastructure such as LoRaWAN gateways of LoRaWAN network servers. The public LoRaWAN network is operated by a commercial company, although community-owned public LoRaWAN networks exist.

The network operator designs and maintains necessary gateway infrastructure and LoRaWAN network software components. Some of the providers allow customers to run LoRaWAN application server on the customer's premises. The LoRaWAN network operator also offers a capability to integrate with customer's applications.

Customers of the public LoRaWAN network are entitled to connect their LoRaWAN devices to the network's network servers through LoRaWAN gateways. After the LoRaWAN device is connected, the data can be exchanged between the LoRaWAN devices and integrates customer's applications. Using public LoRaWAN networks offers advantages such as reduction of implementation and operation effort and coverage of larger geographic areas.

Public LoRaWAN networks can be operated as commercial networks and community networks. When using a commercial public LoRaWAN network, the network operator operates and maintains the LoRaWAN gateways and LoRaWAN server components. When using a community public LoRaWAN network, the LoRaWAN gateways are operated and maintained by the members of the community, and the LoRaWAN server components are operated and maintained by the community network operator.

Using public LoRaWAN networks offers advantages such as reduction of implementation and operation effort, and coverage of larger geographic areas. However, public LoRaWAN networks

have limitations. First, customers of commercial public LoRaWAN networks typically have no influence to the positioning and selection of LoRaWAN gateways, which can limit indoor penetration range and make implementation of indoor use cases challenging or impossible. Further considerations to evaluate when selecting if public LoRaWAN network is a right choice are security, and regulatory or cost efficiency requirements. Especially when using community public LoRaWAN networks, additional limitations such as fair-use-policies may apply.

Private LoRaWAN networks

A private LoRaWAN network is used by a single enterprise to consume the service themselves or offer the service to their end customers. The enterprise performs hardware selection for and location of LoRaWAN gateways, and controls the network coverage. The enterprise has also a capability to finetune the configuration of a LoRaWAN network for the requirements of the use case.

A decision for a private deployment mode of LoRaWAN framework is driven by a combination of coverage, mobility, costs, and regulatory and privacy considerations.

From a coverage standpoint, private LoRaWAN networks can be used in the areas and environments where public LoRaWAN networks do not provide sufficient coverage. Examples include rural areas as well as indoor and underground environments.

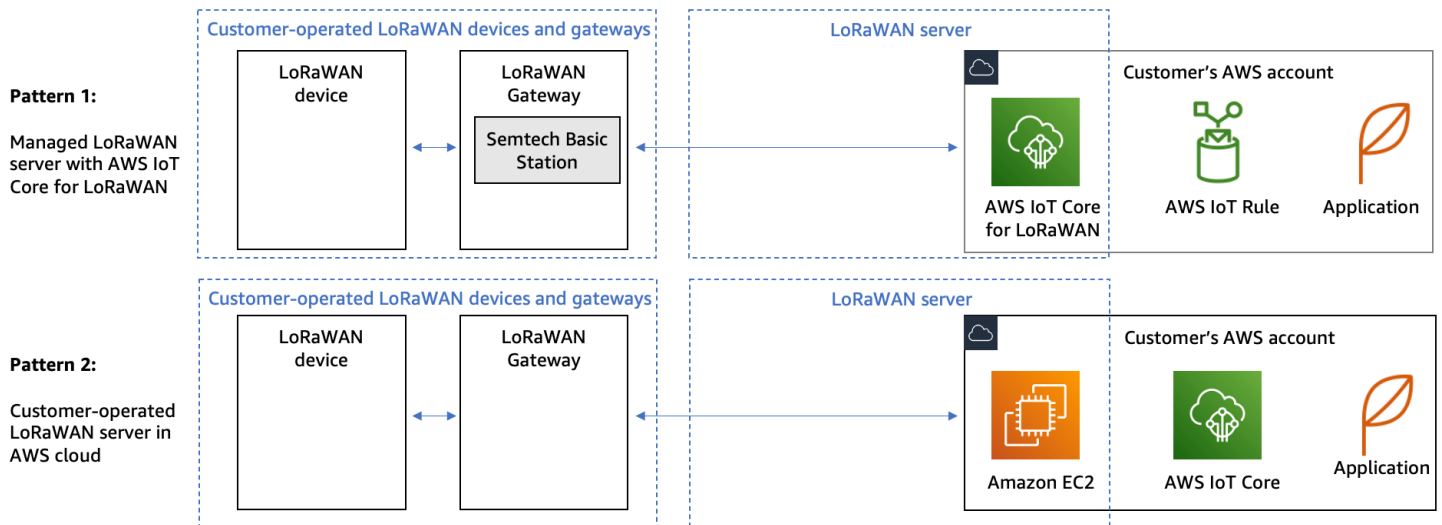
From a mobility standpoint, private LoRaWAN networks are especially beneficial for use cases with either stationary devices or devices having limited range of movement. If your use case requires LoRaWAN network coverage in wide geographical areas (for example, country-wide), a public LoRaWAN network might be a better solution.

The cost, regulatory, and privacy considerations are not further detailed in this whitepaper.

Building a private LoRaWAN network

When deploying and operating a private LoRaWAN network, there are three major components to consider: LoRaWAN devices, LoRaWAN gateways, and LoRaWAN software components. Examples of components are LoRaWAN network server, join server, and application server as described in the *LoRaWAN network architecture* section of this document.

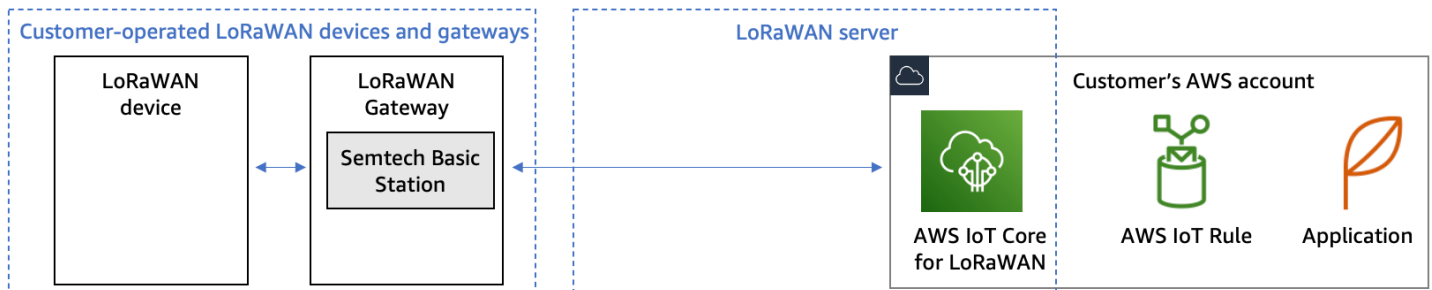
The following figure provides an overview of various deployment patterns for private LoRaWAN networks:



Patterns for deployment of private LoRaWAN networks

The following sections will describe each of the individual options.

Managed LoRaWAN server with AWS IoT Core for LoRaWAN



AWS IoT Core for LoRaWAN architecture

The first pattern is to use AWS IoT Core for LoRaWAN. AWS IoT Core for LoRaWAN is a fully managed feature that enables customers to connect wireless devices that use low-power, long-range wide-area network (LoRaWAN) protocol with AWS Cloud. Using AWS IoT Core, customers can now set up a private LoRaWAN network by connecting their own LoRaWAN devices and gateways to AWS Cloud, without developing or operating a LoRaWAN network server, join server, or application server.

AWS IoT Core for LoRaWAN supports open-source gateway-network server protocol software called LoRa Basics Station. The IoT Core for LoRaWAN partner gateway qualification program enables customers to source pre-qualified LoRaWAN gateways. You can get an overview of pre-qualified LoRaWAN gateways by using [AWS Partner Device Catalog](#).

Customers can also buy any off-the-shelf sensor or actuator compliant with LoRaWAN specification 1.0.x or 1.1 and connect it to AWS IoT Core. This creates a plug and play experience that reduces the device on-boarding friction. As an introduction to AWS IoT Core for LoRaWAN, AWS recommends the following resources:

- [AWS IoT Core for LoRaWAN](#) overview
- [Introducing AWS IoT Core for LoRaWAN](#)
- [AWS IoT Core for LoRaWAN workshop](#)
- [Sample solutions for AWS IoT Core for LoRaWAN](#) with examples for an implementation of binary decoding, device and gateway monitoring, downlink transmission, telemetry dashboarding, and device provisioning automation

Device authentication

IoT Core for LoRaWAN supports both approved methods of device activation as specified by LoRa Alliance: OTA and activation by personalization.

Telemetry ingestion

LoRaWAN devices transmit binary encoded data, increasing transmission efficiency and improving battery lifetime. However, as the data arrive in the cloud, many use cases require a structured format. Transforming the binary data into JSON, for example, enables filtering and enrichment using [AWS IoT SQL](#) and acting on the data using [AWS IoT rule actions](#).

Refer to the code sample [AWS IoT Core for LoRaWAN - deployable reference architecture for binary decoding with examples](#) to receive guidelines on implementing and deploying binary decoders for your LoRaWAN devices.

Device commands

Because LoRaWAN protocol supports a bidirectional communication, you can send messages to your LoRaWAN devices. AWS IoT Core for LoRaWAN provides an API [SendDataToWirelessDevice](#) and related SDK functions (for example, [AWS SDK for Python](#), [AWS SDK for Java](#), and [AWS SDK for Javascript](#)).

Refer to the code sample [Send a downlink payload to a LoRaWAN device](#) to receive implementation guidelines for sending downlink payloads.

Device provisioning

AWS IoT Core for LoRaWAN simplifies device provisioning by offering APIs to provision and manage LoRaWAN gateways and devices. Refer to the [developer guide](#) and [API documentation](#) for details. You can also access [guidelines for automation of provisioning tasks](#).

Gateway firmware updates

For LoRaWAN gateway firmware update guidance, please refer to the qualified gateway documentation. You can find an example of such guidelines for one specific gateway vendor using [Update gateway firmware using CUPS service with AWS IoT Core for LoRaWAN](#).

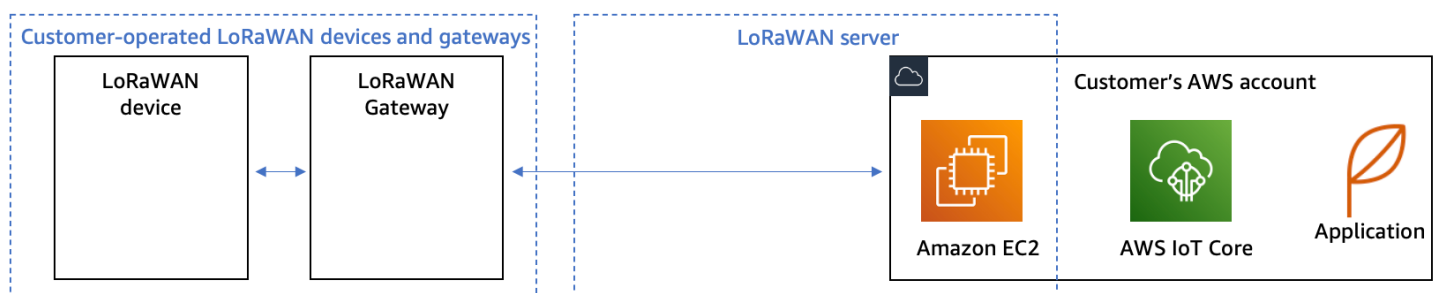
OTA firmware updates

With AWS IoT Core for LoRaWAN's OTA firmware updates, you can deploy new firmware images or delta images to a single device or a group of devices, verify the authenticity and integrity of new firmware after it's deployed to devices, and monitor the progress of a deployment and debug issues in case of a failed deployment. You can find more information on FUOTA with AWS IoT Core for LoRaWAN in the [Firmware Updates Over-The-Air \(FUOTA\) for AWS IoT Core for LoRaWAN devices](#) section of the developer documentation.

Multicast

With AWS IoT Core for LoRaWAN, you can send a downlink payload to multiple devices by sending data to a single multicast address, which is then distributed to an entire group of recipient devices. You can find more information on multicast with AWS IoT Core for LoRaWAN in the [Create multicast groups to send a downlink payload to multiple devices](#) section of the developer documentation.

Customer-operated LoRaWAN network server in AWS Cloud



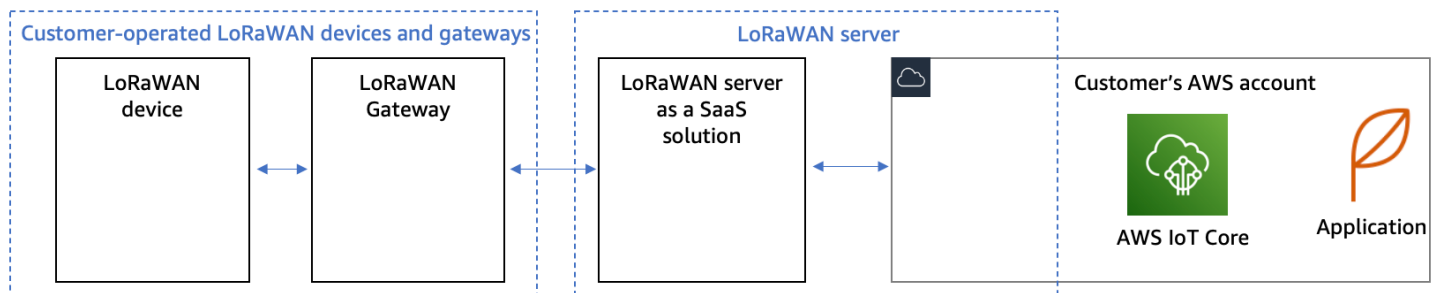
LoRaWAN server provided by AWS Partner, operated by customer

In this pattern, customers deploy and operate LoRaWAN server components in their own AWS account. Examples of LoRaWAN server components are LoRaWAN network server, join server, and application server. When using AWS Partner solutions, the deployment of necessary software components is simplified by using AWS CloudFormation templates provided by AWS Partners.

You can find further information for implementing a customer-operated LoRaWAN server using these resources:

- [LoRaWAN customer-operated solutions in AWS marketplace](#)
- [Connecting Your LoRaWAN Devices from The Things Stack to AWS IoT Core](#)

LoRaWAN server as an AWS Partner software as a service (SaaS) solution



LoRaWAN server provided and operated by an AWS Partner

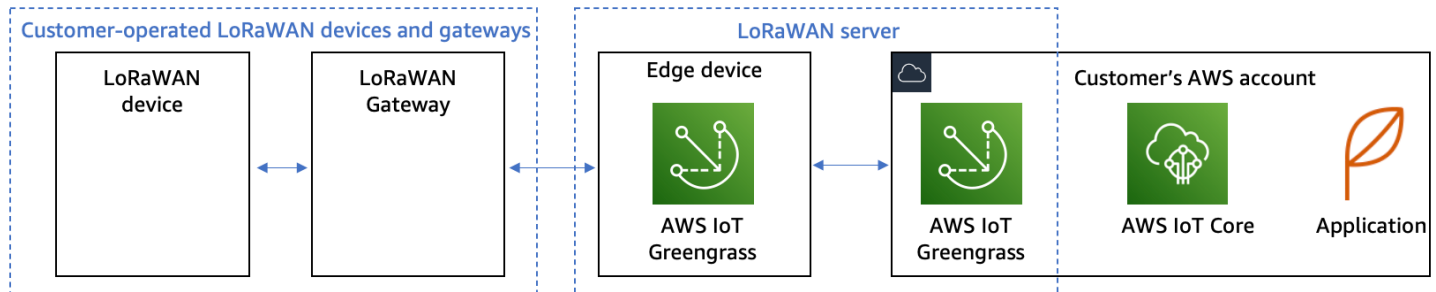
The next pattern uses a LoRaWAN network server of an AWS Partner as an SaaS solution. In this pattern, the AWS Partner is responsible for operating LoRaWAN components such as LoRaWAN network server, join server, and application server.

To enable a customer's application to act upon the telemetry from LoRaWAN devices and send commands to LoRaWAN devices, the SaaS solution providers offer integration capabilities. An example of integration is a feature enabling customers to forward the uplink messages from their LoRaWAN devices to AWS IoT Core service in their AWS account.

After the payload arrives on IoT Core MQTT topic or AWS IoT rule, it can be integrated with more than [20 AWS services supported by AWS IoT rules](#).

Visit AWS Marketplace to find examples of AWS Partner [LoRaWAN SaaS solutions](#).

Customer-operated LoRaWAN server on the edge device



LoRaWAN server operated by customer on the edge device

In this pattern, customers source, deploy, and operate LoRaWAN gateways and LoRaWAN server components. The LoRaWAN server integrates with AWS IoT Core, enabling cloud applications to process data from and to send commands to LoRaWAN devices. Using this pattern results in higher development and operational efforts compared to the patterns previously described.

One of the challenges when implementing this pattern is a need to remotely provision, update, and configure software components of LoRaWAN server. [AWS IoT Greengrass](#) can be used to support faster implementation and simplified operation of these tasks. In particular, the following capabilities of AWS IoT Greengrass will be considered: deploying and updating LoRaWAN server components, managing sensitive configuration, and integrating with AWS IoT Core.

Deploying and updating LoRaWAN server components

Customers can use AWS IoT Greengrass components mechanism to deploy LoRaWAN server components. For example, by using AWS IoT Greengrass [Docker application manager component](#) it is possible to download Docker images from public image registries or private repository in Amazon Elastic Container Registry.

Managing sensitive information

A LoRaWAN server configuration contains sensitive information. Customers can use AWS IoT Greengrass [secret manager component](#) to securely use credentials such as passwords on AWS IoT Greengrass core devices.

Integrating with AWS IoT Core

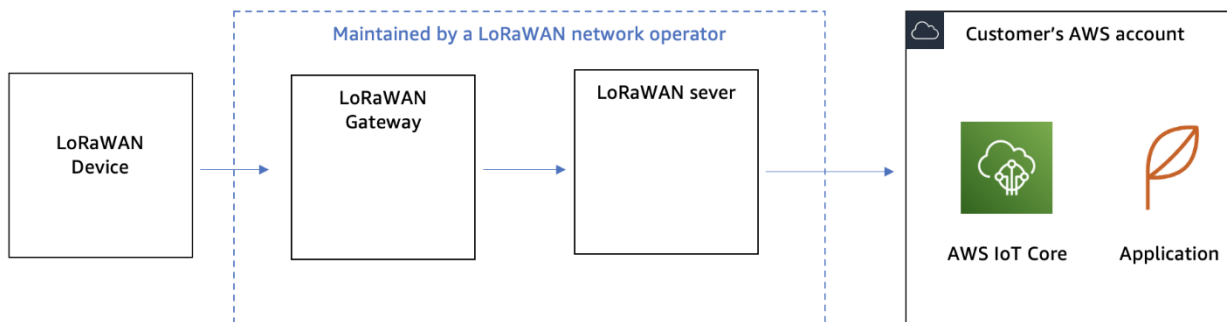
Integration between LoRaWAN server and AWS IoT allows users to forward LoRaWAN device payloads to cloud applications, and send commands from cloud application to LoRaWAN devices.

To implement this integration, customers can use AWS IoT Greengrass [AWS IoT Core Messaging IPC service](#).

For more information for implementing a customer-operated LoRaWAN server on the edge device, refer to [AWS IoT Greengrass V2 Community Component - The Things Stack LoRaWAN](#).

Public LoRaWAN networks

When using a public LoRaWAN network, the LoRaWAN network operator is responsible for management and maintenance of both LoRaWAN gateways and LoRaWAN server components, as outlined in the following figure:



Example architecture for using a public LoRaWAN network with AWS

Most operators of public LoRaWAN networks offer integration capabilities with AWS IoT. Refer to the documentation for the respective LoRaWAN network, or contact the LoRaWAN network operator for additional information.

For examples implementing IoT solutions with public LoRaWAN networks, refer to:

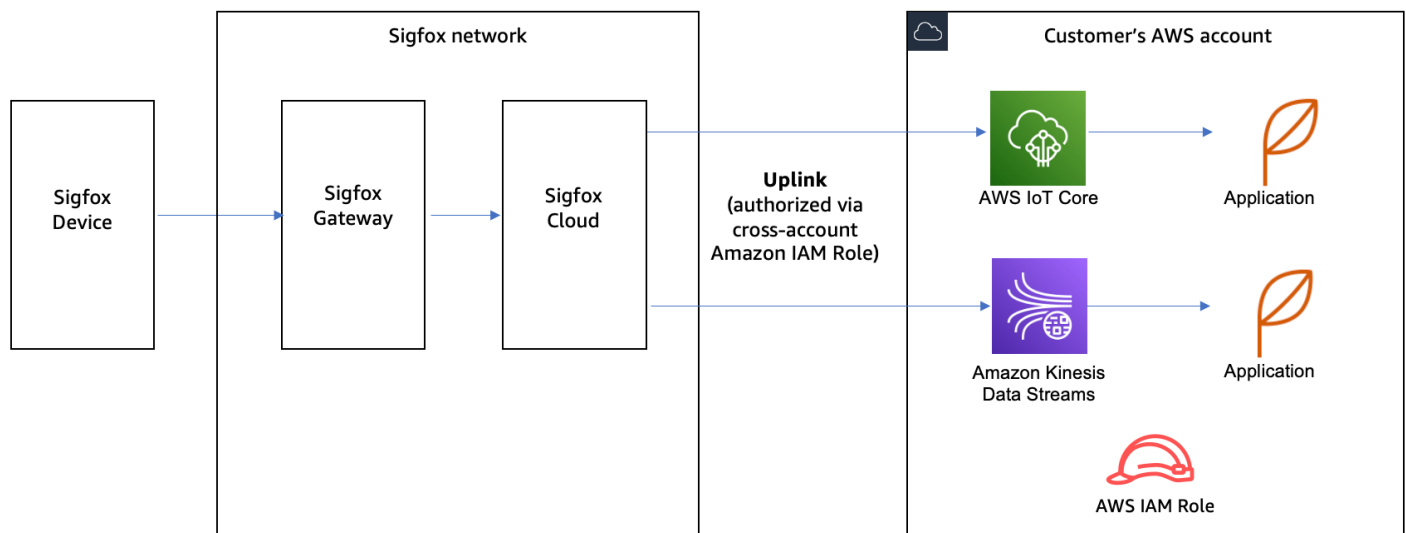
- [Connecting Your LoRaWAN Devices from The Things Stack to AWS IoT Core](#)
- [How to Connect Your LoRaWAN Devices to AWS IoT Core Using Activity ThingPark](#)
- [Connect your devices to AWS IoT using LoRaWAN](#)

Implementing LPWAN IoT solutions on AWS using Sigfox

When implementing an IoT solution with Sigfox, AWS customers can process uplink data from their Sigfox devices (for example, sensor telemetry) and send downlink data to their Sigfox devices (for example, device configuration change). However, the implementation approaches for uplink and downlink vary significantly. Therefore, both approaches will be described separately in the following sections.

Implementing uplink from Sigfox devices to AWS Cloud

The following diagram provides an overview of an IoT solution for processing uplink data from the Sigfox devices:



Example architecture for processing uplink data from Sigfox devices

Sigfox Cloud supports integration of the uplink payloads with AWS IoT Core and Amazon Kinesis Data Streams. When using integration with AWS IoT Core, customer's applications can access the uplink payloads by using [AWS IoT rules](#), or by subscribing to an [MQTT topic of AWS IoT Core](#) message broker. When using integration with Amazon Kinesis Data Streams, customer's application can access the uplink payloads by [reading data from Amazon Kinesis Data Streams](#).

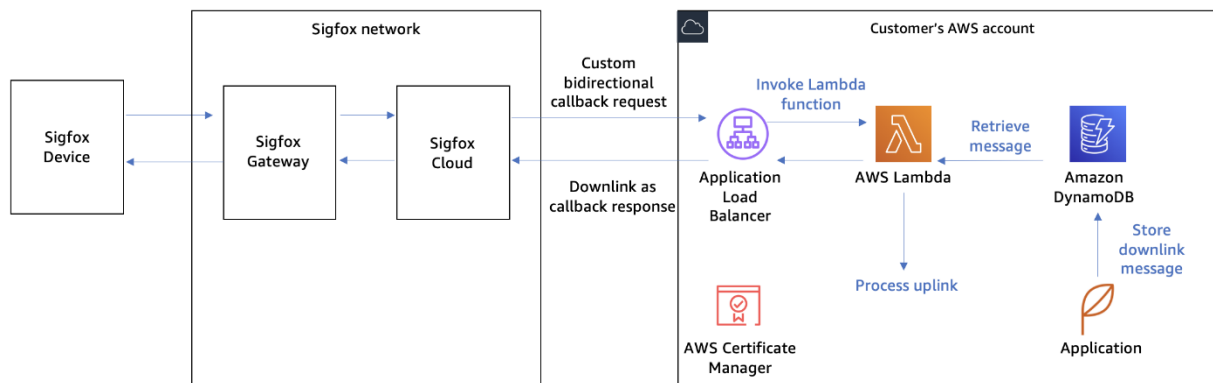
To implement this solution, AWS customers first create an IAM role with an external ID in their AWS account. That IAM role provides Sigfox Cloud permission to ingest messages either to a specific topic of AWS IoT Core message broker, or into a specific Amazon Kinesis Data Stream. After the IAM role with external ID is created, the customer can use Sigfox Cloud to configure the uplink ingestion

in their AWS account. For detailed information on implementing uplink from Sigfox devices, refer to [Connect your devices to AWS IoT using the Sigfox network](#).

Implementing downlink from AWS Cloud to Sigfox devices

The Sigfox network limits the number of downlink messages per day to four. To save energy, Sigfox devices are required to listen for incoming data for only 30 seconds after an uplink transmission. After 30 seconds, the Sigfox device changes into an energy-efficient mode and is not able to receive further incoming messages.

Because of this energy-saving feature, AWS Cloud applications require a mechanism to buffer the downlink data until the next uplink message arrives. After the next uplink message arrives, the buffered downlink message can be sent to the Sigfox device. To implement this approach, customers can use an example architecture as described in the following figure:



Example architecture for sending downlink data to Sigfox devices

First, AWS customers configure a custom bidirectional callback in Sigfox Cloud. The callback specifies a REST API to be invoked each time an uplink message is sent. The REST API request will forward an uplink message to AWS Cloud. The REST API response can optionally contain a downlink payload. If the downlink payload is available in callback response, the Sigfox Cloud will send it to the Sigfox device.

To implement a REST API endpoint, AWS customers can use an Application Load Balancer with a TLS certificate managed by the AWS Certificate Manager service. The Application Load Balancer can specify an AWS Lambda function as a target. Each time the REST API is invoked, the Lambda function will check if new downlink messages for the Sigfox devices are available and return a downlink message payload as a response. Amazon DynamoDB can be used by the customer's applications to buffer the downlink message for the Sigfox device.

Conclusion

Implementing LPWAN connectivity technology for an IoT solution can be a complex challenge. Each of the available LPWAN technologies offers its own unique set of features, but also has limitations that should be considered. This whitepaper outlined the trade-offs between the features of LPWAN technologies. AWS customers have a wide range of viable options for implementing their LPWAN solutions with NB-IoT, LTE-M, LoRaWAN, and Sigfox.

To support architectural decisions, this whitepaper provided an overview and detailed description of the most common architectural patterns. For NB-IoT and LTE-M, implementation using UDP, CoAP, and MQTT was described. For LoRaWAN, implementation patterns for both public and private LoRaWAN networks was covered. For Sigfox, implementation recommendations for both uplink and downlink communication was described.

Contributors

Contributors to this document include:

- Andrei Svirida (Sr. Specialist IoT Solutions Architect)
- Adrian Valenzuela (Sr. IoT GTM Specialist)
- Ali Benfattoum (Sr. Specialist IoT Solutions Architect)
- Ashu Joshi (Global Head, Enterprise Solutions)
- Greg Breen (Specialist IoT Solutions Architect)
- Gaurav Gupta (Principal Partner Solution Architect, IoT)
- Jan Borch (Principal IoT Specialist Solution Architect)
- Rodrigo Merino Gutierrez (EMEA IoT Solutions Architect Leader)

Further reading

Consider the following documents when architecting your IoT solutions on AWS:

- [AWS Architecture Center](#)
- [IoT Lens - AWS Well-Architected Framework](#)
- [IoT Lens Checklist](#)

Document history

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

Change	Description	Date
Initial publication	Whitepaper first published.	December 17, 2021

Note

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

Appendix: Characteristics of LPWAN technologies

The [Comparing LPWAN connectivity technologies](#) section of this document provided a brief overview of the selected characteristics of individual LPWAN technologies. This appendix goes deeper and provides additional depth. The purpose of this section is to support a decision if and how a specific LPWAN technology is appropriate for your use case.

LPWAN connectivity technologies

- [NB-IoT](#)
- [LTE-M](#)
- [LoRaWAN](#)
- [Sigfox](#)

NB-IoT

Narrowband Internet of Thing (NB-IoT) is an LPWAN technology developed by 3rd Generation Partnership Project (3GPP). NB-IoT standard LTE Cat NB1 was first specified as part of 3GPP Release 13 and then updated as LTE Cat NB2 as part of 3GPP Release 14. For simplicity, this whitepaper will use NB-IoT synonymous with LTE Cat NB1.

The design principles of NB-IoT are long device battery life, low device complexity, support for massive number of devices, and support for high coverage to reach devices in wide areas as well as in challenging locations. When using NB-IoT, devices communicate by IP protocol, although non-IP based communication is technically possible.

NB-IoT is a wireless cellular network technology. A *cell* is a geographical area in which the IoT device can communicate over radio with the transceiver station. The transceiver station is called a *base station* or *cell tower*.

Range and coverage

NB-IoT transmission range, in terms of distances between cell tower and the IoT device, is approximately one km in urban areas, and approximately 10 km in rural areas. NB-IoT supports indoor coverage.

Data rate

For NB-IoT, peak data rate is approximately 26 Kbps in downlink and approximately 66 Kbps in uplink when using multi-tone uplink mode, and approximately 20 Kbps when using single-tone uplink mode.

Mobility

NB-IoT is frequently used in applications and use cases with stationary assets (for example, smart metering or smart parking). LTE-M technologies such as LTE Cat M1 are more frequently used for use cases with mobile assets. This is mostly because LTE Cat NB1 technology has limited support for handover between cells. Without handover support, energy-consuming detach and reattach procedures, and related intermittent connectivity are necessary on each cell change.

Latency

When using NB-IoT, latency is from 1.6 seconds to 10 seconds.

Battery life

A battery can last several years without replacement, given appropriate engineering and configuration of the IoT device. For example, reduction of uplink frequency and using battery saving features, Power Saving Mode (PSM) and Extended Discontinuous Reception Mode (eDRX), can help increase energy efficiency.

By using PSM, device engineers can set a device in an energy-efficient extended power saving model for up to 14 days. During extended sleep, the device remains registered in the network, but network is not paging the device. The device can wake up when it needs to send data, and is reachable for downlink for a brief time interval after each data transfer. Because the device remains registered in the network, it does not need to perform an energy-consuming reattach procedure after waking up. By not paging the device during extended sleep, additional energy saving can be achieved.

By using eDRX, device software engineers can adjust device low-power sleep time, which is defined by the network and is often approximately 10 seconds by default. By using eDRX, this wakeup period can be extended to up to approximately 175 minutes. By using eDRX, device software engineers can adjust a tradeoff between energy efficiency and device reaction times. For example, if the application for smart street lights sets eDRX interval of approximately 2.7 min, it can take up to 2.7 minutes until the command from the cloud application to turn on the lights arrives at the device.

Although eDRX does not provide the same level of energy efficiency as PSM, it offers significantly shorter wake-up times than PSM. Also note that both support and upper limits for PSM and eDRX depend on network operator and used device. AWS suggests you contact your network operators and device vendors for further information.

Spectrum licensing

NB-IoT uses licensed spectrum. 3GPP Release 13 defines 14 frequency bands for NB-IoT, with four additional bands added in 3GPP Release 14, and four more bands added in 3GPP Release 15.

Payload size

LTE-M supports payloads up to 1,280 bytes when using IP protocol. However, the exact maximum payload size depends on connectivity module and mobile network used, and chosen approach for integration with your cloud application (for example, by IPsec). Because of these dependencies, AWS recommends that you consult both your connectivity module manufacturer and network operator.

Further considerations

Although out of scope for this whitepaper, other important considerations for LTE-M are security, device, and subscription cost and carrier roaming agreements.

LTE-M

LTE-M is an LPWAN technology developed by 3rd Generation Partnership Project (3GPP). LTE-M standard LTE Cat M1 was first specified as part of 3GPP Release 13, and then updated as standard LTE Cat M2 as part of 3GPP Release 14. For simplicity, this whitepaper will further use LTE-M synonymous with LTE Cat M1.

When using LTE-M, devices communicate by IP protocol, although non-IP based communication is technically possible.

LTE-M compared to NB-IoT

Although LTE-M has many similar characteristics as NB-IoT, there are differences. Compared to NB-IoT, LTE-M tends to offer higher data rates, lower latency, better energy efficiency for big payloads, and is better suited for mobile applications than NB-IoT. LTE-M devices tend to have higher power consumption for small payloads and higher end device costs compared to NB-IoT devices.

Range and coverage

LTE-M transmission range in terms of maximum distance between the tower and the IoT device is approximately one km in urban areas, and approximately 10 km in rural areas. LTE-M supports indoor coverage.

Data rate

For LTE Cat M1, peak data rate is 1 MBps both in uplink and downlink.

Mobility

LTE-M technologies can be used for use cases with fixed and mobile assets (for example, mobile fleet management or mobile asset tracking), assuming availability of network coverage on device location. Compared to NB-IoT, LTE-M is better suited for use cases requiring asset mobility. The first reason is handover support. When an LTE-M device is moving from one cell tower to another, it can seamlessly switch the connection to the new cell tower while staying attached to the network. Avoiding a need to detach from an old cell tower and perform a reattachment to the new cell tower results in increased energy efficiency and continuous connectivity.

Battery life

A battery can last several years without replacement, given appropriate engineering and configuration of the IoT device. For example, reduction of uplink frequency and using battery saving features, PSM and eDRX, can be helpful to increase energy efficiency.

The principles behind PSM and eDRX for LTE-M are similar to those for NB-IoT as described in the previous section. However, note that differences in the implementation between LTE-M and NB-IoT apply, for example for eDRX cycle durations and other parameters.

Latency

When using LTE Cat M1, typical latency is between 10 and 20 ms.

Spectrum licensing

LTE-M uses licensed spectrum. 3GPP Release 13 defines 19 frequency bands for LTE-M, with two additional bands added in 3GPP Release 14.

Payload size

The exact maximum payload size when using NB-IoT depends on used connectivity module, used mobile network, and chosen approach for integration with your cloud application. For example, a message packet (including IP header) greater than 1,280 bytes risks being truncated and undeliverable. Because of these dependencies, AWS recommends consulting both your connectivity module manufacturer and network operator in that matter.

Further considerations

Although out of scope for this whitepaper, other important considerations for LTE-M are security, device and subscription cost, and carrier roaming agreements.

LoRaWAN

The LoRaWAN is an LPWAN standard specified by the LoRa Alliance. The LoRa Alliance describes LoRaWAN as “a Low Power, Wide Area networking protocol designed to wirelessly connect battery operated ‘things’ to the internet in regional, national or global networks, and targets key Internet of Things (IoT) requirements such as bi-directional communication, end-to-end security, mobility and localization services”.

When using LoRaWAN, both public and private deployment modes are supported. In a public deployment mode, IoT devices can access a public LoRaWAN network, operated by the LoRaWAN network provider and shared by several customers. In private deployment mode, you can build and operate your own LoRaWAN network.

Range and coverage

LoRaWAN specification is based on LoRa radio frequency modulation technique, which is patented by Semtech Corporation. LoRa is operated in license free ISM (industrial, scientific, and medical) frequency bands, for example 433 MHz/868 MHz in Europe, or 915 MHz in US. LoRa uses modulation technique based on Chirp Spread Spectrum, allowing transmission range of up to 20 km in line of sight, and up to five km in urban areas. The actual range depends on environment (for example, existence of obstacle for the radio signal), transmission configuration (for example, transmission power), and positioning of gateways.

LoRa radio frequency modulation technique is suitable for use cases that require indoor and underground coverage, because the radio signal can penetrate physical barriers such as walls or ceilings.

Data rate

When using LoRaWAN, the data rate is limited to 0,3 Kbit/s to 27 Kbit/s. The effective maximum data rate depends the spread factor used by the device.

Mobility

LoRaWAN can be used for use cases requiring stationary and moving devices, assuming availability of network coverage on the device location. When using LoRaWAN protocol, a device payload will be received by all the gateways within reach. Each of gateways receiving payload from the device will forward this payload to the LoRaWAN network server. The LoRaWAN network server will perform a deduplication of the message.

Battery life

A battery can last several years without replacement. However, note that long battery duration is only possible when operating LoRaWAN devices as Class A and B (for example, devices that are sleeping most of the time).

Spectrum licensing

LoRaWAN operates in a license-free spectrum; however, region-specific regulations apply. For example, duty cycle regulations in the EU define the maximum air time for a LoRaWAN device.

Application payload size

LoRaWAN maximum application payload size depends both on the LoRaWAN frequency plan (for example, EU868 or US915) and the LoRaWAN data rate (for example, 0 or 7) the device is using for the transmission. Consult the [LoRaWAN Regional Parameters](#) for a detailed description of the maximum payload size for each country. Note that the term *LoRaWAN data rate*, in this and the next section, is LoRaWAN-specific and is used to describe a combination of Spreading Factor (for example, SF7) and bandwidth (125 Khz).

An application payload size of 11 bytes is supported in most frequency plans (for example, EU868 and US915) with any of the LoRaWAN data rates. If your application requires larger payload sizes, a thorough research is recommended. For example, according to the chapter 2.4.6 “EU863-870 Maximum payloads size” of [RP2-1.0.2 LoRaWAN Regional Parameters](#), the maximum payload size when using Data Rate 0 is 51 bytes, assuming the fields of proficiency testing (FOpt) field is used.

Latency and reachability

When using LoRaWAN, typical latency is in the order of magnitude of seconds. Reachability depends on device class.

Class A devices are in energy-efficient power saving mode most of the time and listen for downlink messages only for a short period of time after transmitting. Because of this energy-saving mechanism they also have the longest reachability. These devices are battery-powered sensors.

Class B devices can receive downlink messages in scheduled downlink slots, so that they have improved reachability compared to Class A devices. Class B devices are battery-powered actuators.

Class C devices continuously listen for the incoming messages. Because of this they have the better (shorter) reachability in order of magnitude of seconds.

Further considerations

Although out of scope for this whitepaper, other important considerations for LoRaWAN are security, device, and operation and subscription costs and local regulatory requirements.

Sigfox

Sigfox S.A. is a network operator for a LPWAN connectivity product, which is also called Sigfox. Sigfox is operated as a public network, with coverage in a subset of countries.

Range and coverage

The range of Sigfox devices is up 40 km outdoors, and 10 km in urban areas.

Mobility

Sigfox can be used for use cases requiring stationary and moving devices, assuming the Sigfox network coverage is available on the device locations. When using Sigfox, a device payload will be received by all the gateways within reach. Each of gateways receiving payload from the device will forward this payload to the Sigfox. The Sigfox network will perform a deduplication of the message.

Battery life

A battery of a Sigfox device can last several years without replacement.

Spectrum licensing

Sigfox devices are operated in license-free ISM (industrial, scientific, and medical) frequency bands in ranges from 862 MHz to 928 MHz. The exact frequency band used by the device depends on the geographical zone.

Payload size

Sigfox devices can send payloads of up to 12 bytes size in uplink, and 8 bytes in downlink in a single message. A Sigfox device is allowed to transmit up to 140 uplink messages and receive up to four downlink messages per day.

Latency

When using Sigfox, typical latency is in an order of magnitude of seconds. Reachability depends on the frequency of uplink transmissions. This dependency occurs because Sigfox devices are required to listen for incoming data for only 30 seconds after an uplink transmission. After 30 seconds, the Sigfox device changes into an energy-efficient mode and is not able to receive further incoming messages. For example, if you configure your Sigfox device to send uplink messages once a day, a downlink latency can be up to 24 hours.

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.

© 2021 Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS Glossary Reference*.