SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – III – Internet of Things – SCSA5301

COMMUNICATION AND CONNECTIVE TECHNOLOGIES

IoT Communication Model - Cloud computing in IoT - IoT in cloud architecture - Logging on to cloud - Selecting and Creating cloud service - cloud based IoT platforms - IBM Watson - Google cloud.

1. IoT Communication Model

The term of internet of things (IoT) communication offered by Internet protocols . Many of the devices often called as smart objects operated by humans as components in buildings or vehicles, or are spread out in the environment.

Communication types

1. Device-to-Device Communications
2. Device-to-Cloud Communications

Device-to-Device Communications:

The device-to-device communication model represents two or more devices that directly connect and communicate between one another, rather than through an intermediary application server. These devices communicate over many types of networks, including IP networks or the Internet. Often,however these devices use protocols like Bluetooth-Wave, or ZigBee to establish direct device-to-device communications.

Attack Surfaces on Device to Device Communication:

- Credentials stealing from the firmware
- Sensitive information disclosure
- No proper updating mechanism of firmware
- DoS Attacks

Buffer-overflow

attacks

A **buffer** is a temporary area for data storage. When more data gets placed by a program or system process, the extra data overflows. It causes some of that data

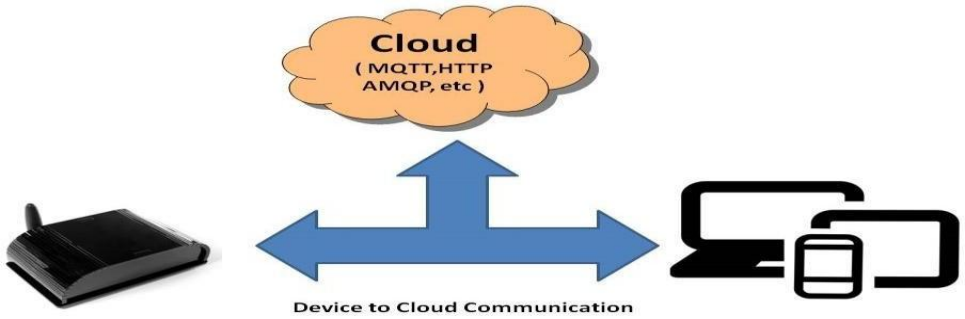
to leak out into other buffers, which can corrupt or overwrite whatever data they were holding. In a **buffer- overflow attack,** the extra data sometimes holds specific instructions for actions intended by a hacker or malicious user; for example, the data could trigger a response that damages files, changes data or unveils private information.

Best Practices for securing Device to Device Communication:

- Evaluate hardware components, firmware, software, communications protocols
- Try to Make the signed Firmware, software and hash your binaries.
- Implement the machine to machine authentication securely.
- Get the feedback from the clients to improve the device security levels

Device-to-Cloud Communications

In a device to cloud communication model, the IoT device connects directly to an Internet cloud service like an application service provider to exchange data and control message traffic. This approach frequently takes advantage of existing communications mechanisms like traditional wired



Ethernet or Wi-Fi connections to establish a connection between the device and the IP network, which ultimately connects to the cloud service.

Figure 1: Device to Cloud Communication

Device to Cloud protocols . Below table 1 explains the details about the protocols :

Protocols	AMQP	MQTT	XMPP	CoAP
Transport	TCP/IP	TCP/IP	TCP/IP	UDP/IP
Message pattern	Publish — Subscribe	Publish — Subscribe	Point — Point Publish – Subscribe	Request – Response

The Advanced Message Queuing Protocol (AMQP) and the MQTT Protocol are often seen as mutually exclusive choices, especially in the Internet of Things (IoT). AMQP is a general-purpose message transfer protocol suitable for a broad range of messaging- middleware infrastructures, and also for peer-to-peer data transfer. It's a symmetric and bi- directional protocol that allows either party on an existing connection to initiate links and transfers, and has rich extensibility and annotation features at practically all layers. Both protocols share that they can be tunneled over Web Sockets and therefore function well in environments that restrict traffic to communication over TCP port 443 (HTTPS).

MQTT Concepts

In MQTT, all messages are published into a shared topic space at the broker level. A

–topic in MQTT is a filter condition on the consolidated message stream that runs through the MQTT broker from all publishers. Publishing topics have a hierarchical structure (a path through topic space) and filters can be expressed as direct matching conditions (topic name and filter expression must match), or the filter can use wild-cards for single or multiple path segments.

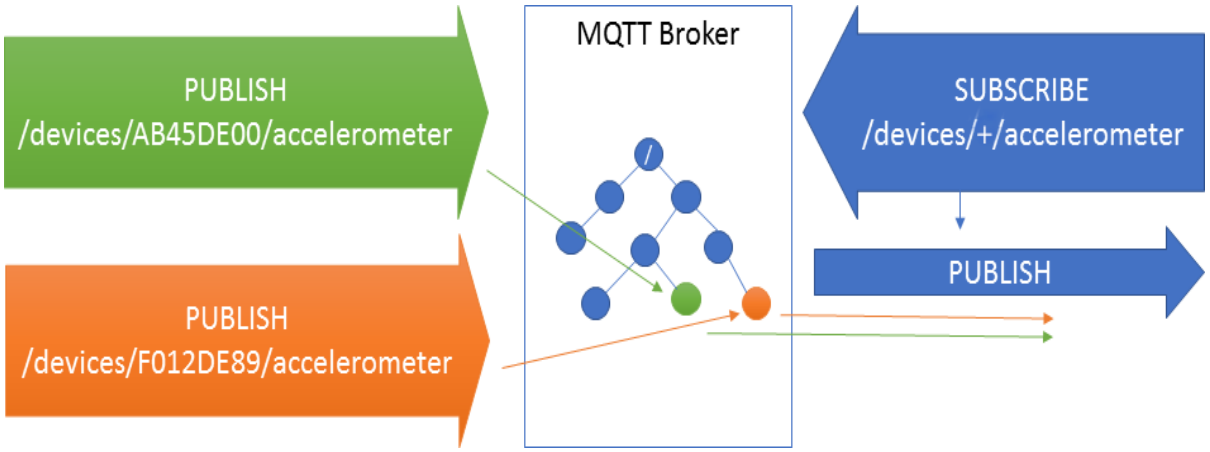


Figure 2: MQTT Protocol

Every published message from any publisher is eligible for delivery into any client session where a subscription exists with a matching topic filter. MQTT is very suitable for fast and –online‖ dispatch and distribution of messages to many subscribers in scenarios where it's feasible for the entirety of the consolidated published message stream to be inspected on behalf of all concurrent subscribers.

MQTT's –subscribe‖ gesture is much lighter weight. It establishes a filter context and simultaneously initiates and unbounded receive operation on that context. If session recovery is used, to scope of undelivered messages is that individual filter context. Subscribing is receiving. In some brokers, such an MQTT subscription context may indeed be backed by a volatile queue to allow leveling between fast and slow subscribers and to allow for caching of messages while a subscriber is temporarily disconnected and if session recovery is supported; but that's an implementation detail, not an explicit construct. The trouble with MQTT is that it uses TCP connections to a MQTT broker. Having an always-on connection will limits the time the devices can be put to sleep. This can be somewhat mitigated by using MQTT-S, which works with UDP

instead of TCP. But MQTT also lacks encryption since the protocol was intended to be lightweight and encryption would add significant overhead.

Advanced Message Queuing Protocol (AMQP) is an open source published standard for asynchronous messaging by wire. AMQP enables encrypted and interoperable messaging between organizations and applications. The protocol is used in client/server messaging and in IoT device management. AMPQ is efficient, portable, multichannel and secure. The messaging protocol is fast and features guaranteed delivery with acknowledgement of received messages. AMPQ works well in multi-client environments and provides a means for delegating tasks and making servers handle immediate requests faster. Because AMPQ is a streamed binary messaging system with tightly mandated messaging behavior, the interoperability of clients from different vendors is assured.
AMQP allows for various guaranteed messaging modes specifying a message be sent:

- At-most-once(sent one time with the possibility of being missed).
- At-least-once (guaranteeing delivery with the possibility of duplicated messages).
- Exactly-once (guaranteeing a one-time only delivery).

eXtensible Messaging and Presence Protocol (XMPP) is a TCP protocol based on XML. It enables the exchange of structured data between two or more connected entities, and out of the box it supports presence and contact list maintenance (since it started as a chat protocol). Because of the open nature of XML, XMPP can be easily extended to include publish- subscribe systems, making it a good choice for information that is handed to a central server and then distributed to numerous IoT devices at once. It is decentralized, and authentication can be built in by using a centralized XMPP server. The downsides of XMPP for IoT is that it lacks end-to-end encryption. It also doesn’t have quality-of-service functionality, which can be a real deal-breaker depending on the application.

Constrained Application Protocol (CoAP) is a protocol specifically developed for resource- constrained devices. It uses UDP instead of TCP, and developers can work with CoAP the same way they work with REST-based APIs. Since it uses minimal resources, it is a good option or low-power sensors. Since it uses UDP, it also can run on top of packet-based technologies such as SMS, and messages can be marked confirmable or nonconfirmable to work with QoS. Datagram Transport Layer Security (DTLS) can be used for encryption. The downside of CoAP is that it is a one-to-one protocol, so broadcast capabilities are not native to the protocol.

Attack Surfaces on Device to Cloud Communication

1. SQL injection , Cross-site scripting , Cross-site Request Forgery possible attacks on cloud application interfaces.

SQL Injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements (also commonly referred to as a malicious *payload*) that control a web application's database server (also commonly referred to as a Relational Database Management System – RDBMS). Since an SQL Injection vulnerability could possibly affect any website or web application that makes use of an SQL-based database, the vulnerability is one of the oldest, most prevalent and most dangerous of web application vulnerabilities.

2. Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts (also commonly referred to as a malicious payload) into a legitimate website or web application. XSS is amongst the most rampant of web application vulnerabilities and occurs when a web application makes use of unvalidated or unencoded user input within the output it generates. By leveraging XSS, an attacker does not target a victim directly. Instead, an attacker would exploit a vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script to the victim's browser.

3. Username and password enumeration attacks

4. MITM attacks

Man-in-the-middle attack (MITM) is an attack where the attacker secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other. One example of man-in-the-middle attacks is active eavesdropping, in which the attacker makes independent connections with the victims and relays messages between them to make them believe they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker. The attacker must be able to intercept all relevant messages passing between the two victims and inject new ones.

5. Man in the Cloud (MiTC) attacks

   Man in the cloud (MitC) attacks are interesting, and worrying, as they do not require any exploits or the running malicious code in order to get a grip during the initial infection stage. Instead they rely upon the type of common file synchronization service that we have all become used to, the likes of DropBox or Google Drive for example, to be the infrastructure providing command and control, data exfiltration and remote access options. Man in the cloud attacks are interesting, and worrying, as they do not require any exploits or the running malicious code .Simply by reconfiguring these cloud services, without end user knowledge and without the need for plaintext credential compromise to have occurred. It is hard for common security measures to detect as the synchronization protocol being used makes it all but impossible to distinguish between malicious and normal traffic.

Best Practices for securing Device to Cloud Security:

- Check all cloud interfaces are reviewed for security vulnerabilities (e.g. API interfaces and cloud-based web interfaces)
- Make sure cloud-based web interface not having weak passwords
- Ensure that any cloud-based web interface has an account lockout mechanism
- Implement two-factor authentication for cloud-based web interfaces

- Maintain transport encryption
- Ensure that any cloud-based web interface has been tested for XSS, SQLi and CSRF vulnerabilities.

2. IoT in Cloud

The advent of cloud computing has acted as a catalyst for the development and deployment of scalable Internet-of-Things business models and applications. Therefore, IoT and cloud are nowadays two very closely affiliated future internet technologies, which go hand-in-hand in non-trivial IoT deployments.

Cloud computing is the next evolutionary step in Internet-based computing, which provides the means for delivering ICT resources as a service. The ICT resources that can be delivered through cloud computing model include computing power, computing infrastructure (e.g.,servers and/or storage resources), applications, business processes and more. Cloud computing infrastructures and services have the following characteristics, which typically differentiate them from similar (distributed computing) technologies:

- **Elasticity and the ability to scale up and down**: Cloud computing services can scale upwards during high periods of demand and downward during periods of lighter demand. This elastic nature of cloud computing facilitates the implementation of flexibly scalable business models, e.g., through enabling enterprises to use more or less resources as their business grows or shrinks.
- **Self-service provisioning and automatic deprovisioning**: Contrary to conventional web-based Application Service Providers (ASP) models (e.g., web hosting), cloud computing enables easy access to cloud services without a lengthy provisioning process. In cloud computing, both provisioning and de-provisioning of resources can take place automatically.
- **Application programming interfaces (APIs)**: Cloud services are accessible via APIs, which enable applications and data sources to communicate with each other.
- **Billing and metering of service usage in a pay-as-you-go model**: Cloud services are associated with a utility-based pay-as-you-go model. To this end, they provide the means for metering resource usage and subsequently issuing

bills.

- **Performance monitoring and measuring**: Cloud computing infrastructures provide a service management environment along with an integrated approach for managing physical environments and IT systems.
- **Security**: Cloud computing infrastructures offer security functionalities towards safeguarding critical data and fulfilling customers' compliance requirements.

The two main business drivers behind the adoption of a cloud computing model and associated services including:

- **Business Agility:** Cloud computing alleviates tedious IT procurement processes, since it facilitates flexible, timely and on-demand access to computing resources (i.e. compute cycles, storage) as needed to meet business targets.

Depending on the types of resources that are accessed as a service, cloud computing is associated with different service delivery models.

- **Infrastructure as a Service (IaaS)**: IaaS deals with the delivery of storage and computing resources towards supporting custom business solutions. Enterprises opt for an IaaS cloud computing model in order to benefit from lower prices, the ability to aggregate resources, accelerated deployment, as well as increased and customized security. The most prominent example of IaaS service Amazon's Elastic Compute Cloud (EC2), which uses the Xen open-source hypervisor to create and manage virtual machines.
- **Platform as a Service (PaaS)**: PaaS provides development environments for creating cloud-ready business applications. It provides a deeper set of capabilities comparing to IaaS, including development, middleware, and deployment capabilities. PaaS services create and encourage deep ecosystem of partners who commit to this environment. Typical examples of PaaS services are Google's App Engine and Microsoft's Azure cloud environment, which both provide a workflow engine, development tools, a testing environment, database integration functionalities, as well as third-party tools and services.
- **Software as a Service (SaaS)**: SaaS services enable access to purpose-built business applications in the cloud. Such services provide the pay-go-go,

reduced CAPEX and elastic properties of cloud computing infrastructures. Cloud services can be offered through infrastructures (clouds) that are publicly accessible (i.e. public cloud services), but also by privately owned infrastructures (i.e. private cloud services). Furthermore, it is possible to offer services supporting by both public and private clouds, which are characterized as hybrid cloud services.

IoT/Cloud Convergence

Internet-of-Things can benefit from the scalability, performance and pay-as-you-go nature of cloud computing infrastructures. Indeed, as IoT applications produce large volumes of data and comprise multiple computational components (e.g., data processing and analytics algorithms), their integration with cloud computing infrastructures could provide them with opportunities for cost-effective on-demand scaling. As prominent examples consider the following settings:

- A Small Medium Enterprise (SME) developing an energy management IoT product, targeting smart homes and smart buildings. By streaming the data of the product (e.g., sensors and WSN data) into the cloud it can accommodate its growth needs in a scalable and cost effective fashion.
- A smart city can benefit from the cloud-based deployment of its IoT systems and applications. A city is likely to deploy many IoT applications, such as applications for smart energy management, smart water management, smart transport management, urban mobility of the citizens and more. These applications comprise multiple sensors and devices, along with computational components. Furthermore, they are likely to produce very large data volumes. Cloud integration enables the city to host these data and applications in a cost-effective way. Furthermore, the elasticity of the cloud can directly support expansions to these applications, but also the rapid deployment of new ones without major concerns about the provisioning of the required cloud computing resources.
- A cloud computing provider offering pubic cloud services can extend them to the IoT area, through enabling third-parties to access its infrastructure in order to integrate IoT data and/or computational components operating over IoT devices. The provider can offer IoT data access and services in a pay-as-you-fashion, through enabling third-parties to access resources of its

infrastructure and accordingly to charge them in a utility-based fashion.

One of the earliest efforts has been the famous Pachube.com infrastructure (used extensively for radiation detection and production of radiation maps during earthquakes in Japan). Pachube.com has evolved (following several evolutions and acquisitions of this infrastructure) to Xively.com, which is nowadays one of the most prominent public IoT clouds. Nevertheless, there are tens of other public IoT clouds as well, such as ThingsWorx, ThingsSpeak, Sensor-Cloud, Realtime.io and more. The list is certainly non-exhaustive. These public IoT clouds offer commercial pay-as-you-go access to end-users wishing to deploying IoT applications on the cloud. Most of them come with developer friendly tools, which enable the development of cloud applications, thus acting like a PaaS for IoT in the cloud.
Similarly to cloud computing infrastructures, IoT/cloud infrastructures and related services can be classified to the following models:

- **Infrastructure-as-a-Service (IaaS) IoT/Clouds**: These services provide the means for accessing sensors and actuator in the cloud. The associated business model involves the IoT/Cloud provide to act either as data or sensor provider. IaaS services for IoT provide access control to resources as a prerequisite for the offering of related pay-as-you-go services.
- **Platform-as-a-Service (PaaS) IoT/Clouds**: This is the most widespread model for IoT/cloud services, given that it is the model provided by all public IoT/cloud infrastructures outlined above. As already illustrate most public IoT clouds come with a range of tools and related environments for applications development and deployment in a cloud environment. A main characteristic of PaaS IoT services is that they provide access to data, not to hardware. This is a clear differentiator comparing to IaaS.
- **Software-as-a-Service (SaaS) IoT/Clouds**: SaaS IoT services are the ones enabling their uses to access complete IoT-based software applications through the cloud, on- demand and in a pay-as-you-go fashion. As soon as sensors and IoT devices are not visible, SaaS IoT applications resemble very much conventional cloud-based SaaS applications.

The benefits of integrating IoT into Cloud are discussed in this section as follows.

1. Communication

The Cloud is an effective and economical solution which can be used to connect, manage, and track anything by using built-in apps and customized portals . The availability of fast systems facilitates dynamic monitoring and remote objects control, as well as data real-time access. It is worth declaring that, although the Cloud can greatly develop and facilitate the IoT interconnection, it still has weaknesses in certain areas. Thus, practical restrictions can appear when an enormous amount of data needs to be transferred from the Internet to the Cloud.

2. Storage

As the IoT can be used on billions of devices, it comprises a huge number of information sources, which generate an enormous amount of semi-structured or non-structured data . This is known as Big Data, and has three characteristics : variety (e.g. data types), velocity (e.g. data generation frequency), and volume (e.g. data size). The Cloud is considered to be one of the most cost-effective and suitable solutions when it comes to dealing with the enormous amount of data created by the IoT. Moreover, it produces new chances for data integration, aggregation, and sharing with third parties .

3. Processing capabilities

IoT devices are characterized by limited processing capabilities which prevent on-site and complex data processing. Instead, gathered data is transferred to nodes that have high capabilities; indeed, it is here that aggregation and processing are accomplished. However, achieving scalability remains a challenge without an appropriate underlying infrastructure. Offering a solution, the Cloud provides unlimited virtual processing capabilities and an on- demand usage model . Predictive algorithms and data-driven decisions making can be integrated into the IoT in order to increase revenue and reduce risks at a lower cost .

4. Scope

With billions of users communicating with one another together and a variety of information being collected, the world is quickly moving towards the Internet of Everything (IoE) realm - a network of networks with billions of things that generate new chances and risks . The Cloud-based IoT approach provides new applications and services based on the expansion of the Cloud through the IoT

objects, which in turn allows the Cloud to work with a number of new real world scenarios, and leads to the emergence of new services .

5. New abilities

The IoT is characterised by the heterogeneity of its devices, protocols, and technologies. Hence, reliability, scalability, interoperability, security, availability and efficiency can be very hard to achieve. Integrating IoT into the Cloud resolves most of these issues. It provides other features such as easeof-use and ease-of-access, with low deployment costs .

6. New Models

Cloud-based IoT integration empowers new scenarios for smart objects, applications, and services. Some of the new models are listed as follows:

- SaaS (Sensing as a Service) , which allows access to sensor data;
- EaaS (Ethernet as a Service), the main role of which is to provide ubiquitous connectivity to control remote devices;
- SAaaS (Sensing and Actuation as a Service), which provides control logics automatically.
- IPMaaS (Identity and Policy Management as a Service) , which provides access to policy and identity management.
- DBaaS (Database as a Service), which provides ubiquitous database management;
- SEaaS (Sensor Event as a Service) , which dispatches messaging services that are generated by sensor events;
- SenaaS (Sensor as a Service) , which provides management for remote sensors;
- DaaS (Data as a Service), which provides ubiquitous access to any type of data.

3. Cloud Architecture

The cloud components of IoT architecture are positioned within a three-tier architecture pattern comprising edge, platform and enterprise tiers, as described in the Industrial Internet Consortium Reference Architecture . The edge-tier includes Proximity Networks and Public Networks where data is collected from devices and transmitted to devices. Data flows through the IoT gateway or optionally directly from/to the device then through edge services into the cloud provider via IoT transformation and connectivity. The Platform tier is the provider cloud, which receives, processes and analyzes data flows from the edge tier and provides API Management and Visualization. It provides the capability to initiate control commands from the enterprise network to the public network as well. The Enterprise tier is represented by the Enterprise

Network comprised of Enterprise Data, Enterprise User Directory, and Enterprise Applications. The data flow to and from the enterprise network takes place via a Transformation and Connectivity component. The data collected from structured and non-structured data sources, including real-time data from stream computing, can be stored in the enterprise data.

One of the features of IoT systems is the need for application logic and control logic in a hierarchy of locations, depending on the timescales involved and the datasets that need to be brought to bear on the decisions that need to be made. Some code may execute directly in the devices at the very edge of the network, or alternatively in the IoT Gateways close to the devices. Other code executes centrally in the provider cloud services or in the enterprise network. This is sometimes alternatively called ‑fog computing‖ to contrast with centralised ‑cloud computing‖, although fog computing can also contain one or more layers below the cloud that each could potentially provide capabilities for a variety of services like analytics.

Aspects of the architecture include:

- The user layer is independent of any specific network domain. It may be in or outside any specific domain.
- The proximity network domain has networking capabilities that typically extend the public network domain. The devices (including sensor/actuator, firmware and management agent) and the physical entity are part of the proximity network domain. The devices communicate for both data flow and control flow either via an IoT Gateway and edge services or directly over the public network via edge services.
- The public network and enterprise network domains contain data sources

  that feed the entire architecture. Data sources include traditional systems of record from the enterprise as well as new sources from Internet of Things (IoT). The public network includes communication with peer clouds.
- The provider cloud captures data from devices, peer cloud services and other data sources (for example Weather services). It can use integration technologies or stream processing to transform, filter and analyse this data in real time and it can store

the data into repositories where further analytics can be performed. This processing, which can be augmented with the use of Cognitive and Predictive analytics, is used to generate Actionable Insights. These insights are used by users and enterprise applications and can also be used to trigger actions to be performed by IoT Actuators. All of this needs to be done in a secure and governed environment.

The following figure 3 shows the capabilities and relationships for supporting IoT using cloud computing.

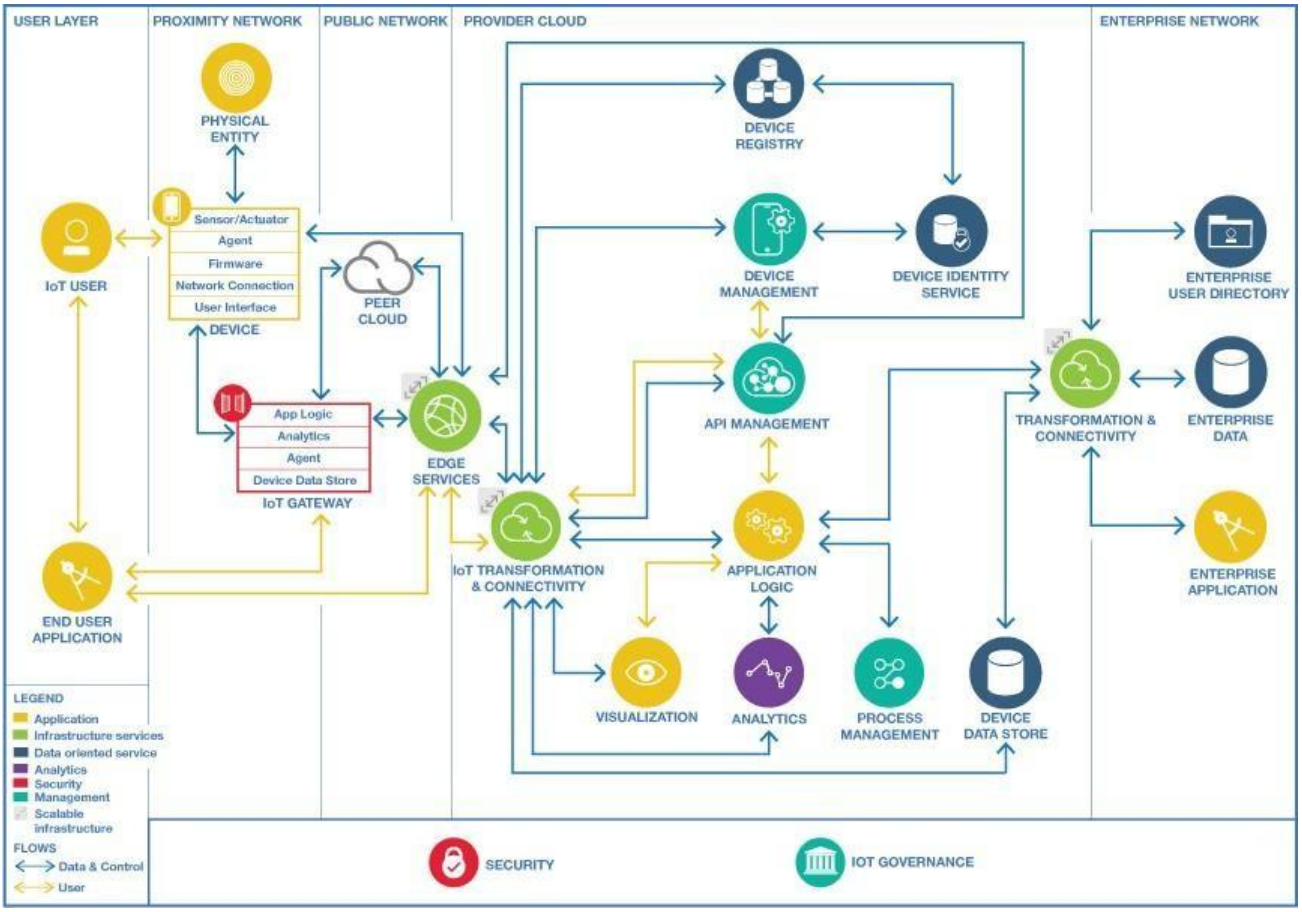


Figure 3: Cloud Components for IoT

User Layer - contains IoT users and their end user applications.

- IoT User (people/system) - a person or alternatively an automated system that makes use of one or more end user applications to achieve some goal. The IoT User is one of the main beneficiaries of the IoT solution.
- End User Application - domain specific or device specific application. The IoT user may use end user applications that run on smart phones, tablets, PCs or alternatively on specialised IoT devices including control panels.

Proximity Network - contains the physical entities that are at the heart of the IoT system, along with the devices that interact with the physical entities and connect them to the IoT system.

Physical Entity - the physical entity is the real-world object that is of interest – it is subject to sensor measurements or to actuator behavior. It is the ―thing‖ in the Internet of Things. This architecture distinguishes between the physical entities and the IT devices that sense them or act on them. For example, the thing can be the ocean and the device observing is it a water temperature thermometer.

Device - contains sensor(s) and/or actuator(s) plus a network connection that enables interaction with the wider IoT system. There are cases where the device is also the physical entity being monitored by the sensors – such as an accelerometer inside a smart phone.

- Sensor/Actuator - senses and acts on physical entities. A sensor is a component that senses or measures certain characteristics of the real world and converts them into a digital representation. An actuator is a component that accepts a digital command to act on a physical entity in some way.
- Agent - provides remote management capabilities for the device, supporting a device management protocol that can be used by the Device Management service or IoT management system.
- Firmware - software that provides control, monitoring and data

manipulation of engineered products and systems. The firmware contained in devices such as consumer electronics provides the low-level control program for the devices.

- Network Connection - provides the connection from the device to the IoT system. This is often a local network that connects the device with an IoT gateway – low power and low range in many cases to reduce the power demands on the device.
- User Interface - allows users to interact with applications, agents, sensors and actuators (optional – some devices have no user interface and all interaction takes place from remote applications over the network).

IoT Gateway - acts as a means for connecting one or more devices to the public network (typically the Internet). It is commonly the case that devices have limited network connectivity – they may not be able to connect directly to the Internet. This can be for a number of reasons, including the limitation of power on the device, which can restrict the device to using a low-power local network. The local network enables the devices to communicate with a local IoT Gateway, which is then able to communicate with the public network. The IoT Gateway contains the following components:

- App Logic - provides domain specific or IoT solution specific logic that runs on the IoT Gateway. For IoT systems that have Actuators which act on physical entities, a significant capability of the app logic is the provision of control logic which makes decisions on how the actuators should operate, given input from sensors and data of other kinds, either held locally or held centrally.
- Analytics - provides Analytics capability locally rather than in the provider cloud.
- Agent - allows management of the IoT Gateway itself and can also enable management of the attached devices by providing a connection to the provider cloud layer's Device Management.service via the device management protocol.
- Device Data Store - stores data locally. Devices may generate a large amount of data in real time it may need to be stored locally rather than being transmitted to a central location. Data in the device data store

can be used by the application logic and analytics capability in the IoT Gateway.

Public Network - contains the wide area networks (typically the internet), peer cloud systems, the edge services.

Peer Cloud - a 3rd party cloud system that provides services to bring data and capabilities to the IoT platform. Peer clouds for IoT may contribute to the data in the IoT system and may also provide some of the capabilities defined in this IoT architecture. For example an IoT for Insurance solution may use services from partners, such as weather data.

Edge Services - services needed to allow data to flow safely from the internet into the provider cloud and into the enterprise. Edge services also support end user applications. Edge services include:

Domain Name System Server - resolves the URL for a particular web resource to the TCP-IP address of the system or service that can deliver that resource.

Content Delivery Networks (CDN) - support end user applications by providing geographically distributed systems of servers deployed to minimize the response time for serving resources to geographically distributed users, ensuring that content is highly available and provided to users with minimum latency. Which servers are engaged will depend on server proximity to the user, and where the content is stored or cached.

Firewall - controls communication access to or from a system permitting only traffic meeting a set of policies to proceed and blocking any traffic that does not meet the policies. Firewalls can be implemented as separate dedicated hardware, or as a component in other networking hardware such as a load-balancer or router or as integral software to an operating system.

Load Balancers - provides distribution of network or application traffic across many resources (such as computers, processors, storage, or network links) to maximize throughput, minimize response time, increase capacity and increase reliability of applications. Load balancers can balance loads locally and globally. Load balancers should be highly available without a single point of failure. Load balancers are sometimes integrated as part of the provider cloud analytical system

components like stream processing, data integration, and repositories.

Provider Cloud - provides core IoT applications and associated services including storage of device data; analytics; process management for the IoT system; create visualizations of data. Also hosts components for device management including a device registry.

A cloud computing environment provides scalability and elasticity to cope with varying data volume, velocity and related processing requirements. Experimentation and iteration using different cloud service configurations is a good way to evolve the IoT system, without upfront capital investment.

IoT Transformation and Connectivity - enables secure connectivity to and from IoT devices. This component must be able to handle and perhaps transform high volumes of messages and quickly route them to the right components in the IoT solution. The Transformation and Connectivity component includes the following capabilities:

- Secure Connectivity - provides the secured connectivity which authenticates and authorizes access to the provider cloud.
- Scalable Messaging - provides messaging from and to IoT devices. Scalability of the messaging component is essential to support high data volume applications and applications with highly variable data rates, like weather.
- Scalable Transformation - provides transformation of device IoT data before it gets to provider cloud layer, to provide a form more suitable for processing and analysis. This may include decoding messages that are encrypted, translating a compressed formatted message, and/or normalizing messages from varying devices.

Application Logic - The core application components, typically coordinating the handling of IoT device data, the execution of other services and supporting end user applications. An Event based programming model with trigger, action and rules is often a good way to write IoT application logic. Application logic can include workflow. Application logic may also include control logic, which determines how to use actuators to affect physical entities, for those IoT systems that have actuators.

Visualization - enables users to explore and interact with data from the

data repositories, actionable insight applications, or enterprise applications. Visualization capabilities include End user UI, Admin UI & dashboard as sub components.

- End User UI - allows users to communicate and interact with Enterprise applications, analytics results, etc. This also includes internal or customer facing mobile user interfaces.
- Admin UI - enables administrators to access metrics, operation data, and various logs.
- Dashboard - allows users to view various reports. Admin UI and Dashboard are internal facing user interfaces.

Analytics - Analytics is the discovery and communication of meaningful patterns of information found in IoT data, to describe, to predict, and to improve business performance.

Process Management - activities of planning, developing, deploying and monitoring the performance of a business process. For IoT systems, real-time process management may provide significant benefits.

Device Data Store - stores data from the IoT devices so that the data can be integrated with processes and applications that are part of the IoT System. Devices may generate a large amount of data in real time calling for the Device Data Store to be elastic and scalable.

API Management - publishes catalogues and updates APIs in a wide variety of deployment environments. This enables developers and end users to rapidly assemble solutions through discovery and reuse of existing data, analytics and services.

Device Management - provides an efficient way to manage and connect devices securely and reliably to the cloud platform. Device management contains device provisioning, remote administration, software updating, remote control of devices, monitoring devices. Device management may communicate with management agents on devices using management protocols as well as communicate with management systems for the IoT solutions.

Device Registry - stores information about devices that the IoT system may read, communicate with, control, provision or manage. Devices may need to be registered before they can connect to and or be managed by the IoT system. IoT deployments may have a large number of devices therefore scalability of the registry is important.

Device Identity Service - ensures that devices are securely identified before being granted access to the IoT systems and applications. In the IoT systems, device identification can help address threats that arise from fake servers or fake devices.

Transformation and Connectivity - enables secure connections to enterprise systems and the ability to filter, aggregate, or modify data or its format as it moves between cloud and IoT systems components and enterprise systems (typically systems of record). Within the IoT reference architecture the transformation and connectivity component sits between the cloud provider and enterprise network. However, in a hybrid cloud model these lines might become blurred. The Transformation and Connectivity component includes the following capabilities:

- Enterprise Secure Connectivity - integrates with enterprise data security systems to authenticate and authorize access to enterprise systems.
- Transformation - transforms data going to and from enterprise systems.
- Enterprise Data Connectivity - enables provider cloud components to connect securely to enterprise data. Examples include VPN and gateway tunnels.

Enterprise Network - host a number of business specific enterprise applications that deliver critical business solutions along with supporting elements including enterprise data. Typically, enterprise applications have sources of data that are extracted and integrated with services provided by the cloud provider. Analysis is performed in the cloud computing environment, with output consumed by the enterprise applications.

Enterprise Data - includes metadata about the data as well as systems of record for enterprise applications. Enterprise data may flow directly to data integration or the data repositories providing a feedback loop in the

analytical system for IoT. IoT systems may store raw, analyzed, or processed data in appropriate Enterprise Data elements. Enterprise Data includes:

Enterprise User Directory - stores user information to support authentication, authorization, or profile data. The security services and edge services use this to control access to the enterprise network, enterprise services, or enterprise specific cloud provider services.

Enterprise Applications - Enterprise applications consume cloud provider data and analytics to produce results that address business goals and objectives. Enterprise applications can be updated from enterprise data or from IoT applications or they can provide input and content for enterprise data and

Security and Privacy - Security and Privacy in IoT deployments must address both information technology (IT) security as well as operations technology (OT) security elements. Furthermore, the level of attention to security and the topic areas to address varies depending upon the application environment, business pattern, and risk assessment. A risk assessment will take into account multiple threats and attacks along with an estimate of the potential costs associated with such attacks. In addition to security considerations, the connecting of IT systems with physical systems also brings with it the need to consider the impact to safety that the IoT system may have. IoT systems must be designed, deployed, and managed such that they can always bring the system to a safe operating state, even when disconnected from communications with other systems that are part of the deployment. **Identity and Access Management-** As with any computing system, there must be strong identification of all participating entities – users, systems, applications, and, in the case of IoT, devices and the IoT gateways through which those devices communicate with the rest of the system. Device identity and management necessarily involves multiple entities, starting with chip and device manufacturers, including IoT platform providers, and also including enterprise users and operators of the devices. In IoT solutions it is often the case that multiple of these entities will continue to communicate and address the IoT devices throughout their operational lifetime.

Data Protection -Data in the device, in flight throughout the public network, provider cloud, and enterprise network, as well as at rest in a variety of locations and formats must be protected from inappropriate access and use. Multiple methods can be utilized, and indeed, in many cases, multiple methods are applied simultaneously to provide different levels of protection of dataagainst different types ofthreatsor isolation from different entities supporting the system.

4. AWS IoT

AWS IoT provides secure, bi-directional communication between Internet-connected devices such as sensors, actuators, embedded micro-controllers, or smart appliances and the AWS Cloud. This enables you to collect telemetry data from multiple devices, and store and analyze the data. You can also create applications that enable your users to control these devices from their phones or tablets.

AWS IoT consists of the following components:

Device gateway -Enables devices to securely and efficiently communicate with AWS IoT. **Message broker-**Provides a secure mechanism for devices and AWS IoT applications to publish and receive messages from each other. You can use either the MQTT protocol directly or MQTT over WebSocket to publish and subscribe. You can use the HTTP REST interface to publish.
Rules engine-Provides message processing and integration with other AWS services. You can use an SQL-based language to select data from message payloads, and then process and send the data to other services, such as Amazon S3, Amazon DynamoDB, and AWS Lambda. You can also use the message broker to republish messages to other subscribers.
Security and Identity service-Provides shared responsibility for security in the AWS Cloud. Your devices must keep their credentials safe in order to securely send data to the message broker. The message broker and rules engine use AWS security features to send data securely to devices or other AWS services.
Registry-Organizes the resources associated with each device in the AWS Cloud. You register your devices and associate up to three custom attributes with each one. You can also associate certificates and MQTT client IDs with each device to improve your ability to manage and troubleshoot them.

Group registry-Groups allow you to manage several devices at once by categorizing them into groups. Groups can also contain groups—you can build a hierarchy of groups. Any action you perform on a parent group will apply to its child groups, and to all the devices in it and in all of its child groups as well. Permissions given to a group will apply to all devices in the group and in all of its child groups.

Device shadow-A JSON document used to store and retrieve current state information for a device.

Device Shadow service-Provides persistent representations of your devices in the AWS Cloud. You can publish updated state information to a device's shadow, and your device can synchronize its state when it connects. Your devices can also publish their current state to a shadow for use by applications or other devices.

Device Provisioning service- Allows you to provision devices using a template that describes the resources required for your device: a *thing*, a certificate, and one or more policies. A thing is an entry in the registry that contains attributes that describe a device. Devices use certificates to authenticate with AWS IoT. Policies determine which operations a device can perform in AWS IoT.

Custom Authentication service- You can define custom authorizers that allow you to manage your own authentication and authorization strategy using a custom authentication service and a Lambda function. Custom authorizers allow AWS IoT to authenticate your devices and authorize operations using bearer token authentication and authorization strategies. Custom authorizers can implement various authentication strategies (for example: JWT verification, OAuth provider call out, and so on) and must return policy documents which are used by the device gateway to authorize MQTT operations.

Jobs Service- Allows you to define a set of remote operations that are sent to and executed on one or more devices connected to AWS IoT. For example, you can define a job that instructs a set of devices to download and install application or firmware updates, reboot, rotate certificates, or perform remote troubleshooting operations. To create a job, you specify a description of the remote operations to be performed and a list of targets that should perform them. The targets can be individual devices, groups or both.

Accessing AWS IoT

AWS IoT provides the following interfaces to create and interact with your devices:

- **AWS Command Line Interface (AWS CLI)**—Run commands for AWS IoT on Windows, macOS, and Linux. These commands allow you to create and manage things, certificates, rules, and policies. To get started, see the AWS Command Line Interface User Guide.
- **AWS IoT API**—Build your IoT applications using HTTP or HTTPS requests. These API actions allow you to programmatically create and manage things, certificates, rules, and policies.
- **AWS SDKs**—Build your IoT applications using language-specific APIs. These SDKs wrap the HTTP/HTTPS API and allow you to program in any of the supported languages.

- **AWS IoT Device SDKs**—Build applications that run on devices that send messages to and receive messages from AWS IoT.

Related Services

AWS IoT integrates directly with the following AWS services:

- **Amazon Simple Storage Service**—Provides scalable storage in the AWS Cloud. **Amazon DynamoDB**—Provides managed NoSQL databases.
- **Amazon Kinesis**—Enables real-time processing of streaming data at a massive scale. **AWS Lambda**—Runs your code on virtual servers from Amazon EC2 in response to events.
- **Amazon Simple Notification Service**—Sends or receives notifications.
- **Amazon Simple Queue Service**—Stores data in a queue to be retrieved by applications.

Working of AWS IoT

- AWS IoT enables Internet-connected devices to connect to the AWS Cloud and lets applications in the cloud interact with Internet-connected devices. Common IoT applications either collect and process telemetry

from devices or enable users to control a device remotely.

- Devices report their state by publishing messages, in JSON format, on MQTT topics. Each MQTT topic has a hierarchical name that identifies the device whose state is being updated. When a message is published on an MQTT topic, the message is sent to the AWS IoT MQTT message broker, which is responsible for sending all messages published on an MQTT topic to all clients subscribed to that topic.
- Communication between a device and AWS IoT is protected through the use of X.509 certificates. AWS IoT can generate a certificate for you or you can use your own. In either case, the certificate must be registered and activated with AWS IoT, and then copied onto your device. When your device communicates with AWS IoT, it presents the certificate to AWS IoT as a credential.
- We recommend that all devices that connect to AWS IoT have an entry in the registry. The registry stores information about a device and the certificates that are used by the device to secure communication with AWS IoT.
- You can create rules that define one or more actions to perform based on the data in a message. For example, you can insert, update, or query a DynamoDB table or invoke a Lambda function. Rules use expressions to filter messages. When a rule matches a message, the rules engine invokes the action using the selected properties. Rules also contain an IAM role that grants AWS IoT permission to the AWS resources used to perform the action.

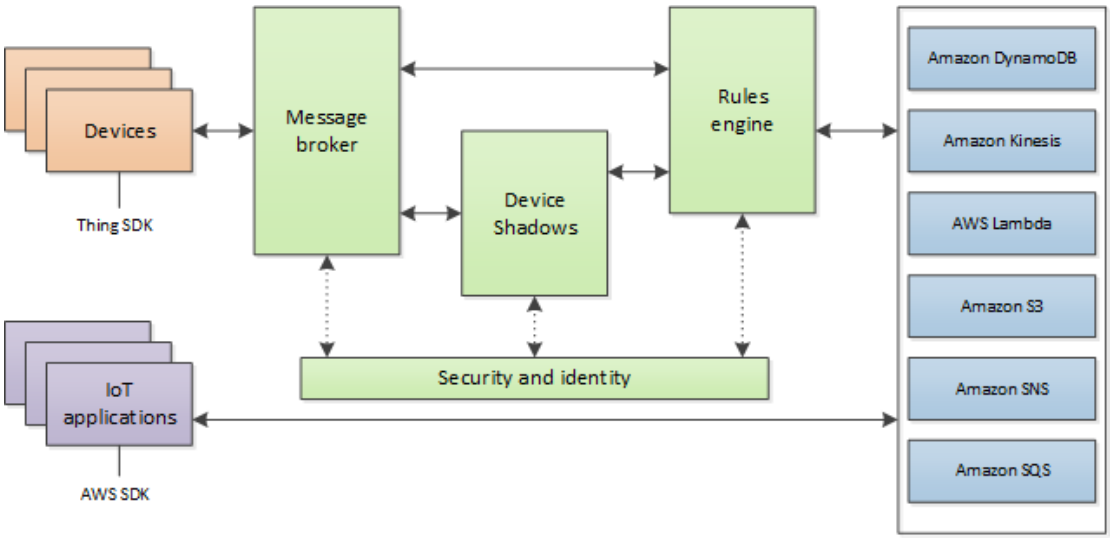


Figure 4: AWS IOT Architecture

- Each device has a shadow that stores and retrieves state information. Each item in the state information has two entries: the state last reported by the device and the desired state requested by an application. An application can request the current state information for a device. The shadow responds to the request by providing a JSON document with the state information (both reported and desired), metadata, and a version number. An application can control a device by requesting a change in its state. The shadow accepts the state change request, updates its state information, and sends a message to indicate the state information has been updated. The device receives the message, changes its state, and then reports its new state.

5. Managing Cloud Account Credentials

If you do not have an AWS account, create one.

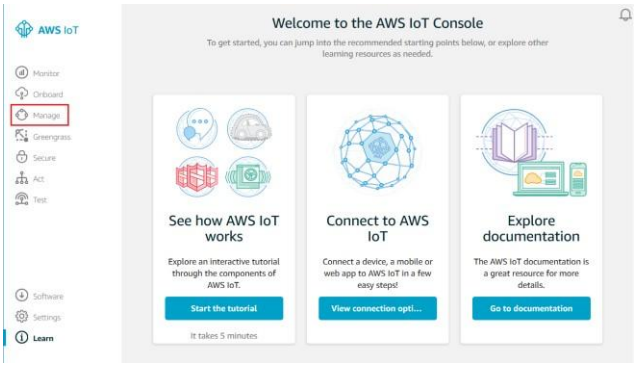
To create an AWS account:

1. Open the AWS home page and choose **Create an AWS Account**.
2. Follow the online instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using your phone's keypad.
3. Sign in to the AWS Management Console and open the AWS IoT console.
4. On the **Welcome** page, choose **Get started**.

Register a Device in the Registry

Devices connected to AWS IoT are represented by things in the registry. The registry allows you to keep a record of all of the devices that are connected to your AWS IoT account. The fastest way to start using your AWS IoT Button is to download the mobile app for iOS or Android. The mobile app creates the required AWS IoT resources for you, and adds an event source to your button that uses a Lambda blueprint to invoke a new AWS Lambda function of your choice. If you are unable to use the mobile apps, follow these instructions.

1. On the **Welcome to the AWS IoT Console** page, in the left navigation pane, choose **Manage** to expand the choices, and then choose **Things**.



2. On the page that says You don't have any things yet, choose Register a thing.

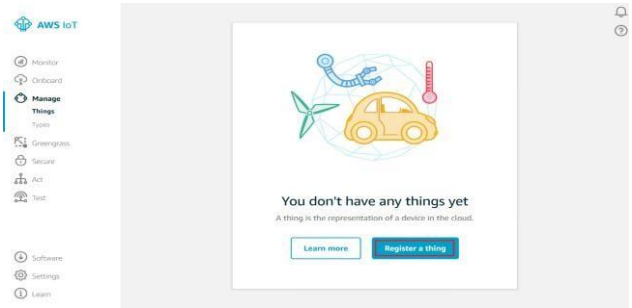


Figure 6: Register a Thing

3. On the **Creating AWS IoT things** page, choose **Create a single thing**.
4. On the **Create a thing** page, in the **Name** field, type a name for your device, such as **MyIoTButton**. Choose **Next** to add your device to the registry.

Create and Activate a Device Certificate

Communication between your device and AWS IoT is protected through the use of X.509 certificates. AWS IoT can generate a certificate for you or you can use your own X.509 certificate. AWS IoT generates the X.509 certificate for you.
Certificates must be activated prior to use.

1. Choose **Create certificate**.

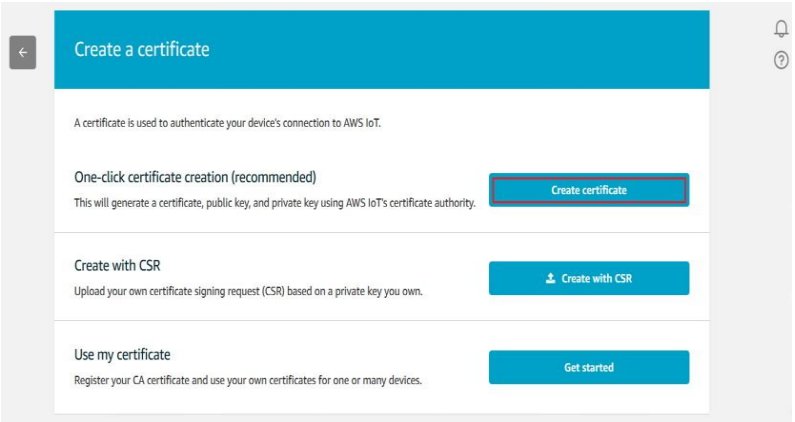


Figure 7: Certificate Creation

2. On the **Certificate created!** page, choose **Download** for the certificate, private key, and the root CA for AWS IoT (the public key need not be

downloaded). Save each of them to your computer, and then choose **Activate** to continue.Be aware that the downloaded filenames may be different than those listed on the **Certificate created!** page. For example:

- 2a540e2346-certificate.pem.crt.txt
- 2a540e2346-private.pem.key
- 2a540e2346-public.pem.key

Note

Although it is unlikely, root CA certificates are subject to expiration and/or revocation. If this should occur, you must copy new a root CA certificate onto your device.

3. Choose the back arrow until you have returned to the main **AWS IoT** console screen.

Create an AWS IoT Policy

X.509 certificates are used to authenticate your device with AWS IoT. AWS IoT policies are used to authorize your device to perform AWS IoT operations, such as subscribing or publishing to MQTT topics. Your device will presents its certificate when sending messages to AWS IoT. To allow your device to perform AWS IoT operations, you must create an AWS IoT policy and attach it to your device certificate.

1. In the left navigation pane, choose **Secure**, and then **Policies**. On the **You don't have a policy yet** page, choose **Create a policy**.
2. On the **Create a policy** page, in the **Name** field, type a name for the policy (for example,**MyIoTButtonPolicy**). In the **Action** field, type **iot:Connect**. In the **Resource ARN** field, type *. Select the Allow checkbox. This allows all clients to connect to AWS IoT.

You can restrict which clients (devices) are able to connect by specifying a client ARN as the resource. The client ARNs follow this format:

arn:aws:iot:*your-region*:*your-aws-account*:client/<*my-client-id*>

Finally, select the **Allow** check box. This allows your device to publish messages to the specified topic.After you have entered the information for your policy, choose **Create**.

Attach an AWS IoT Policy to a Device Certificate

Now that you have created a policy, you must attach it to your device certificate. Attaching an AWS IoT policy to a certificate gives the device the permissions specified in the policy.

1. In the left navigation pane, choose **Secure**, and then **Certificates**.

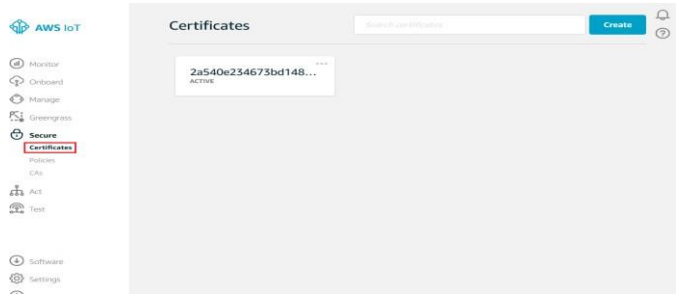


Figure 8: Attach Certificate

2. In the box for the certificate you created, choose **...** to open a drop-down menu, and then choose **Attach policy**.
3. In the **Attach policies to certificate(s)** dialog box, select the check box next to the policy you created in the previous step, and then choose **Attach**.

Attach a Certificate to a Thing

A device must have a certificate, private key and root CA certificate to authenticate with AWS IoT. We recommend that you also attach the device certificate to the thing that represents your device in AWS IoT. This allows you to create AWS IoT policies that grant permissions based on certificates attached to your things. For more information. see Thing Policy Variables

1. In the box for the certificate you created, choose **...** to open a drop-down menu, and then choose **Attach thing**.
2. In the **Attach things to certificate(s)** dialog box, select the check box next to the thing you registered, and then choose **Attach**.
3. To verify the thing is attached, select the box representing the certificate.
4. On the **Details** page for the certificate, in the left navigation pane, choose **Things**.
5. To verify the policy is attached, on the **Details** page for the certificate, in the left navigation pane, choose **Policies**.

Configure Your Device and Button

Configuring your device allows it to connect to your Wi-Fi network. Your device must be connected to your Wi-Fi network to install required files and send messages to AWS IoT. All devices must install a device certificate, private key, and root CA certificate in order to communicate with AWS IoT. The easiest way to configure your AWS IoT button is to use the AWS IoT button smart phone app. You can download it from the Apple App Store or the Google Play Store. If you are unable to use the smart phone app, follow these directions to configure your button.

Turn on your device

1. Remove the AWS IoT button from its packaging, and then press and hold the button until a blue blinking light appears. (This should take no longer than 15 seconds.)
2. The button acts as a Wi-Fi access point, so when your computer searches for Wi-Fi networks, it will find one called **Button ConfigureMe - XXX** where *XXX* is a three- character string generated by the button. Use your computer to connect to the button's Wi-Fi access point.

Configure a Different Device

Consult your device's documentation to connect to it and copy your device certificate, private key, and root CA certificate onto your device. You can use the AWS IoT MQTT client to better understand the MQTT messages sent by a device. Devices publish MQTT messages on topics. You can use the AWS IoT MQTT client to subscribe to these topics to see these messages.

To view MQTT messages:

1. In the AWS IoT console, in the left navigation pane, choose **Test**.

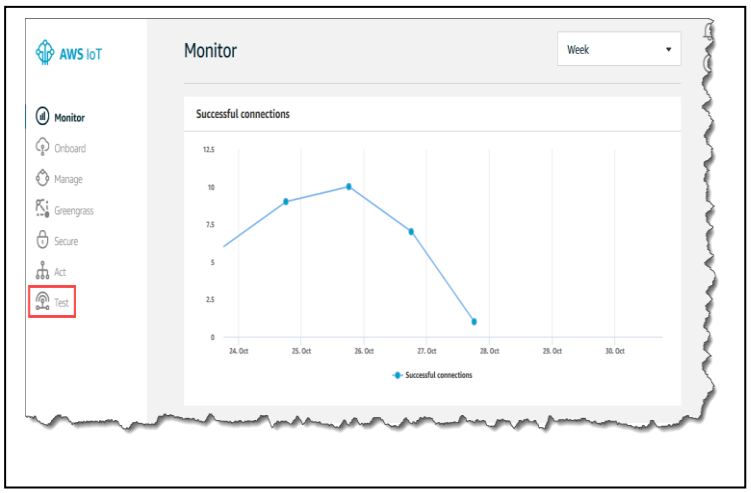


Figure 9: MQTT Messages

2. Subscribe to the topic on which your thing publishes. In the case of the AWS IoT button, you can subscribe to **iotbutton/+** (note that + is the wildcard character). In **Subscribe to a topic**, in the **Subscription topic** field, type **iotbutton/+**, and then choose **Subscribe to topic**. Choosing **Subscribe to topic** above, results in the topic **iotbutton/+** appearing in the**Subscriptions** column.
3. Press your AWS IoT button, and then view the resulting message in the AWS IoT MQTT client. If you do not have a button, you will simulate a

button press in the next step.

4. To use the AWS IoT console to publish a message:

   On the MQTT client page, in the **Publish** section, in the **Specify a topic and a message to publish**… field, type **iotbutton/ABCDEFG12345**. In the message payload section, type the following JSON:

```
{
   "serialNumber": "ABCDEFG12345",
   "clickType": "SINGLE",
   "batteryVoltage": "2000 mV"
}
```

   Choose **Publish to topic**. You should see the message in the AWS IoT MQTT client (choose **iotbutton/+** in the **Subscription** column to see the message).

Configure and Test Rules

The AWS IoT rules engine listens for incoming MQTT messages that match a rule. When a matching message is received, the rule takes some action with the data in the MQTT message (for example, writing data to an Amazon S3 bucket, invoking a Lambda function, or sending a message to an Amazon SNS topic). In this step, you will create and configure a rule to send the data received from a device to an Amazon SNS topic. Specifically, you will:

- Create an Amazon SNS topic.
- Subscribe to the Amazon SNS topic using a cell phone number.
- Create a rule that will send a message to the Amazon SNS topic when a message is received from your device.
- Test the rule using your AWS IoT button or an MQTT client.

6. Microsoft Azure

Millions of developers know how to create applications using the Windows Server programming model. Yet applications written for Windows

Azure, don't exactly use this familiar model. While most of a Windows developer's skills still apply, Windows Azure provides its own programming model. Many vendors' cloud platforms do just this, providing virtual machines (VMs) that act like on-premises VMs. This approach, commonly called Infrastructure as a Service (IaaS), certainly has value, and it's the right choice for some applications.

Instead of IaaS, Windows Azure offers a higher-level abstraction that's typically categorized as Platform as a Service (PaaS). While it's similar in many ways to the on- premises Windows world, this abstraction has its own programming model meant to help developers build better applications. Applications built using the Windows Azure programming model can be easier to administer, more available, and more scalable (up or down) than those built on traditional Windows servers.

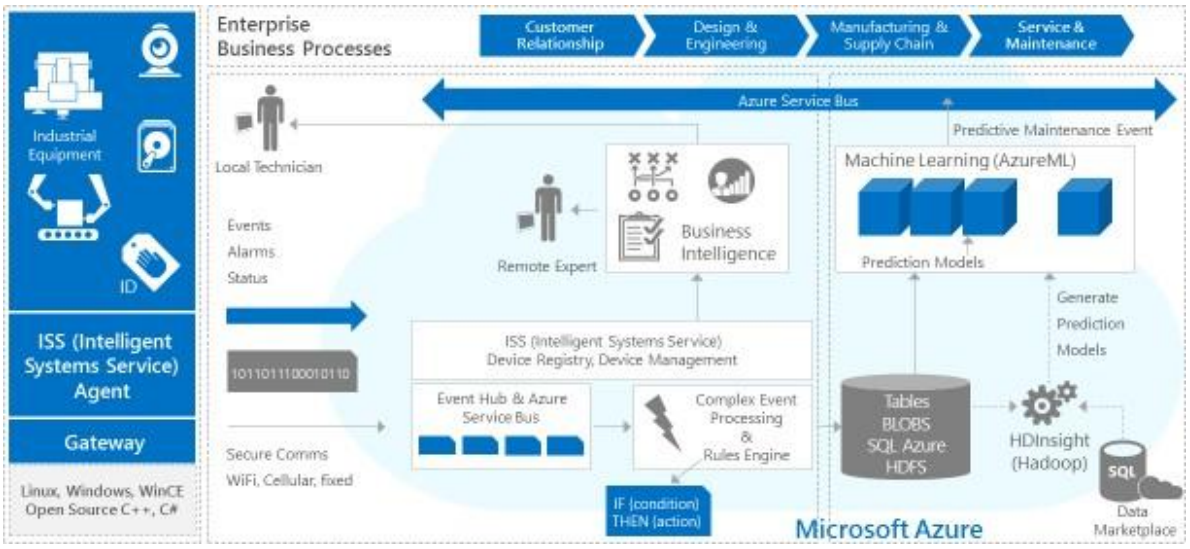


Figure 10: Microsoft Azure Architecture

Administration

In PaaS, the platform itself handles most of the administrative tasks. With Windows Azure, this means that the platform automatically takes care of things such as applying Windows patches and installing new versions of system software. The goal is to reduce the effort and the cost of administering the application environment.

Availability

Whether planned or not, today's applications usually have down time for patches, application upgrades, hardware failures, and other reasons. With cloud platforms there is no need for any downtime. The Windows Azure programming model is designed to let applications be continuously available, even in the face of software upgrades and hardware failures.

Scalability

The kinds of applications that people want to host in the cloud are often meant to handle lots of users. Yet the traditional Windows Server programming model wasn't explicitly designed to support Internet-scale applications. The Windows Azure programming model, however, was intended from the start to do this. Created for the cloud era, it's designed to let developers build the scalable applications that massive cloud data centres can support. Just as important, it also allows applications to scale down when necessary, letting them use just the resources they need and pay for only the computing resources used.

Windows Azure has three core components: Compute, Storage and Fabric. As the names suggest, Compute provides a computation environment with Web Role and Worker Role while Storage focuses on providing scalable storage (Blobs, Tables, Queue and Drives) for large-scale needs.

Fabric Controller

Windows Azure is designed to run in data centres containing lots of computers. Accordingly, every Windows Azure application runs on multiple machines simultaneously. All the computers in a particular Windows Azure data centre are managed by an application called the fabric controller. The fabric controller is itself a distributed application that runs across multiple computers. When a developer gives Windows Azure an application to run, he provides the code for the application's roles together with the service definition and service configuration files for this application. Among other things, this information tells the fabric controller how many instances of each role it should create. The fabric controller chooses a physical machine for

each instance, then creates a VM on that machine and starts the instance running. The role instances for a single application are spread across different machines within this data centre. Once it's created these instances, the fabric controller continues to monitor them. If an instance fails for any reason—hardware or software—the fabric controller will start a new instance for that role. While failures might cause an application's instance count to temporarily drop below what the developer requested, the fabric controller will always start new instances as needed to maintain the target number for each of the application's roles.

Windows Azure programming model:

- A Windows Azure application is built from one or more roles.
- A Windows Azure application runs multiple instances of each role.
- A Windows Azure application behaves correctly when any role instance fails.

Azure storage

Windows Azure offers blobs, tables, queues etc., as data storage options. They are a new type of data storage, they are fast and they are non-relational. Storage must be external to role instances. This is to ensure if a role instance fails, any data it contains is not lost. So Windows Azure stores data persistently outside role instances. This way another role instance can now access data that otherwise would have been lost if that data had been stored locally on a failed instance. Storage is replicated. Just as a Windows Azure application runs multiple role instances to allow for failures, Windows Azure storage provides multiple copies of data. Without this, a single failure would make data unavailable, something that's not acceptable for highly available applications.

Storage must be able to handle very large amounts of data. Traditional relational systems aren't necessarily the best choice for very large data sets. Since Windows Azure is designed in part for massively scalable applications, it must provide storage mechanisms for handling data at this scale. We can use blobs for storing binary data and tables for storing large structured data sets.

Windows Azure Application Deployment

When we deploy the application, you can select the subregion (which at the moment determines the data centre) where you want to host the application. You can also define affinity groups that you can use to group inter-dependent Azure applications and storage accounts together in order to improve performance and reduce costs. Performance improves because Windows Azure co-locates members of the affinity group in the same data centre. This reduces costs because data transfers within the same data centre do not incur bandwidth charges. Affinity groups offer a small advantage over simply selecting the same subregion for your hosted services, because Windows Azure makes a ―best effort‖ to optimise the location of those services.

Identity Management

All applications and services must manage user identity. This is particularly important in cloud-based scenarios that can potentially serve a very large number of customers and each of these customers may have their own identity framework. The ideal solution is a solution that takes advantage of the customers existing on-premises or federated directory service to enable single sign on (SSO) across their local and all external hosted services. This reduces the development effort of building individual and separate identity management systems. SSO allows users to access the application or service using their existing credentials.

Windows Azure - One or more instances of web roles and worker roles: Every Windows Azure application consists of one or more roles. When it executes, an application that conforms to the Windows Azure programming model must run at least two copies—two distinct instances—of each role it contains. Each instance runs as its own VM. Every instance of a particular role runs the exact same code. In fact, with most Windows Azure applications, each instance is just like all of the other instances of that role—they’re interchangeable. For example, Windows Azure automatically load balances HTTP requests

across an application’s Web role instances.

This load balancing doesn’t support sticky sessions, so there’s no way to direct all of a client’s requests to the same Web role instance. Storing client-specific state, such as a shopping cart, in a particular Web role instance won’t work, because Windows Azure provides no way to guarantee that all of a client’s requests will be handled by that instance. Instead, this kind of state must be stored externally, for example in SQL Azure. An application that follows the Windows Azure programming model must be built using roles, and it must run two or more instances of each of those roles. A Windows Azure application behaves correctly when any role instance fails. If all instances of a particular role fail, an application will stop behaving as it should—this can’t be helped. The requirement to work correctly during partial failures is fundamental to the Win