

U.S. Patent 9,146,777

Claim Chart

U.S. Patent 9,146,777

Claim Chart

Device Shadow

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

- [Reference 1] AWS IoT Core Developer Guide
 [Reference 4] AWS IoT Core Overview
 [Reference 6] Designing MQTT Topics for AWS IoT Core
 [Reference 10] IP Version 4 (IPv4) Datagram General Format
 [Reference 11] IP Version 6 (IPv6) Datagram Main Header Format
 [Reference 12] AWS Management Console Overview
 [Reference 13] Getting Started with AWS IoT Core
 [Reference 20] Amazon IoT Core now supports multiple shadows for a single IoT device

Patent '777 Claim Elements	AWS
1. An apparatus for parallel processing of a large computing requirement, the apparatus comprising:	The apparatus is illustrated on Appendix A.
a central processing unit ("CPU");	"The AWS Management Console brings the unmatched breadth and depth of AWS right to your computer or mobile phone with a secure, easy-to-access, web-based portal" [Reference 12, page 1]
a task pool in electronic communication with the CPU;	"Technicians can use mobile apps to set new desired states [Device Shadows]" [Reference 4, page 1; also see Appendix B] "While devices, apps, and other cloud services are connected to AWS IoT, they can access and control the current state of a device through its shadows. For example, an app can request a change in a device's state by updating a shadow." [Reference 1, page 454] "You can also access AWS IoT through the AWS IoT console, which provides a graphical user interface (GUI) through which you can configure and manage the thing objects, certificates, rules, jobs, policies, and other elements of your IoT solutions." [Reference 1, page 2]
and a first solidarity cell in electronic communication with the task pool, the first solidarity cell comprising a first agent configured to proactively retrieve, from the task pool, without requiring an instruction from the CPU, a matching task for the solidarity cell to process;	"Devices publish MQTT messages that are identified by topics (p. 80) to communicate their state to AWS IoT, and AWS IoT publishes MQTT messages to inform the devices and apps of changes and events." [Reference 1, page 56] "Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device's shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP." [Reference 6, page 10] "The following are the MQTT topics used for interacting with shadows.

	Topics • /get • /get/accepted • /get/rejected • /update • /update/delta • /update/accepted • /update/documents • /update/rejected • /delete • /delete/accepted • /delete/rejected" [Reference 1, page 482] "AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel." [Reference 6, page 1] "Use the AWS IoT Shadow in situations where a command persists for later use, even if the device is currently offline. For example, if a GPS system is sent a new destination through the Shadow desired state but is not immediately reachable, the new coordinates remain in the GPS IoT Shadow. Once the GPS system regains connectivity, it can actively request its last shadow state and retrieve the new coordinates." [Reference 6, page 11] "The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic." [Reference 6, page 19] "A device's shadow document has the following properties: State • desired The desired state of the device. Apps can write to this portion of the document to update the state of a device without having to directly connect to a [sic] it. • reported The reported state of the device. Devices write to this portion of the document to report their new state. Apps read this portion of the document to determine the device's last-reported state." [Reference 1, page 494] "Shadows in apps and services This section describes how an app or service interacts with the AWS IoT Device Shadow service. This example assumes the app or service is interacting only with the shadow and, through the shadow, the device. This example doesn't include any management actions, such as creating or deleting shadows." [Reference 1, page 460]
wherein the CPU populates the task pool by dividing the requirement into one or more threads and placing the threads in the task pool, each thread comprising one or more tasks, and the matching task being one of the tasks;	"Technicians can use mobile apps to set new desired states [Device Shadows]" [Reference 4, page 1; also see Appendix B] "While devices, apps, and other cloud services are connected to AWS IoT, they can access and control the current state of a device through its shadows. For example, an app can request a change in a device's state by updating a shadow." [Reference 1, page 454]

	“AWS IoT Core stores the latest state of a connected device so that it can be read or set at anytime, making the device appear to your applications as if it were online all the time. This means that your application can read a device’s state even when it is disconnected, and also allows you to set a device state and have it implemented when the device reconnects.” [Reference 4, page 1] “IoT devices often have multiple state data such as user settings, manufacturer configuration, and operational status. Each of these state data needs to be accessed by different users or at differing frequencies. Until today, you can only associate one shadow to a device and are required to store all sets of state data in one shadow. As a result, all users must have access to the entire shadow and can consequently read and update state data they shouldn’t. In addition, devices with large state data set may quickly run into the shadow size limit. Now, you can store different device state data into different shadows, and as a result access only the required state data when needed and reduce individual shadow size.” [Reference 20, page 1] “Shadows can make a device’s state available to apps and other services whether the device is connected to AWS IoT or not. AWS IoT thing objects can have multiple named shadows so that your IoT solution has more options for connecting your devices to other apps and services.” [Reference 1, page 454] “IoT devices should not share shadows. To separate commands for each device, make sure that each device has permissions to its own shadow and that devices do not share a single shadow” [Reference 6, page 11] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
wherein each task comprises a descriptor, the descriptor containing at least: a function to be executed;	“Shadows use JSON shadow documents (p. 489) to store and retrieve data.” [Reference 1, page 455] “A device's shadow document has the following properties: State • desired The desired state of the device. Apps can write to this portion of the document to update the state of a device without having to directly connect to a [sic] it. • reported The reported state of the device. Devices write to this portion of the document to report their new state. Apps read this portion of the document to determine the device's last-reported state.” [Reference 1, page 494]
and a memory location of data upon which the function is to be executed;	“In this example, an administrator is sending a command to a wind turbine to change the current angle of the turbine blade to accommodate an approaching wind change. To do so, the administrator first leverages the AWS IoT Shadow as the primary mechanism for issuing commands to a device and as a storage location for essential device readings related to the device.” [Reference 6, page 15] “After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs. In this case, the command is related to credentials. The device also extracts the client ID, the session ID, and the response topic from the command payload.”

	[Reference 6, page 19]
wherein the first agent is a data frame comprising: a source address, a destination address and a payload;	“MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: • Who is the originator of the command? • Who is the intended receiver of the command? • Was the command processed successfully? • What is the current status of the command? • If the command was not processed successfully, what is the error?” [Reference 6, page 13] “In the payload of this request, the IoT application includes a field that denotes where the device (device-1) should send its response and a session identifier for tracking” [Reference 6, page 15] “AWS IoT Core supports devices and clients that use the MQTT and the MQTT over WebSocket Secure (WSS) protocols to publish and subscribe to messages, and devices and clients that use the HTTPS protocol to publish messages. All protocols support IPv4 and IPv6.” [Reference 1, page 72] “The IPv4 datagram is conceptually divided into two pieces: the header and the payload. The header contains addressing and control fields, while the payload carries the actual data to be sent over the internetwork.” [Reference 10, page 1] “Table 56: Internet Protocol Version 4 (IPv4) Datagram Format Source Address: The 32-bit IP address of the originator of the datagram. Destination Address: The 32-bit IP address of the intended recipient of the datagram” [Reference 10, page 2] “Table 68: IPv6 Main Header Format Source Address: The 128-bit IP address of the originator of the datagram. As with IPv4, this is always the device that originally sent the datagram. Destination Address: The 128-bit IP address of the intended recipient of the datagram; unicast, anycast or multicast. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.” [Reference 11, page 2]
wherein the first agent retrieves the matching task by:	“Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device’s shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP.” [Reference 6, page 10] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1]

	“The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
being dispatched by the first solidarity cell to the task pool, during which the source address is the first solidarity cell's address, the destination address is the task pool's address, and the payload comprises a list of functions the first solidarity cell is configured to perform;	“Use the AWS IoT Shadow in situations where a command persists for later use, even if the device is currently offline. For example, if a GPS system is sent a new destination through the Shadow desired state but is not immediately reachable, the new coordinates remain in the GPS IoT Shadow. Once the GPS system regains connectivity, it can actively request its last shadow state and retrieve the new coordinates.” [Reference 6, page 11] “Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device's shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP.” [Reference 6, page 10] “MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: • Who is the originator of the command? • Who is the intended receiver of the command? • Was the command processed successfully? • What is the current status of the command? • If the command was not processed successfully, what is the error?” [Reference 6, page 13] “After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs. In this case, the command is related to credentials. The device also extracts the client ID, the session ID, and the response topic from the command payload. Because a home can have multiple authorized homeowners, the client ID determines which homeowner has requested this change. In this example, the action field includes the type of credentials request, generatepassword, and the associated user for the temporary key. Last, the device obtains the response topic field in the MQTT message and uses this information to publish its response.” [Reference 6, page 19] Also see Appendix C which illustrates a multi-command and control through the use of the Device Shadow.
searching the task pool for a task that is ready to be processed and has a function that the first solidarity cell can perform;	“AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “Processing messages when the device reconnects to AWS IoT When a device with one or more shadows connects to AWS IoT, it should synchronize its state with that of all the shadows that it supports by:

	1. Reading all /update/delta messages received and synchronizing the device state to match. 2. Publishing an /update message with a reported message body that has the device's current state." [Reference 1, page 460]
and returning to the first solidarity cell, during which the source address is the task pool's address, the destination address is the first solidarity cell's address, and the payload comprises the descriptor of the matching task.	"Figure 9: Command sent from the AWS IoT Device Gateway to the smart lock" [Reference 6, page 19; also see Appendix D] "MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: • Who is the originator of the command? • Who is the intended receiver of the command? • Was the command processed successfully? • What is the current status of the command? • If the command was not processed successfully, what is the error?" [Reference 6, page 13] "Shadows use JSON shadow documents (p. 489) to store and retrieve data" [Reference 1, page 455] "The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic." [Reference 6, page 19] "AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel." [Reference 6, page 1]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

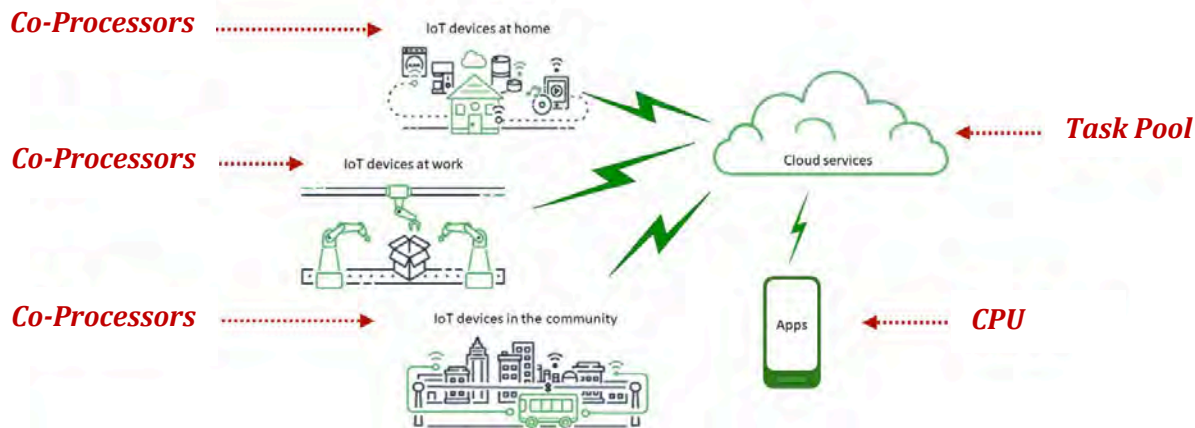
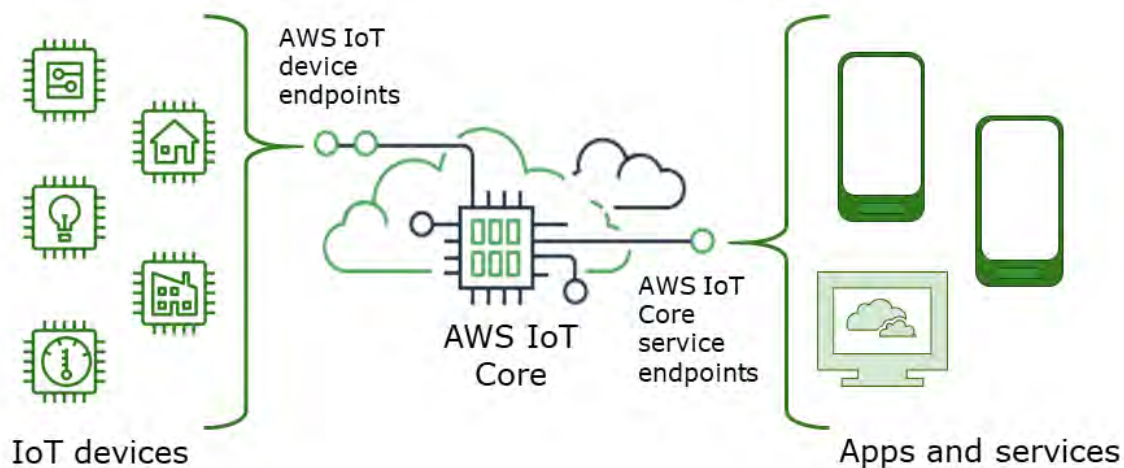


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

Appendix B



Source: Reference 1, page 6

Appendix C

Shadow			
updated as state changes			
HOVERING STATUS	NONE ALERT	STEADY MOTION	OK WIND
			OK VIBRATION LEVEL
Attribute	Units	Reported	Desired
Max Altitude	meters	2	
Max Vertical Speed	meters/second	1	max_vertical_speed ▾
Max Rotation Speed	degrees/second	10	max_rotation_speed ▾
Max Tilt	degrees	20.0	max_tilt ▾
Gimbal Pitch	degrees	-90.0	Downward (-90.0) ▾
Gimbal Speed	degrees	45.0	gimbal_speed ▾
Takeoff Altitude	meters	1	Film (1.0) ▾
Detection Enabled	-	true	detection_enabled ▾
Detection Mode	-	cars	detection_mode ▾
Targeted Car	-	red-car	red-car ▾

Multi-command and control through the use of the Device Shadow.

Drone Zone – AWS re:Invent 2020. Presentation by Jon Slominski, Sr. Prototyping Architect, Envision Engineering Americas, AWS. The figure appears on the Drone Zone presentation, time 10:27

Source: https://virtual.awsevents.com/media/t/1_j5l5wfij (sign in required)

Appendix D

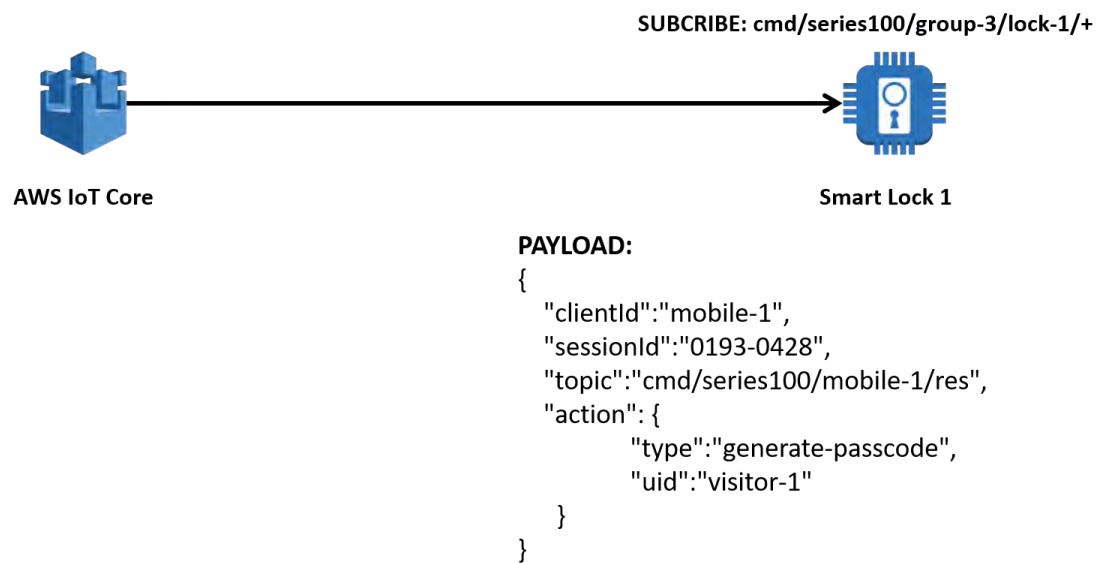


Figure 9: Command sent from the AWS IoT Device Gateway to the smart lock

Source: Reference 6, page 19

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

- [Reference 1] AWS IoT Core Developer Guide
- [Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
2. The apparatus of claim 1 wherein the task pool notifies the CPU when the tasks of a thread are completed.	The apparatus is illustrated on Appendix A. “Devices publish MQTT messages that are identified by topics (p. 80) to communicate their state to AWS IoT, and AWS IoT publishes MQTT messages to inform the devices and apps of changes and events.” [Reference 1, page 56] “Figure 11: Command response sent from AWS IoT Core to mobile client” [Reference 6, page 20; also see Appendix B]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

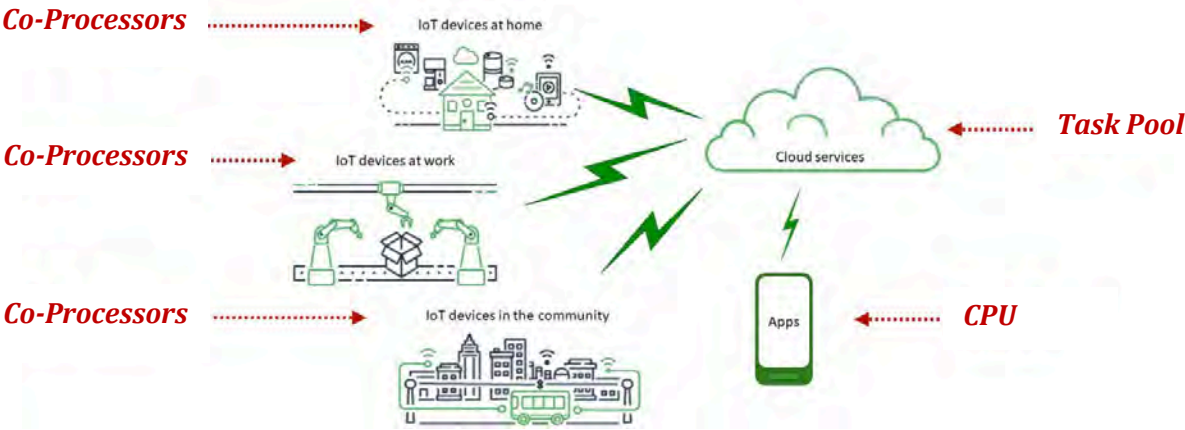


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

Appendix B

SUBSCRIBE: cmd/series100/mobile-1/+



Mobile Client



AWS IoT Core

PAYLOAD:

```
{
  "clientId":"lock-1",
  "sessionId":"0193-0428",
  "passcode":"728e84c6cd8c40ecd47beb5f9cd7f",
  "ttl":"3600",
  "res": {
    "code":"200"
  }
}
```

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

[Reference 20] Amazon IoT Core now supports multiple shadows for a single IoT device

Patent '777 Claim Elements	AWS
3. The apparatus of claim 1 wherein the tasks each comprise a task type selected from a set of task types, and wherein the first solidarity cell is configured to perform tasks of one or more of the task types.	The apparatus is illustrated on Appendix A. “IoT devices should not share shadows. To separate commands for each device, make sure that each device has permissions to its own shadow” [Reference 6, page 11] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic. This topic design also makes it possible for the lock to subscribe using a single level MQTT wildcard after its identifier. The single level wildcard command is backward compatible as the IoT application adds new command types.” [Reference 6, page 19] “Amazon IoT Core is announcing a new feature called named shadow, which allows you to create multiple shadows for a single IoT device. A device’s shadow is a JSON document that is used to store and retrieve state information for a device.” [Reference 20, page 1] “After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs.” [Reference 6, page 19]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

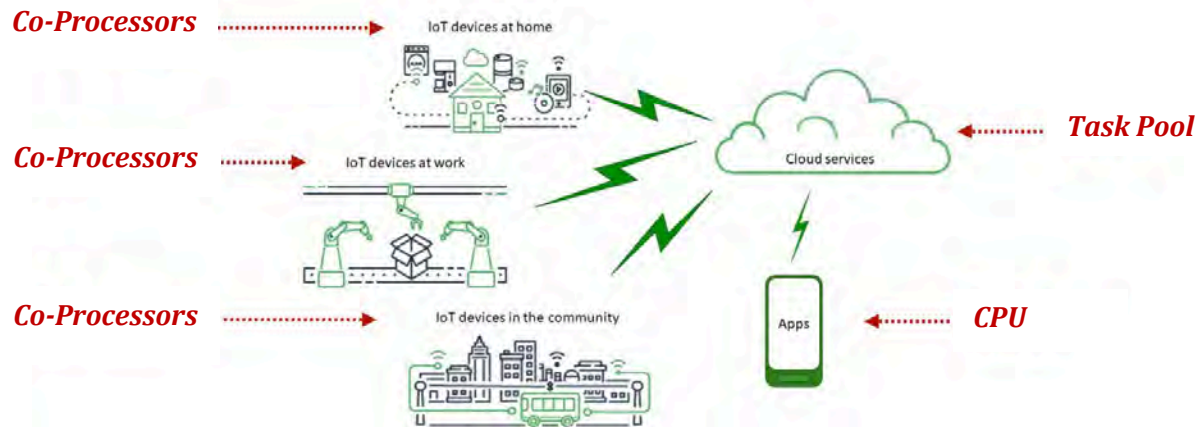


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
4. The apparatus of claim 3 wherein the matching task is a task that is ready to be processed and has a task type that the first solidarity cell can perform.	The apparatus is illustrated on Appendix A. “MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “IoT devices should not share shadows. To separate commands for each device, make sure that each device has permissions to its own shadow” [Reference 6, page 11] “A device's shadow document has the following properties: State • desired The desired state of the device. Apps can write to this portion of the document to update the state of a device without having to directly connect to a [sic] it. • reported The reported state of the device. Devices write to this portion of the document to report their new state. Apps read this portion of the document to determine the device's last-reported state.” [Reference 1, page 494] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic. This topic design also makes it possible for the lock to subscribe using a single level MQTT wildcard after its identifier. The single level wildcard command is backward compatible as the IoT application adds new command types.” [Reference 6, page 19] “After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs.” [Reference 6, page 19]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

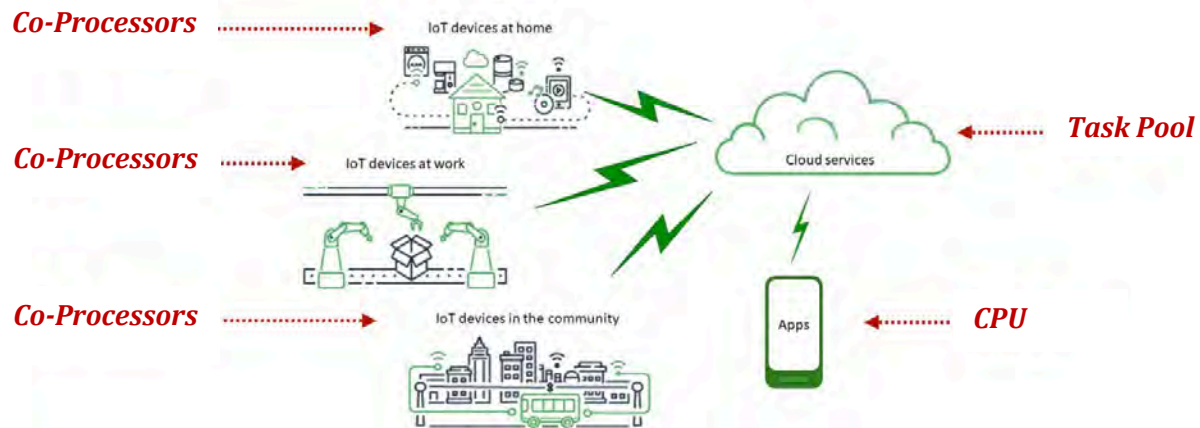


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
5. The apparatus of claim 4 wherein the first agent retrieves the matching task by:	The apparatus is illustrated on Appendix A. “Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device’s shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP.” [Reference 6, page 10] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
searching the task pool for a task that is ready to be processed and has a task type that the first solidarity cell can perform;	“AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “Processing messages when the device reconnects to AWS IoT When a device with one or more shadows connects to AWS IoT, it should synchronize its state with that of all the shadows that it supports by: 1. Reading all /update/delta messages received and synchronizing the device state to match. 2. Publishing an /update message with a reported message body that has the device’s current state.” [Reference 1, page 460] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic. This topic design also makes it possible for the lock to subscribe using a single level MQTT wildcard after its

	identifier. The single level wildcard command is backward compatible as the IoT application adds new command types.” [Reference 6, page 19] “After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs.” [Reference 6, page 19]
and identifying the matching task.	“AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

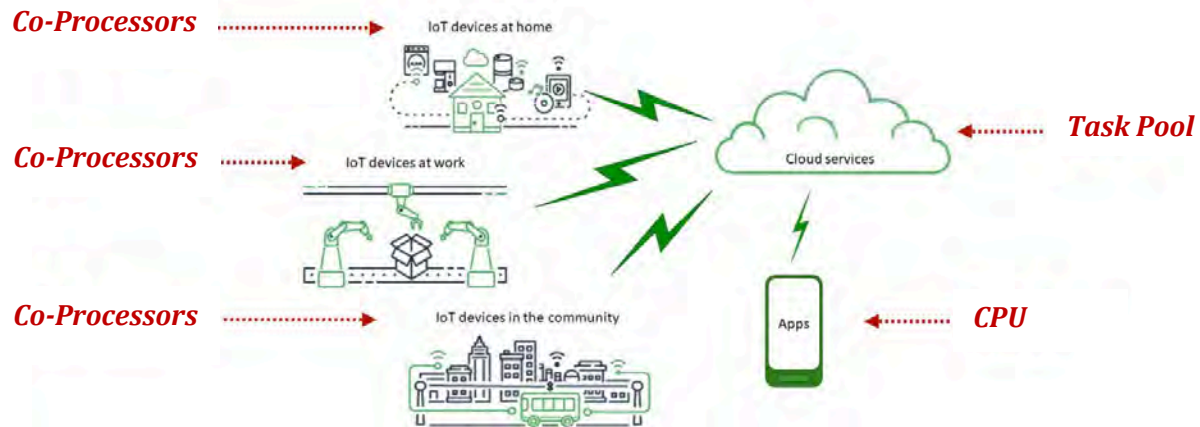


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
6. The apparatus of claim 1 wherein the matching task is a task that is ready to be processed and the function of the matching task can be performed by the first solidarity cell.	The apparatus is illustrated on Appendix A. “MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “Processing messages when the device reconnects to AWS IoT When a device with one or more shadows connects to AWS IoT, it should synchronize its state with that of all the shadows that it supports by: 1. Reading all /update/delta messages received and synchronizing the device state to match. 2. Publishing an /update message with a reported message body that has the device’s current state.” [Reference 1, page 460] “A device's shadow document has the following properties: State • desired The desired state of the device. Apps can write to this portion of the document to update the state of a device without having to directly connect to a [sic] it. • reported The reported state of the device. Devices write to this portion of the document to report their new state. Apps read this portion of the document to determine the device's last-reported state.” [Reference 1, page 494] “After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs. In this case, the command is related to credentials. The device also extracts the client ID, the session ID, and the response topic from the command payload. Because a home can have multiple authorized homeowners, the client ID determines which homeowner has requested this change. In this example, the action field includes the type of credentials request, generatepassword, and the associated user for the temporary key. Last, the device obtains the response topic field in the MQTT message and uses this information to publish its response.”

[Reference 6, page 19]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

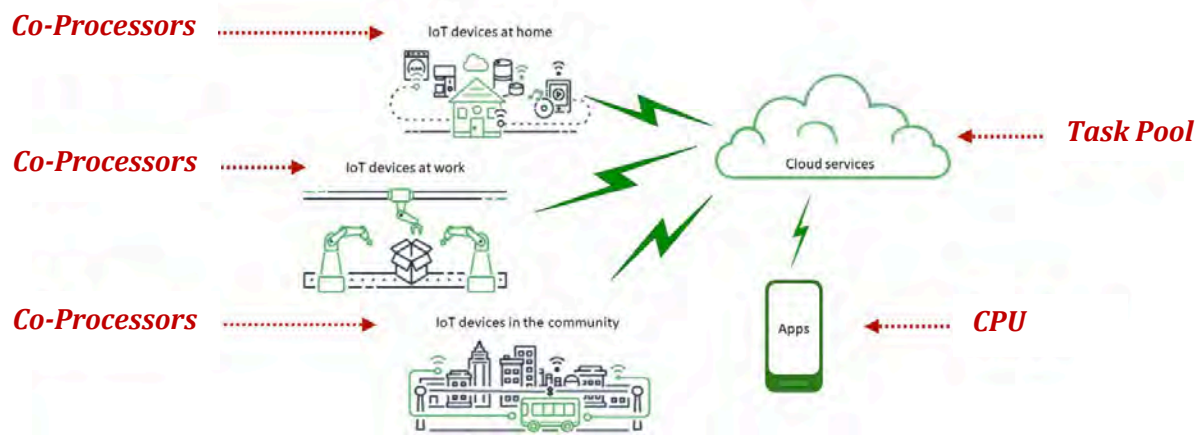


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
7. The apparatus of claim 6 wherein the first agent retrieves the matching task by searching the task pool for a task that is ready to be processed and has a function that the first solidarity cell can perform, and identifying the matching task.	The apparatus is illustrated on Appendix A. “Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device’s shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP.” [Reference 6, page 10] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “Processing messages when the device reconnects to AWS IoT When a device with one or more shadows connects to AWS IoT, it should synchronize its state with that of all the shadows that it supports by: 1. Reading all /update/delta messages received and synchronizing the device state to match. 2. Publishing an /update message with a reported message body that has the device’s current state.” [Reference 1, page 460] “A device’s shadow document has the following properties: State • desired The desired state of the device. Apps can write to this portion of the document to update the state of a device without having to directly connect to a [sic] it. • reported

	The reported state of the device. Devices write to this portion of the document to report their new state. Apps read this portion of the document to determine the device's last-reported state.” [Reference 1, page 494] “After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs. In this case, the command is related to credentials. The device also extracts the client ID, the session ID, and the response topic from the command payload. Because a home can have multiple authorized homeowners, the client ID determines which homeowner has requested this change. In this example, the action field includes the type of credentials request, generatepassword, and the associated user for the temporary key. Last, the device obtains the response topic field in the MQTT message and uses this information to publish its response.” [Reference 6, page 19] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
--	---

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

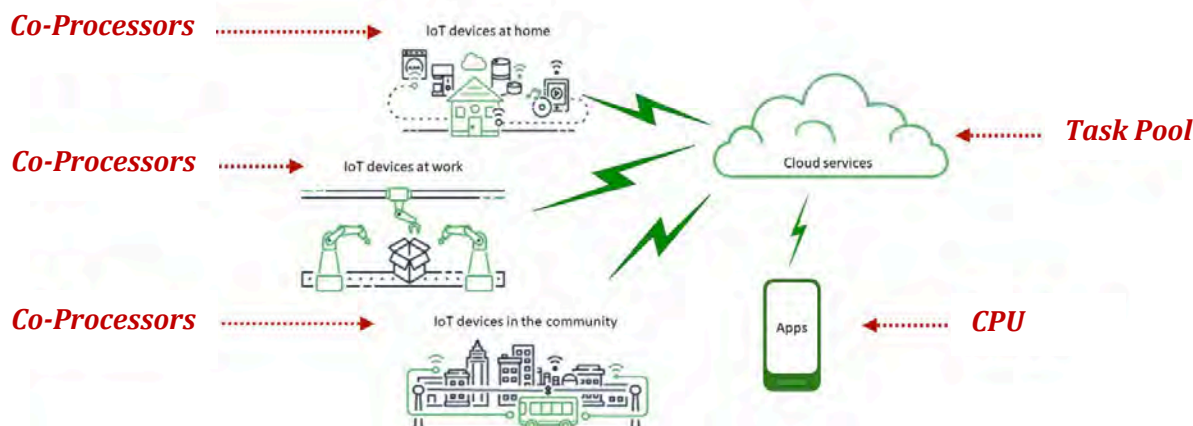


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
8. The apparatus of claim 1 wherein the descriptor further contains a memory location where processed data is to be stored.	The apparatus is illustrated on Appendix A. “Shadows use JSON shadow documents (p. 489) to store and retrieve data” [Reference 1, page 455] “In this example, an administrator is sending a command to a wind turbine to change the current angle of the turbine blade to accommodate an approaching wind change. To do so, the administrator first leverages the AWS IoT Shadow as the primary mechanism for issuing commands to a device and as a storage location for essential device readings [Reference 6, page 15]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

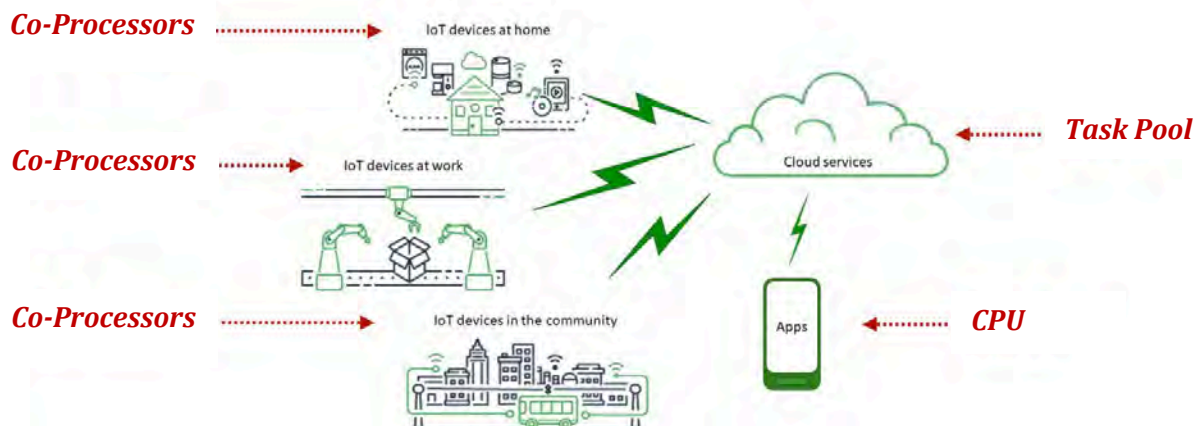


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

[Reference 18] Amazon Redshift Database Developer Guide

Patent '777 Claim Elements	AWS
9. The apparatus of claim 1 wherein the descriptor is a data structure and the task contains a reference to the memory location of the descriptor.	The apparatus is illustrated on Appendix A. “Shadows use JSON shadow documents (p. 489) to store and retrieve data” [Reference 1, page 455] “The JSON data structure is made up of a set of objects or arrays. A JSON object begins and ends with braces, and contains an unordered collection of name-value pairs.” [Reference 18, page 576] “In this example, an administrator is sending a command to a wind turbine to change the current angle of the turbine blade to accommodate an approaching wind change. To do so, the administrator first leverages the AWS IoT Shadow as the primary mechanism for issuing commands to a device and as a storage location for essential device readings related to the device.” [Reference 6, page 15]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

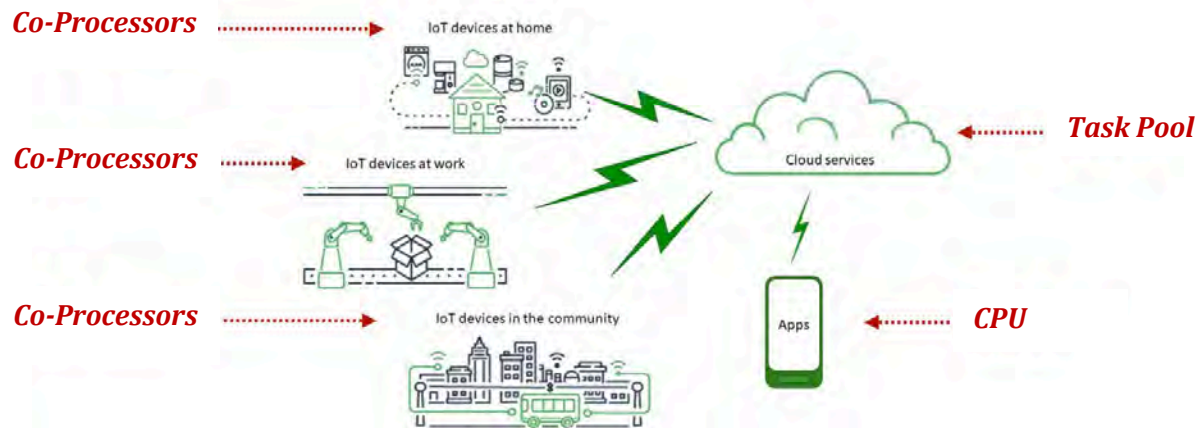


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

[Reference 7] AWS IoT Greengrass Developer Guide

[Reference 19] Introducing AWS IoT SiteWise Edge

Patent '777 Claim Elements	AWS
10. The apparatus of claim 1 wherein the task pool occupies a region of physical memory.	The apparatus is illustrated on Appendix A. “In this example, an administrator is sending a command to a wind turbine to change the current angle of the turbine blade to accommodate an approaching wind change. To do so, the administrator first leverages the AWS IoT Shadow as the primary mechanism for issuing commands to a device and as a storage location for essential device readings related to the device.” [Reference 6, page 15] “At a minimum, AWS IoT SiteWise Edge requires an industrial computer running Linux with a x86 64 bit quad-core processor, 16GB RAM, and 256GB in disk space. In preview, we recommend you use dedicated hardware for use with AWS IoT SiteWise Edge. In preview, AWS IoT SiteWise Edge software runs on AWS IoT Greengrass v1 (v1.10.2 recommended) installed on Ubuntu Server 18.04+ LTS OS for x86.” [Reference 19, page 2-3] “AWS IoT Greengrass uses the spooler system component (the GGCloudSpooler Lambda function) to manage the message queue. You can use the following GGCloudSpooler environment variables to configure storage settings. - GG_CONFIG_STORAGE_TYPE. The location of the message queue. The following are valid values: 0 FileSystem. Store unprocessed messages in the local storage cache on the disk of the physical core device. When the core restarts, queued messages are retained for processing. Messages are removed after they are processed.” [Reference 7, page 79]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

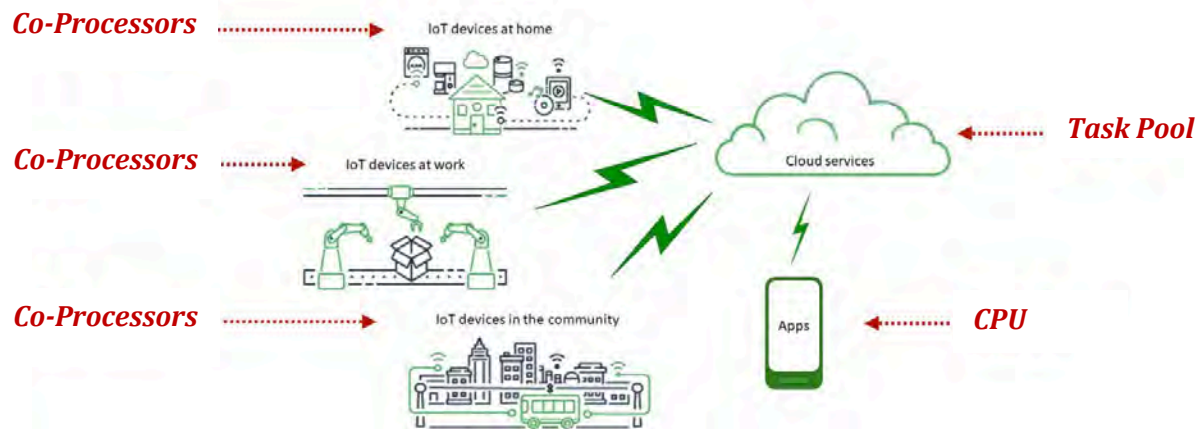


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 7] AWS IoT Greengrass Developer Guide

[Reference 19] Introducing AWS IoT SiteWise Edge

Patent '777 Claim Elements	AWS
11. The apparatus of claim 10 wherein the task pool is disposed in a hardware block dedicated to the task pool.	The apparatus is illustrated on Appendix A. “At a minimum, AWS IoT SiteWise Edge requires an industrial computer running Linux with a x86 64 bit quad-core processor, 16GB RAM, and 256GB in disk space. In preview, we recommend you use dedicated hardware for use with AWS IoT SiteWise Edge. In preview, AWS IoT SiteWise Edge software runs on AWS IoT Greengrass v1 (v1.10.2 recommended) installed on Ubuntu Server 18.04+ LTS OS for x86.” [Reference 19, page 2-3]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

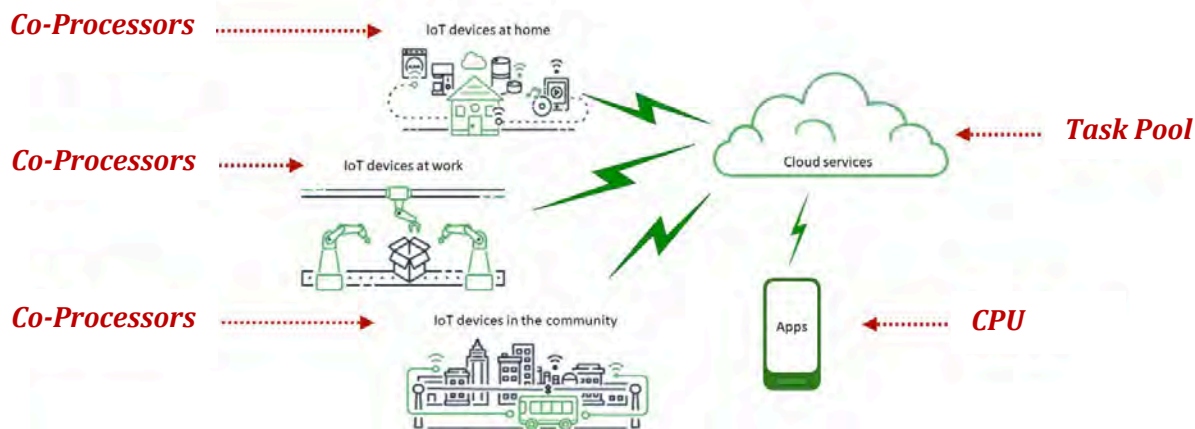


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
12. The apparatus of claim 1 further comprising a second solidarity cell comprising a second agent that proactively retrieves matching tasks from the task pool for the second solidarity cell to process, wherein the matching task for each solidarity cell is a task in the task pool that is ready to be processed and can be performed by the solidarity cell.	The apparatus is illustrated on Appendix A. “Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device’s shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP.” [Reference 6, page 10] “The following are the MQTT topics used for interacting with shadows. Topics • /get • /get/accepted • /get/rejected • /update • /update/delta • /update/accepted • /update/documents • /update/rejected • /delete • /delete/accepted • /delete/rejected” [Reference 1, page 482] “Use the AWS IoT Shadow in situations where a command persists for later use, even if the device is currently offline. For example, if a GPS system is sent a new destination through the Shadow desired state but is not immediately reachable, the new coordinates remain in the GPS IoT Shadow. Once the GPS system regains connectivity, it can actively request its last shadow state and retrieve the new coordinates.” [Reference 6, page 11] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]

	“Processing messages when the device reconnects to AWS IoT When a device with one or more shadows connects to AWS IoT, it should synchronize its state with that of all the shadows that it supports by: 1. Reading all /update/delta messages received and synchronizing the device state to match. 2. Publishing an /update message with a reported message body that has the device’s current state.” [Reference 1, page 460] “A device’s shadow document has the following properties: State • desired The desired state of the device. Apps can write to this portion of the document to update the state of a device without having to directly connect to a [sic] it. • reported The reported state of the device. Devices write to this portion of the document to report their new state. Apps read this portion of the document to determine the device’s last-reported state.” [Reference 1, page 494] “After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs.” [Reference 6, page 19]
--	---

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

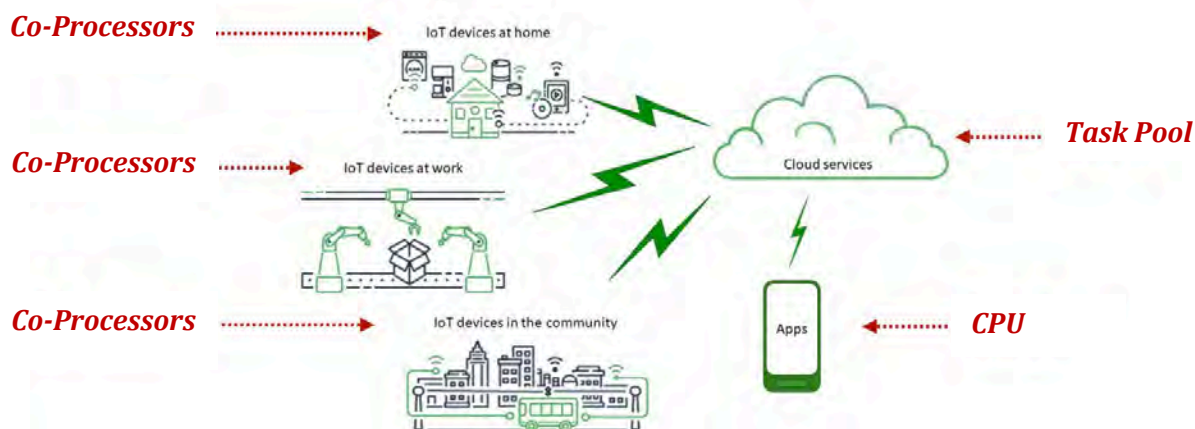


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
13. The apparatus of claim 12 wherein each solidarity cell sends its agent to the task pool when the solidarity cell does not have a matching task to process.	The apparatus is illustrated on Appendix A. “Use the AWS IoT Shadow in situations where a command persists for later use, even if the device is currently offline. For example, if a GPS system is sent a new destination through the Shadow desired state but is not immediately reachable, the new coordinates remain in the GPS IoT Shadow. Once the GPS system regains connectivity, it can actively request its last shadow state and retrieve the new coordinates.” [Reference 6, page 11] “Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device’s shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP.” [Reference 6, page 10] “The following are the MQTT topics used for interacting with shadows. Topics • /get • /get/accepted • /get/rejected • /update • /update/delta • /update/accepted • /update/documents • /update/rejected • /delete • /delete/accepted • /delete/rejected” [Reference 1, page 482] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “iot:GetThingShadow Represents the permission to retrieve a device's shadow. The iot:GetThingShadow permission is checked every time a request is made to retrieve the shadow's contents” [Reference 1, page 218]

	“After the device updates the state to match the desired state in the message, it must send the new reported state back to AWS IoT by publishing an update message. This procedure simulates this in the MQTT client.” [Reference 1, page 467] “When a device goes offline, an app can still communicate with AWS IoT and the device's shadows. When the device reconnects, it receives the current state of its shadows so that it can update its state to match that of its shadows, and then publish a message with its updated state. Likewise, when an app goes offline and the device state changes while it's offline, the device keeps the shadow updated so the app can query the shadows for its current state when it reconnects.” [Reference 1, page 454]
--	--

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

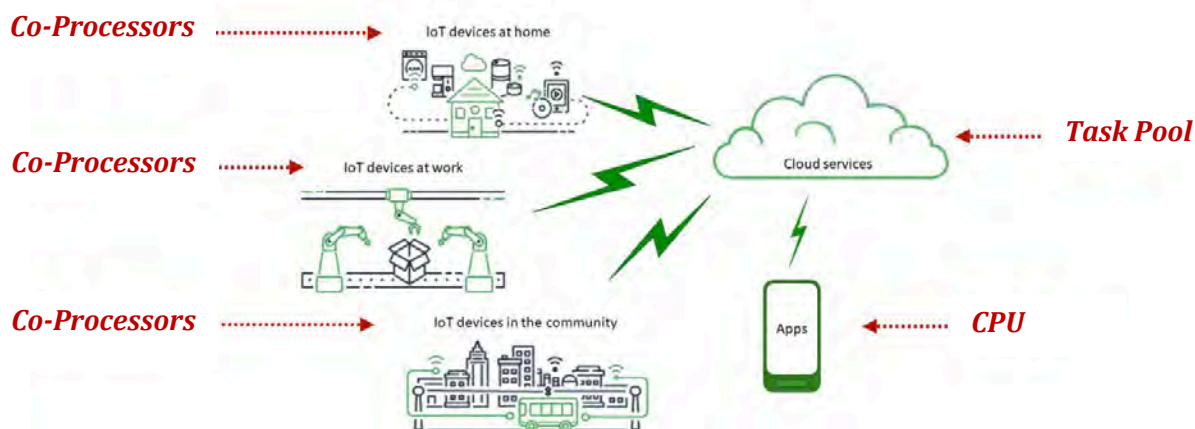


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent ‘777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide
 [Reference 6] Designing MQTT Topics for AWS IoT Core
 [Reference 10] IP Version 4 (IPv4) Datagram General Format
 [Reference 11] IP Version 6 (IPv6) Datagram Main Header Format
 [Reference 12] AWS Management Console Overview
 [Reference 13] Getting Started with AWS IoT Core

Patent '777 Claim Elements	AWS
14. An apparatus for parallel processing of a large computing requirement, the apparatus comprising:	The apparatus is illustrated on Appendix A.
a central processing unit ("CPU");	"The AWS Management Console brings the unmatched breadth and depth of AWS right to your computer or mobile phone with a secure, easy-to-access, web-based portal" [Reference 12, page 1]
a task pool in electronic communication with the CPU;	"Technicians can use mobile apps to set new desired states [Device Shadows]" [Reference 4, page 1; also see Appendix B] "While devices, apps, and other cloud services are connected to AWS IoT, they can access and control the current state of a device through its shadows. For example, an app can request a change in a device's state by updating a shadow." [Reference 1, page 454] "You can also access AWS IoT through the AWS IoT console, which provides a graphical user interface (GUI) through which you can configure and manage the thing objects, certificates, rules, jobs, policies, and other elements of your IoT solutions." [Reference 1, page 2]
first solidarity cell in electronic communication with the task pool, the first solidarity cell comprising a first agent configured to proactively retrieve, from the task pool, a matching task for the solidarity cell to process;	"Devices publish MQTT messages that are identified by topics (p. 80) to communicate their state to AWS IoT, and AWS IoT publishes MQTT messages to inform the devices and apps of changes and events." [Reference 1, page 56] "Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device's shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP." [Reference 6, page 10] "The following are the MQTT topics used for interacting with shadows. Topics /get

	• /get/accepted • /get/rejected • /update • /update/delta • /update/accepted • /update/documents • /update/rejected • /delete • /delete/accepted • /delete/rejected" [Reference 1, page 482] "AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel." [Reference 6, page 1] "Use the AWS IoT Shadow in situations where a command persists for later use, even if the device is currently offline. For example, if a GPS system is sent a new destination through the Shadow desired state but is not immediately reachable, the new coordinates remain in the GPS IoT Shadow. Once the GPS system regains connectivity, it can actively request its last shadow state and retrieve the new coordinates." [Reference 6, page 11] "The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic." [Reference 6, page 19]
and a second solidarity cell comprising a second agent that proactively retrieves matching tasks from the task pool for the second solidarity cell to process, wherein the matching task for each solidarity cell is a task in the task pool that is ready to be processed and can be performed by the solidarity cell;	"Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device's shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP." [Reference 6, page 10] "The following are the MQTT topics used for interacting with shadows. Topics • /get • /get/accepted • /get/rejected • /update • /update/delta • /update/accepted • /update/documents • /update/rejected • /delete • /delete/accepted • /delete/rejected" [Reference 1, page 482] "Use the AWS IoT Shadow in situations where a command persists for later use, even if the device is currently offline. For example, if a GPS system is sent a new destination through the Shadow desired state but is not immediately reachable, the new coordinates remain in the GPS IoT Shadow. Once the GPS system regains connectivity, it can actively request its last shadow state and retrieve the new coordinates." [Reference 6, page 11]

	“AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “Processing messages when the device reconnects to AWS IoT When a device with one or more shadows connects to AWS IoT, it should synchronize its state with that of all the shadows that it supports by: 1. Reading all /update/delta messages received and synchronizing the device state to match. 2. Publishing an /update message with a reported message body that has the device’s current state.” [Reference 1, page 460] “A device’s shadow document has the following properties: State • desired The desired state of the device. Apps can write to this portion of the document to update the state of a device without having to directly connect to a [sic] it. • reported The reported state of the device. Devices write to this portion of the document to report their new state. Apps read this portion of the document to determine the device’s last-reported state.” [Reference 1, page 494] “After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs.” [Reference 6, page 19]
wherein each solidarity cell sends its agent to the task pool when the solidarity cell does not have a matching task to process;	“Use the AWS IoT Shadow in situations where a command persists for later use, even if the device is currently offline. For example, if a GPS system is sent a new destination through the Shadow desired state but is not immediately reachable, the new coordinates remain in the GPS IoT Shadow. Once the GPS system regains connectivity, it can actively request its last shadow state and retrieve the new coordinates.” [Reference 6, page 11] “Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device’s shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP.” [Reference 6, page 10] “The following are the MQTT topics used for interacting with shadows. Topics • /get • /get/accepted • /get/rejected • /update

	• /update/delta • /update/accepted • /update/documents • /update/rejected • /delete • /delete/accepted • /delete/rejected” [Reference 1, page 482] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “iot:GetThingShadow Represents the permission to retrieve a device's shadow. The iot:GetThingShadow permission is checked every time a request is made to retrieve the shadow's contents” [Reference 1, page 218] “After the device updates the state to match the desired state in the message, it must send the new reported state back to AWS IoT by publishing an update message. This procedure simulates this in the MQTT client.” [Reference 1, page 467] “When a device goes offline, an app can still communicate with AWS IoT and the device's shadows. When the device reconnects, it receives the current state of its shadows so that it can update its state to match that of its shadows, and then publish a message with its updated state. Likewise, when an app goes offline and the device state changes while it's offline, the device keeps the shadow updated so the app can query the shadows for its current state when it reconnects.” [Reference 1, page 454]
and wherein each agent comprises a source address, a destination address, and a payload, and wherein each agent retrieves a matching task by:	“MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: • Who is the originator of the command? • Who is the intended receiver of the command? • Was the command processed successfully? • What is the current status of the command? • If the command was not processed successfully, what is the error?” [Reference 6, page 13] “In the payload of this request, the IoT application includes a field that denotes where the device (device-1) should send its response and a session identifier for tracking” [Reference 6, page 15] “AWS IoT Core supports devices and clients that use the MQTT and the MQTT over WebSocket Secure (WSS) protocols to publish and subscribe to messages, and devices and clients that use the HTTPS protocol to publish messages. All protocols support IPv4 and IPv6.” [Reference 1, page 72] “Table 56: Internet Protocol Version 4 (IPv4) Datagram Format

	Source Address: The 32-bit IP address of the originator of the datagram. Destination Address: The 32-bit IP address of the intended recipient of the datagram” [Reference 10, page 2] “Table 68: IPv6 Main Header Format Source Address: The 128-bit IP address of the originator of the datagram. As with IPv4, this is always the device that originally sent the datagram. Destination Address: The 128-bit IP address of the intended recipient of the datagram; unicast, anycast or multicast. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.” [Reference 11, page 2] “Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device’s shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP.” [Reference 6, page 10] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
being dispatched by its solidarity cell to the task pool, during which the source address is its solidarity cell’s address, the destination address is the task pool’s address, and the payload comprises a list of functions the agent’s solidarity cell is configured to perform;	“Use the AWS IoT Shadow in situations where a command persists for later use, even if the device is currently offline. For example, if a GPS system is sent a new destination through the Shadow desired state but is not immediately reachable, the new coordinates remain in the GPS IoT Shadow. Once the GPS system regains connectivity, it can actively request its last shadow state and retrieve the new coordinates.” [Reference 6, page 11] “Using the AWS IoT Shadow for Commands The Device Shadow service acts as a state intermediary, allowing devices and applications to retrieve and update a device’s shadow state. You can use the shadow to get and set the state of a device over MQTT or HTTP.” [Reference 6, page 10] “MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: • Who is the originator of the command? • Who is the intended receiver of the command? • Was the command processed successfully? • What is the current status of the command? • If the command was not processed successfully, what is the error?” [Reference 6, page 13]

	“After receiving the MQTT payload, the smart lock parses the command request to determine what type of action it runs. In this case, the command is related to credentials. The device also extracts the client ID, the session ID, and the response topic from the command payload. Because a home can have multiple authorized homeowners, the client ID determines which homeowner has requested this change. In this example, the action field includes the type of credentials request, generatepassword, and the associated user for the temporary key. Last, the device obtains the response topic field in the MQTT message and uses this information to publish its response.” [Reference 6, page 19] Also see Appendix C which illustrates a multi-command and control through the use of the Device Shadow.
searching the task pool for a task that is ready to be processed and has a function that the agent's solidarity cell can perform;	“AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “Processing messages when the device reconnects to AWS IoT When a device with one or more shadows connects to AWS IoT, it should synchronize its state with that of all the shadows that it supports by: 3. Reading all /update/delta messages received and synchronizing the device state to match. 4. Publishing an /update message with a reported message body that has the device's current state.” [Reference 1, page 460]
and returning to its solidarity cell, during which the source address is the task pool's address, the destination address is the agent's solidarity cell's address, and the payload comprises a descriptor of the matching task.	“Figure 9: Command sent from the AWS IoT Device Gateway to the smart lock” [Reference 6, page 19; also see Appendix D] “MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: • Who is the originator of the command? • Who is the intended receiver of the command? • Was the command processed successfully? • What is the current status of the command? • If the command was not processed successfully, what is the error?” [Reference 6, page 13] “Shadows use JSON shadow documents (p. 489) to store and retrieve data” [Reference 1, page 455] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between

	publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1]
--	---

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

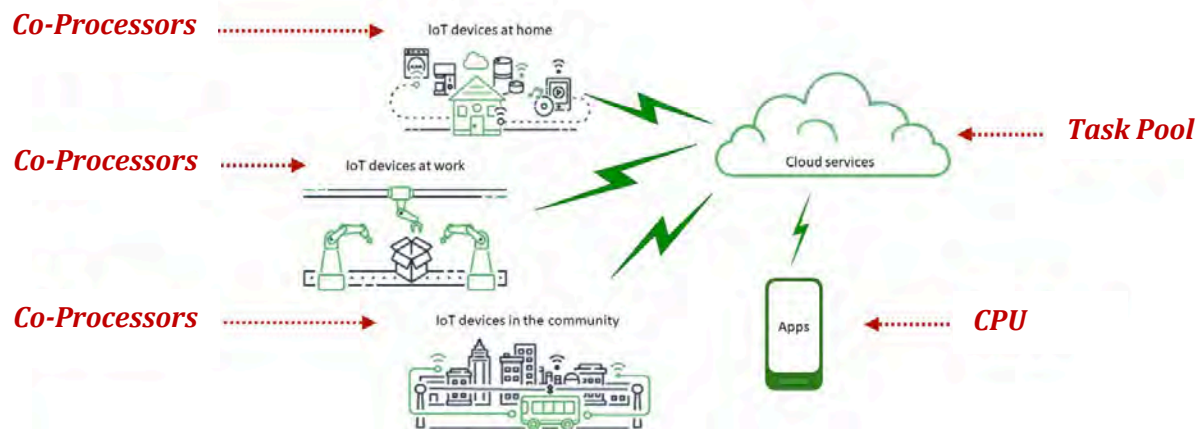
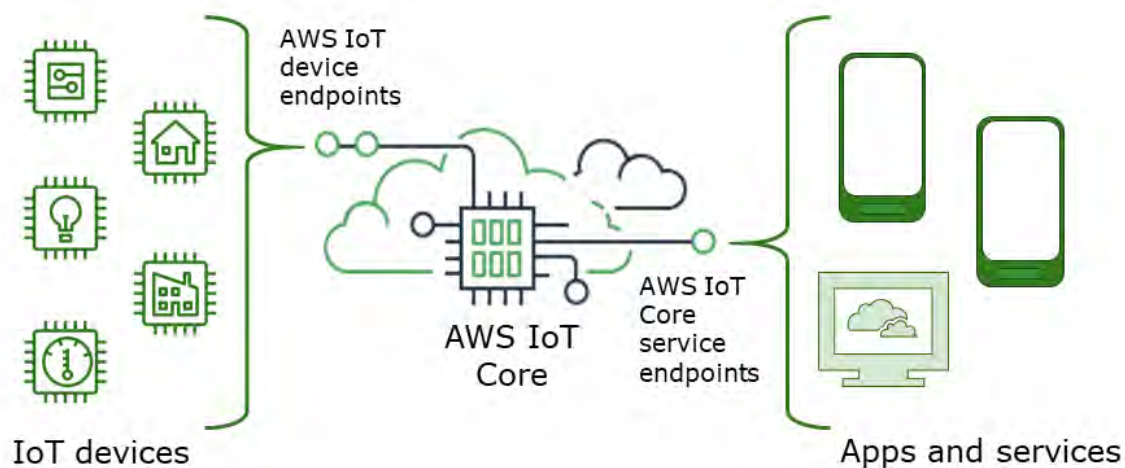


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

Appendix B



Source: Reference 1, page 6

Appendix C

Shadow				
updated as state changes				
HOVERING STATUS	NONE ALERT	STEADY MOTION	OK WIND	OK VIBRATION LEVEL
Attribute	Units	Reported	Desired	
Max Altitude	meters	2		
Max Vertical Speed	meters/second	1	max_vertical_speed	▼
Max Rotation Speed	degrees/second	10	max_rotation_speed	▼
Max Tilt	degrees	20.0	max_tilt	▼
Gimbal Pitch	degrees	-90.0	Downward (-90.0)	▼
Gimbal Speed	degrees	45.0	gimbal_speed	▼
Takeoff Altitude	meters	1	Film (1.0)	▼
Detection Enabled	-	true	detection_enabled	▼
Detection Mode	-	cars	detection_mode	▼
Targeted Car	-	red-car	red-car	▼

Multi-command and control through the use of the Device Shadow.

Drone Zone – AWS re:Invent 2020. Presentation by Jon Slominski, Sr. Prototyping Architect, Envision Engineering Americas, AWS. The figure appears on the Drone Zone presentation, time 10:27

Source: https://virtual.awsevents.com/media/t/1_j5l5wfij (sign in required)

Appendix D

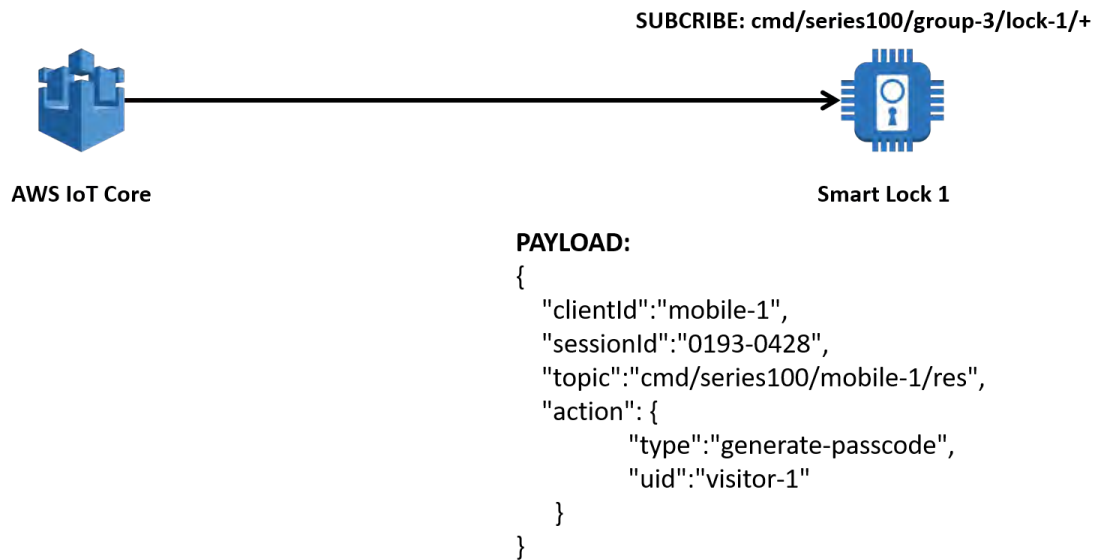


Figure 9: Command sent from the AWS IoT Device Gateway to the smart lock

Source: Reference 6, page 19

U.S. Patent 9,146,777

Claim Chart

Jobs

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

- [Reference 1] AWS IoT Core Developer Guide
 [Reference 5] AWS IoT Core Features
 [Reference 6] Designing MQTT Topics for AWS IoT Core
 [Reference 10] IP Version 4 (IPv4) Datagram General Format
 [Reference 11] IP Version 6 (IPv6) Datagram Main Header Format
 [Reference 12] AWS Management Console Overview
 [Reference 13] Getting Started with AWS IoT Core

Patent '777 Claim Elements	AWS
1. An apparatus for parallel processing of a large computing requirement, the apparatus comprising:	The apparatus is illustrated on Appendix A.
a central processing unit (" CPU ");	"The AWS Management Console brings the unmatched breadth and depth of AWS right to your computer or mobile phone with a secure, easy-to-access, web-based portal" [Reference 12, page 1] "You can use the AWS IoT console, the Jobs HTTPS API, the AWS Command Line Interface, or the AWS SDKs to create and manage jobs." [Reference 1, page 501]
a task pool in electronic communication with the CPU ;	"The AWS Management Console provides a web-based interface for accessing and managing all your AWS IoT resources. Programmatic access to AWS IoT Core is enabled by the AWS CLI and AWS SDK." [Reference 13, page 1] "Job management and control data types The following data types are used by management and control applications to communicate with the AWS IoT Jobs service." [Reference 1, page 570]
and a first solidarity cell in electronic communication with the task pool , the first solidarity cell comprising a first agent configured to proactively retrieve, from the task pool , without requiring an instruction from the CPU , a matching task for the solidarity cell to process;	"Using the MQTT protocol Communication between the AWS IoT Jobs service and your devices can occur over the MQTT protocol. Devices subscribe to MQTT topics to be notified of new jobs and to receive responses from the AWS IoT Jobs service. Devices publish on MQTT topics to query or update the state of a job execution. Each device has its own general MQTT topic." [Reference 1, page 511] "Fleet provisioning for AWS IoT Core makes it easy to onboard any number of manufactured devices to the cloud at scale. It provides an end-to-end managed device onboarding experience that sets up devices with unique digital identities and performs the

	device-side and cloud-side configuration needed for each device to connect and operate with AWS IoT automatically upon its first connection to AWS IoT Core” [Reference 5, page 1] “Devices can: ... • Retrieve the next pending job execution by calling the DescribeJobExecution (p. 586) API with jobId \$next.” [Reference 1, page 512] “Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails.” [Reference 1, page 514] “MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “Devices connect to the AWS IoT Core so they can send data to and receive data from AWS IoT services and other devices. Apps and other services also connect to AWS IoT Core to control and manage the IoT devices and process the data from your IoT solution.” [Reference 1, page 61; also see Appendix B]
wherein the CPU populates the task pool by dividing the requirement into one or more threads and placing the threads in the task pool, each thread comprising one or more tasks, and the matching task being one of the tasks;	“You can use the AWS IoT console, the Jobs HTTPS API, the AWS Command Line Interface, or the AWS SDKs to create and manage jobs.” [Reference 1, page 501] “To create a job 1. Browse to the AWS IoT console. 2. In the navigation pane, choose Manage, and then choose Jobs. 3. Choose Create a job. 4. Choose Create a custom job. 5. Enter an alphanumeric ID for your job and an optional description. Note We do not recommend using personally identifiable information in your job IDs or descriptions. 6. Select the device or device groups that you want to update. 7. Under Add a job file, choose Select, and then select your job document. 8. Choose Sign image for me. If you are not code signing your update, you can skip this step. 9. Create or choose a code-signing profile. If you are not code signing your update, you can skip this step. 10. Under Pre-sign resource URLs, choose I want to pre-sign my URLs and have configured my job file. If you are not code signing your update, you can skip this step. 11. Choose a role and an expiry time for the presigned URL.

	12. Under Job type, choose the appropriate option for your update, and then choose Next. 13. Specify values for any advanced configurations, and then choose Create. After you create the job, the console generates a JSON signature and places it in your job document.” [Reference 1, page 503] “Use staged rollouts to deploy commands using Device Jobs. Device Jobs are the ideal solution for delivering fleet-wide operations to devices. Create multiple smaller deployments first to subsets of the fleet, letting the devices apply your changes, and then rolling out the commands to a greater number of devices over time. By allowing changes to progress over weeks and months, you can have more confidence that there are fewer unforeseen issues and you can more quickly react if there is an issue during an earlier rollout.” [Reference 6, page 12] “AWS IoT Jobs is a service that allows you to define a set of remote operations that are sent to and executed on one or more things connected to AWS IoT. For command use cases, Jobs allows applications to run tasks that require executing multiple steps. An AWS IoT Job contains instructions that the thing must run to complete its transaction. AWS IoT Jobs are the recommended feature for fleet-wide operational tasks, such as software updates, that are only executed by trusted administrators of the entire IoT application.” [Reference 6, page 12] “MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
wherein each task comprises a descriptor, the descriptor containing at least: a function to be executed;	“To create a job, you must first create a job document that is a description of the remote operations to be performed by the devices. Job documents are UTF-8 encoded JSON documents and should contain information that your devices need to perform a job. A job document contains one or more URLs where the device can download an update or some other data.” [Reference 1, page 499]
and a memory location of data upon which the function is to be executed;	“To create a job, you must first create a job document that is a description of the remote operations to be performed by the devices. Job documents are UTF-8 encoded JSON documents and should contain information that your devices need to perform a job. A job document contains one or more URLs where the device can download an update or some other data.” [Reference 1, page 499]
wherein the first agent is a data frame comprising: a source address, a destination address and a payload;	“MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: • Who is the originator of the command? • Who is the intended receiver of the command?

	• Was the command processed successfully? • What is the current status of the command? • If the command was not processed successfully, what is the error?" [Reference 6, page 13] "In the payload of this request, the IoT application includes a field that denotes where the device (device-1) should send its response and a session identifier for tracking" [Reference 6, page 15] "AWS IoT Core supports devices and clients that use the MQTT and the MQTT over WebSocket Secure (WSS) protocols to publish and subscribe to messages, and devices and clients that use the HTTPS protocol to publish messages. All protocols support IPv4 and IPv6." [Reference 1, page 72] "The IPv4 datagram is conceptually divided into two pieces: the header and the payload. The header contains addressing and control fields, while the payload carries the actual data to be sent over the internetwork." [Reference 10, page 1] "Table 56: Internet Protocol Version 4 (IPv4) Datagram Format Source Address: The 32-bit IP address of the originator of the datagram. Destination Address: The 32-bit IP address of the intended recipient of the datagram" [Reference 10, page 2] "Table 68: IPv6 Main Header Format Source Address: The 128-bit IP address of the originator of the datagram. As with IPv4, this is always the device that originally sent the datagram. Destination Address: The 128-bit IP address of the intended recipient of the datagram; unicast, anycast or multicast. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination." [Reference 11, page 2]
wherein the first agent retrieves the matching task by:	"Devices can: ... • Retrieve the next pending job execution by calling the DescribeJobExecution (p. 586) API with jobId \$next." [Reference 1, page 512] "Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails." [Reference 1, page 514] "thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target." [Reference 1, page 561]

	“AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
being dispatched by the first solidarity cell to the task pool, during which the source address is the first solidarity cell's address, the destination address is the task pool's address, and the payload comprises a list of functions the first solidarity cell is configured to perform;	“Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. 5. Call the UpdateJobExecution (p. 591) MQTT API to update the job status. If the includeJobDocument field is set to true in this command, the device can skip the previous step and retrieve the job document at this point. 6. Optionally, you can add a step timer by setting a value for stepTimeoutInMinutes when you call UpdateJobExecution (p. 591). 7. Perform the actions specified by the job document using the UpdateJobExecution (p. 591) MQTT API to report on the progress of the job.” [Reference 1, page 514] “MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: 1. Who is the originator of the command? 2. Who is the intended receiver of the command? 3. Was the command processed successfully? 4. What is the current status of the command? 5. If the command was not processed successfully, what is the error?” [Reference 6, page 13] “AWS IoT Jobs is a service that allows you to define a set of remote operations that are sent to and executed on one or more things connected to AWS IoT. For command use cases, Jobs allows applications to run tasks that require executing multiple steps. An AWS IoT Job contains instructions that the thing must run to complete its transaction. AWS IoT Jobs are the recommended feature for fleet-wide operational tasks, such as software updates, that are only executed by trusted administrators of the entire IoT application.” [Reference 6, page 12] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target.”

	[Reference 1, page 499]
searching the task pool for a task that is ready to be processed and has a function that the first solidarity cell can perform;	“thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. 5. Call the UpdateJobExecution (p. 591) MQTT API” [Reference 1, page 514] “The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT.” [Reference 1, page 499]
and returning to the first solidarity cell, during which the source address is the task pool's address, the destination address is the first solidarity cell's address, and the payload comprises the descriptor of the matching task.	“Option b: Pick from available jobs 6. When a device first comes online, it should subscribe to the thing's notify topic. 7. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 8. If the list contains one or more job executions, pick one. 9. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. 10. Call the UpdateJobExecution (p. 591) MQTT API” [Reference 1, page 514] “The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT.” [Reference 1, page 499] “MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: • Who is the originator of the command? • Who is the intended receiver of the command? • Was the command processed successfully? • What is the current status of the command? • If the command was not processed successfully, what is the error?” [Reference 6, page 13]

	“To create a job, you must first create a job document that is a description of the remote operations to be performed by the devices. Job documents are UTF-8 encoded JSON documents and should contain information that your devices need to perform a job. A job document contains one or more URLs where the device can download an update or some other data.” [Reference 1, page 499] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561]
--	--

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

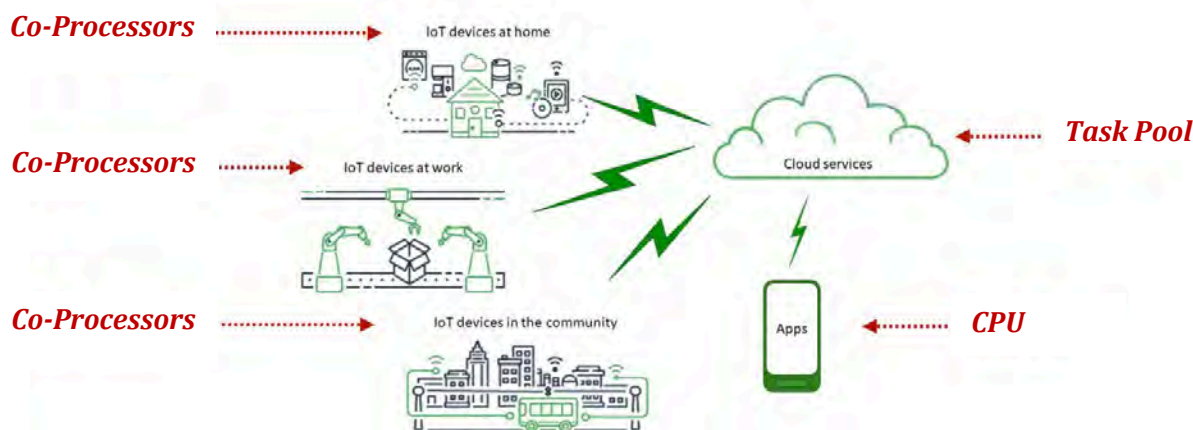
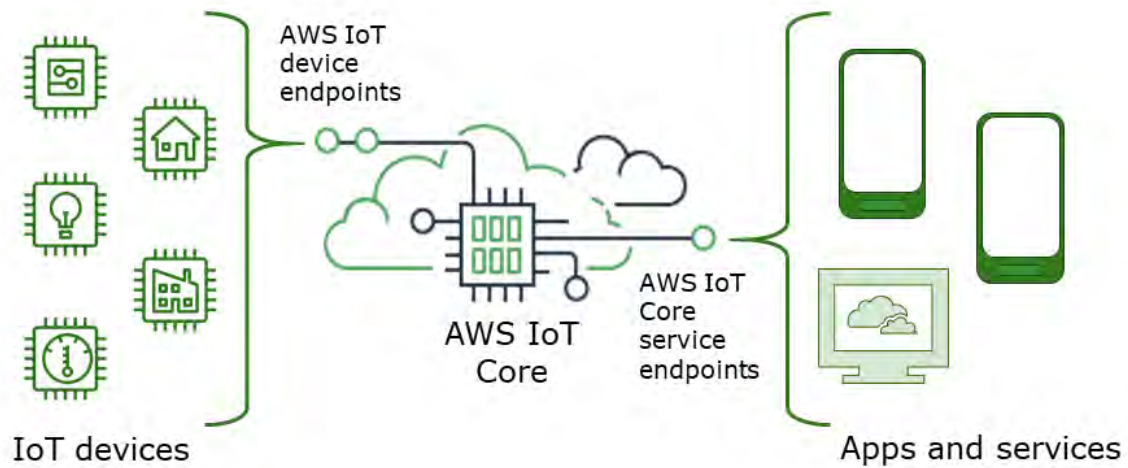


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

Appendix B



Source: Reference 1, page 61

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
2. The apparatus of claim 1 wherein the task pool notifies the CPU when the tasks of a thread are completed.	The apparatus is illustrated on Appendix A. “Devices publish MQTT messages that are identified by topics (p. 80) to communicate their state to AWS IoT, and AWS IoT publishes MQTT messages to inform the devices and apps of changes and events.” [Reference 1, page 56] “The AWS IoT Jobs service publishes to reserved topics on the MQTT protocol when jobs are pending, completed, or canceled, and when a device reports success or failure when executing a job. Devices or management and monitoring applications can keep track of the status of jobs by subscribing to these topics.” [Reference 1, page 857] “Figure 11: Command response sent from AWS IoT Core to mobile client” [Reference 6, page 20; also see Appendix B]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.



Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

Appendix B

SUBSCRIBE: cmd/series100/mobile-1/+



Mobile Client



AWS IoT Core

PAYLOAD:

```
{
  "clientId":"lock-1",
  "sessionId":"0193-0428",
  "passcode":"728e84c6cd8c40ecd47beb5f9cd7f",
  "ttl":"3600",
  "res": {
    "code":"200"
  }
}
```

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
3. The apparatus of claim 1 wherein the tasks each comprise a task type selected from a set of task types, and wherein the first solidarity cell is configured to perform tasks of one or more of the task types.	The apparatus is illustrated on Appendix A. “Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. 5. Call the UpdateJobExecution (p. 591) MQTT API to update the job status. If the includeJobDocument field is set to true in this command, the device can skip the previous step and retrieve the job document at this point. 6. Optionally, you can add a step timer by setting a value for stepTimeoutInMinutes when you call UpdateJobExecution (p. 591). 7. Perform the actions specified by the job document using the UpdateJobExecution (p. 591) MQTT API to report on the progress of the job.” [Reference 1, page 514] “MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target.” [Reference 1, page 499] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic. This topic design also makes it possible for the lock to subscribe using a single level MQTT wildcard after its identifier. The single level wildcard command is backward compatible as the IoT application adds new command types.” [Reference 6, page 19]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

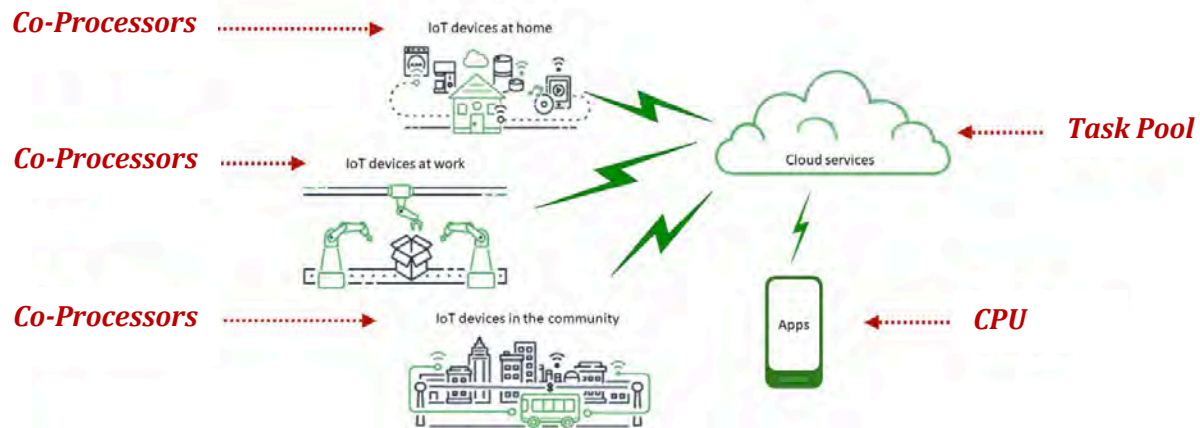


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
4. The apparatus of claim 3 wherein the matching task is a task that is ready to be processed and has a task type that the first solidarity cell can perform.	The apparatus is illustrated on Appendix A. “Option b: Pick from available jobs - When a device first comes online, it should subscribe to the thing's notify topic. - Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. - If the list contains one or more job executions, pick one. - Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. - Call the UpdateJobExecution (p. 591) MQTT API to update the job status. If the includeJobDocument field is set to true in this command, the device can skip the previous step and retrieve the job document at this point. - Optionally, you can add a step timer by setting a value for stepTimeoutInMinutes when you call UpdateJobExecution (p. 591). - Perform the actions specified by the job document using the UpdateJobExecution (p. 591) MQTT API to report on the progress of the job.” [Reference 1, page 514] “MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic. This topic design also makes it possible for the lock to subscribe using a single level MQTT wildcard after its identifier. The single level wildcard command is backward compatible as the IoT application adds new command types.” [Reference 6, page 19] “A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target.” [Reference 1, page 499]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

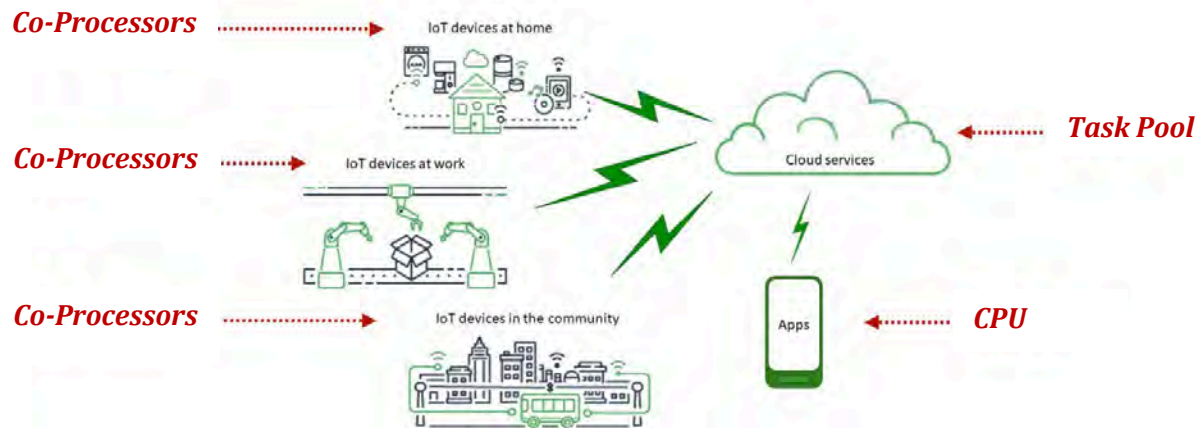


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
5. The apparatus of claim 4 wherein the first agent retrieves the matching task by:	The apparatus is illustrated on Appendix A. “Devices can: ... - Retrieve the next pending job execution by calling the DescribeJobExecution (p. 586) API with jobId \$next.” [Reference 1, page 512] “Option b: Pick from available jobs - When a device first comes online, it should subscribe to the thing's notify topic. - Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. - If the list contains one or more job executions, pick one. - Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails.” [Reference 1, page 514] “thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
searching the task pool for a task that is ready to be processed and has a task type that the first solidarity cell can perform;	“thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between

	publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. 5. Call the UpdateJobExecution (p. 591) MQTT API to update the job status. If the includeJobDocument field is set to true in this command, the device can skip the previous step and retrieve the job document at this point. 6. Optionally, you can add a step timer by setting a value for stepTimeoutInMinutes when you call UpdateJobExecution (p. 591). 7. Perform the actions specified by the job document using the UpdateJobExecution (p. 591) MQTT API to report on the progress of the job.” [Reference 1, page 514] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic. This topic design also makes it possible for the lock to subscribe using a single level MQTT wildcard after its identifier. The single level wildcard command is backward compatible as the IoT application adds new command types.” [Reference 6, page 19] “A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target.” [Reference 1, page 499]
and identifying the matching task.	“thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails.” [Reference 1, page 514] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic. This topic design also makes it possible for the lock to subscribe using a single level MQTT wildcard after its

	identifier. The single level wildcard command is backward compatible as the IoT application adds new command types.” [Reference 6, page 19]
--	--

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

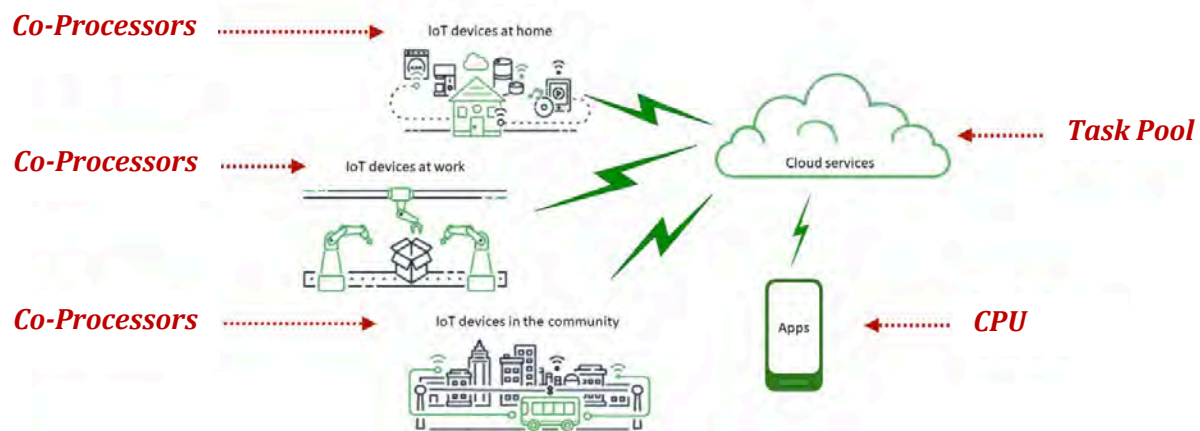


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
6. The apparatus of claim 1 wherein the matching task is a task that is ready to be processed and the function of the matching task can be performed by the first solidarity cell.	The apparatus is illustrated on Appendix A. “Option b: Pick from available jobs - When a device first comes online, it should subscribe to the thing's notify topic. - Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. - If the list contains one or more job executions, pick one. - Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. - Call the UpdateJobExecution (p. 591) MQTT API to update the job status. If the includeJobDocument field is set to true in this command, the device can skip the previous step and retrieve the job document at this point. - Optionally, you can add a step timer by setting a value for stepTimeoutInMinutes when you call UpdateJobExecution (p. 591). - Perform the actions specified by the job document using the UpdateJobExecution (p. 591) MQTT API to report on the progress of the job.” [Reference 1, page 514] “MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic. This topic design also makes it possible for the lock to subscribe using a single level MQTT wildcard after its identifier. The single level wildcard command is backward compatible as the IoT application adds new command types.” [Reference 6, page 19] “A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target.” [Reference 1, page 499]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

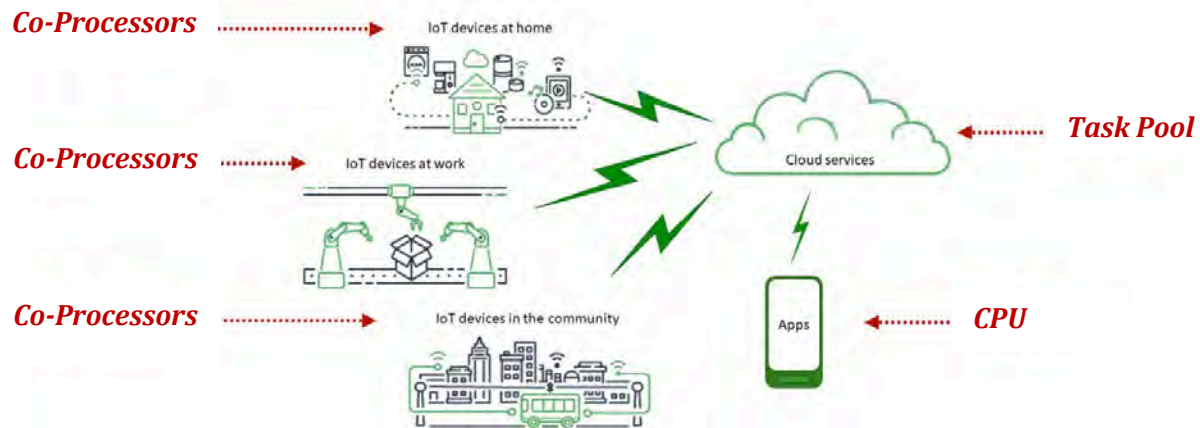


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
7. The apparatus of claim 6 wherein the first agent retrieves the matching task by searching the task pool for a task that is ready to be processed and has a function that the first solidarity cell can perform, and identifying the matching task.	The apparatus is illustrated on Appendix A. “Devices can: ... - Retrieve the next pending job execution by calling the DescribeJobExecution (p. 586) API with jobId \$next.” [Reference 1, page 512] “Option b: Pick from available jobs - When a device first comes online, it should subscribe to the thing's notify topic. - Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. - If the list contains one or more job executions, pick one. - Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. - Call the UpdateJobExecution (p. 591) MQTT API to update the job status. If the includeJobDocument field is set to true in this command, the device can skip the previous step and retrieve the job document at this point. - Optionally, you can add a step timer by setting a value for stepTimeoutInMinutes when you call UpdateJobExecution (p. 591). - Perform the actions specified by the job document using the UpdateJobExecution (p. 591) MQTT API to report on the progress of the job.” [Reference 1, page 514] “thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic. This topic design also makes it possible for the lock to subscribe using a single level MQTT wildcard after its identifier. The single level wildcard command is backward compatible as the IoT application adds new command types.” [Reference 6, page 19] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1]

“A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target.”
[Reference 1, page 499]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.



Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent ‘777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

Patent '777 Claim Elements	AWS
8. The apparatus of claim 1 wherein the descriptor further contains a memory location where processed data is to be stored.	The apparatus is illustrated on Appendix A. “The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT.” [Reference 1, page 499] “When the device is finished executing the job, it calls the UpdateJobExecution MQTT API. If the job was successful, set status to SUCCEEDED and, in the message payload, in statusDetails, add other information about the job as name-value pairs. The in-progress and step timers end when the job execution is complete.” as name-value pairs. The in-progress and step timers end when the job execution is complete.” [Reference 1, page 516]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

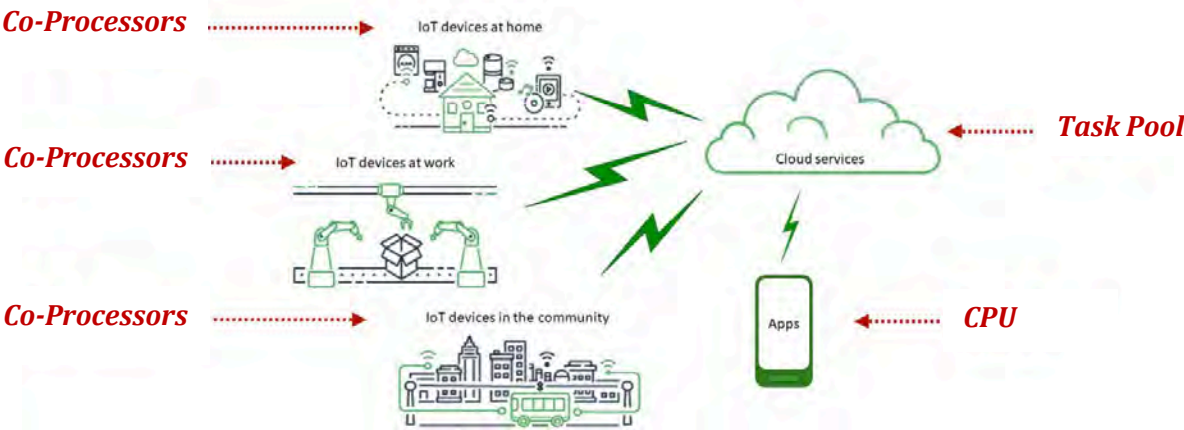


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 18] Amazon Redshift Database Developer Guide

Patent '777 Claim Elements	AWS
9. The apparatus of claim 1 wherein the descriptor is a data structure and the task contains a reference to the memory location of the descriptor.	The apparatus is illustrated on Appendix A. “To create a job, you must first create a job document that is a description of the remote operations to be performed by the devices. Job documents are UTF-8 encoded JSON documents and should contain information that your devices need to perform a job. A job document contains one or more URLs where the device can download an update or some other data.” [Reference 1, page 499] “The JSON data structure is made up of a set of objects or arrays. A JSON object begins and ends with braces, and contains an unordered collection of name-value pairs.” [Reference 18, page 576]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

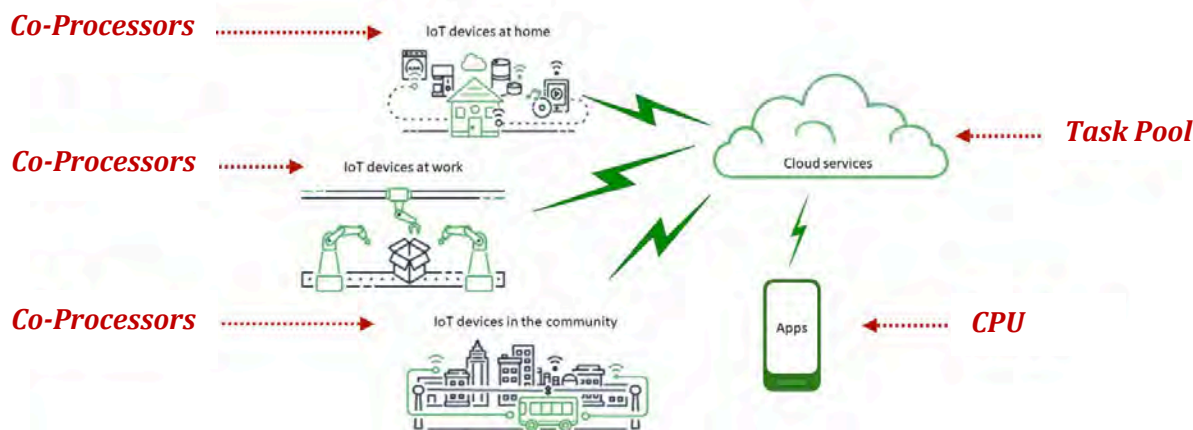


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 7] AWS IoT Greengrass Developer Guide

[Reference 19] Introducing AWS IoT SiteWise Edge

Patent '777 Claim Elements	AWS
10. The apparatus of claim 1 wherein the task pool occupies a region of physical memory.	The apparatus is illustrated on Appendix A. “At a minimum, AWS IoT SiteWise Edge requires an industrial computer running Linux with a x86 64 bit quad-core processor, 16GB RAM, and 256GB in disk space. In preview, we recommend you use dedicated hardware for use with AWS IoT SiteWise Edge. In preview, AWS IoT SiteWise Edge software runs on AWS IoT Greengrass v1 (v1.10.2 recommended) installed on Ubuntu Server 18.04+ LTS OS for x86.” [Reference 19, page 2-3] “AWS IoT Greengrass uses the spooler system component (the GGCloudSpooler Lambda function) to manage the message queue. You can use the following GGCloudSpooler environment variables to configure storage settings. - GG_CONFIG_STORAGE_TYPE. The location of the message queue. The following are valid values: 0 FileSystem. Store unprocessed messages in the local storage cache on the disk of the physical core device. When the core restarts, queued messages are retained for processing. Messages are removed after they are processed.” [Reference 7, page 79]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

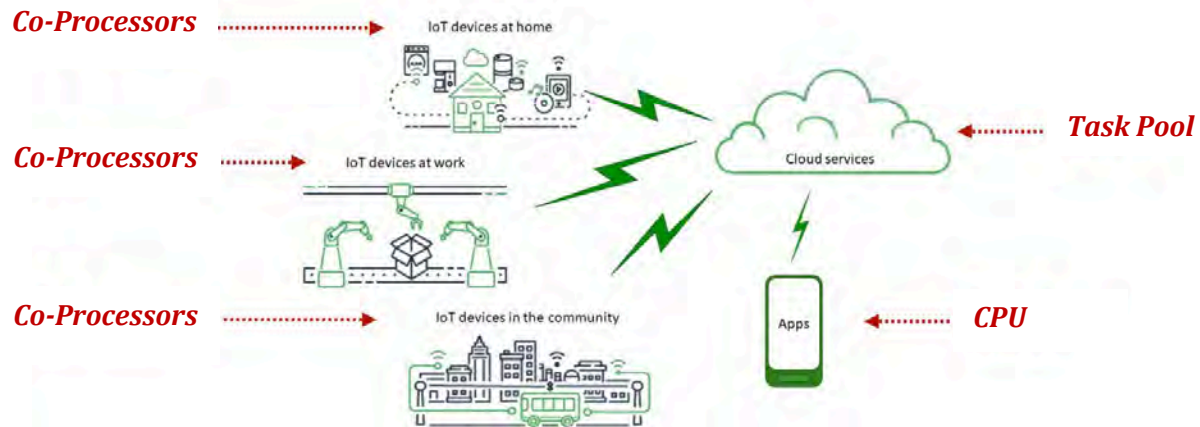


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 7] AWS IoT Greengrass Developer Guide

[Reference 19] Introducing AWS IoT SiteWise Edge

Patent '777 Claim Elements	AWS
11. The apparatus of claim 10 wherein the task pool is disposed in a hardware block dedicated to the task pool.	The apparatus is illustrated on Appendix A. “At a minimum, AWS IoT SiteWise Edge requires an industrial computer running Linux with a x86 64 bit quad-core processor, 16GB RAM, and 256GB in disk space. In preview, we recommend you use dedicated hardware for use with AWS IoT SiteWise Edge. In preview, AWS IoT SiteWise Edge software runs on AWS IoT Greengrass v1 (v1.10.2 recommended) installed on Ubuntu Server 18.04+ LTS OS for x86.” [Reference 19, page 2-3] “AWS IoT Greengrass uses the spooler system component (the GGCloudSpooler Lambda function) to manage the message queue. You can use the following GGCloudSpooler environment variables to configure storage settings. - GG_CONFIG_STORAGE_TYPE. The location of the message queue. The following are valid values: 0 FileSystem. Store unprocessed messages in the local storage cache on the disk of the physical core device. When the core restarts, queued messages are retained for processing. Messages are removed after they are processed.” [Reference 7, page 79]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

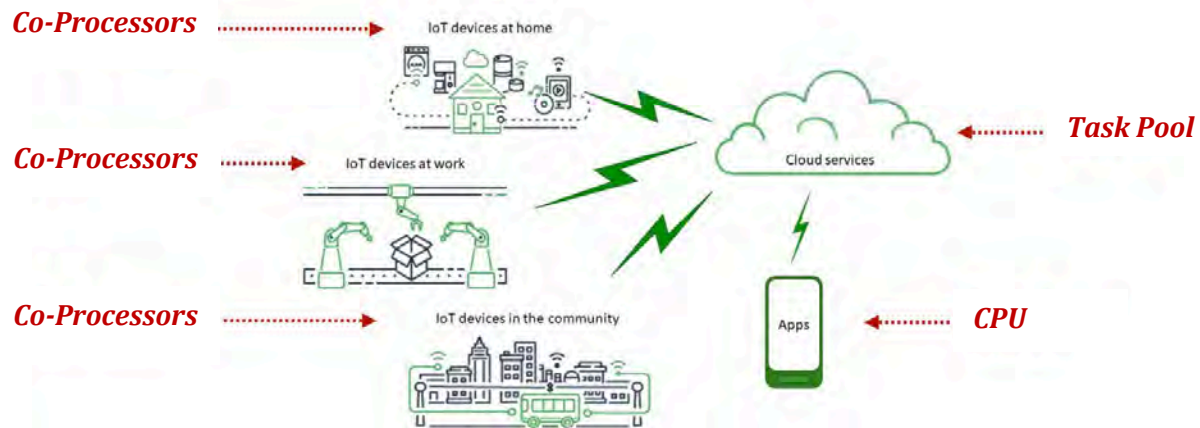


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 5] AWS IoT Core Features

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
12. The apparatus of claim 1 further comprising a second solidarity cell comprising a second agent that proactively retrieves matching tasks from the task pool for the second solidarity cell to process, wherein the matching task for each solidarity cell is a task in the task pool that is ready to be processed and can be performed by the solidarity cell.	The apparatus is illustrated on Appendix A. “Using the MQTT protocol Communication between the AWS IoT Jobs service and your devices can occur over the MQTT protocol. Devices subscribe to MQTT topics to be notified of new jobs and to receive responses from the AWS IoT Jobs service. Devices publish on MQTT topics to query or update the state of a job execution. Each device has its own general MQTT topic.” [Reference 1, page 511] “Fleet provisioning for AWS IoT Core makes it easy to onboard any number of manufactured devices to the cloud at scale. It provides an end-to-end managed device onboarding experience that sets up devices with unique digital identities and performs the device-side and cloud-side configuration needed for each device to connect and operate with AWS IoT automatically upon its first connection to AWS IoT Core” [Reference 5, page 1] “Devices can: ... 1. Retrieve the next pending job execution by calling the DescribeJobExecution (p. 586) API with jobId \$next.” [Reference 1, page 512] “Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails.” 5. Call the UpdateJobExecution (p. 591) MQTT API to update the job status. If the includeJobDocument field is set to true in this command, the device can skip the previous step and retrieve the job document at this point. 6. Optionally, you can add a step timer by setting a value for stepTimeoutInMinutes when you call UpdateJobExecution (p. 591). 7. Perform the actions specified by the job document using the UpdateJobExecution (p. 591) MQTT API to report on the progress of the job.” [Reference 1, page 514]

	“MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target.” [Reference 1, page 499]
--	--

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

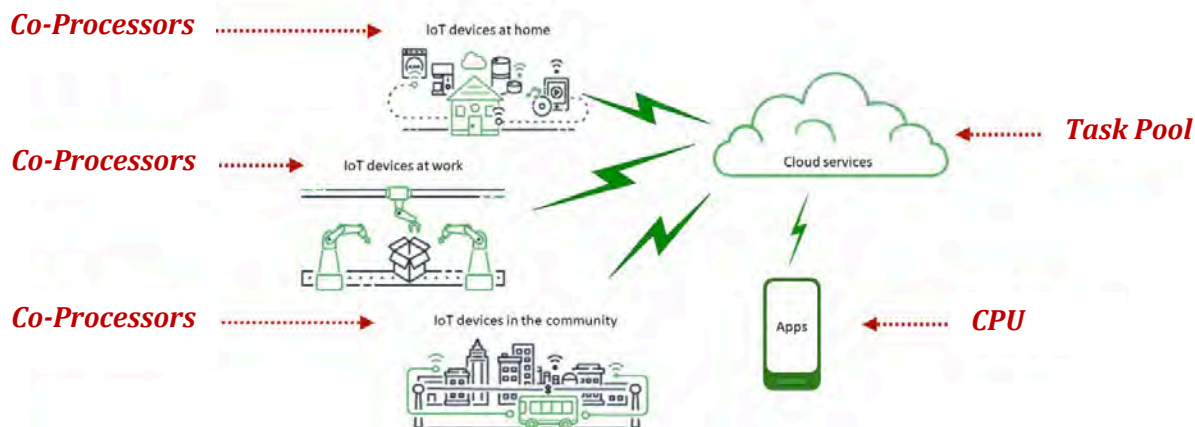


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent ‘777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

[Reference 1] AWS IoT Core Developer Guide

[Reference 6] Designing MQTT Topics for AWS IoT Core

Patent '777 Claim Elements	AWS
13. The apparatus of claim 12 wherein each solidarity cell sends its agent to the task pool when the solidarity cell does not have a matching task to process.	The apparatus is illustrated on Appendix A. “Option a: Get the next job 1. When a device first comes online, it should subscribe to the device's notify-next topic. 2. Call the <u>DescribeJobExecution</u> MQTT API with jobId \$next to get the next job, its job document, and other details, including any state saved in statusDetails. If the job document has a code file signature, you must verify the signature before proceeding with processing the job request. ... If the device remains online, it continues to receive a notifications of the next pending job execution, including its job execution data, when it completes a job or a new pending job execution is added. When this occurs, the device continues as described in step 2.” [Reference 1, page 514] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “Using the MQTT protocol Communication between the AWS IoT Jobs service and your devices can occur over the MQTT protocol. Devices subscribe to MQTT topics to be notified of new jobs and to receive responses from the AWS IoT Jobs service. Devices publish on MQTT topics to query or update the state of a job execution. Each device has its own general MQTT topic.” [Reference 1, page 511]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

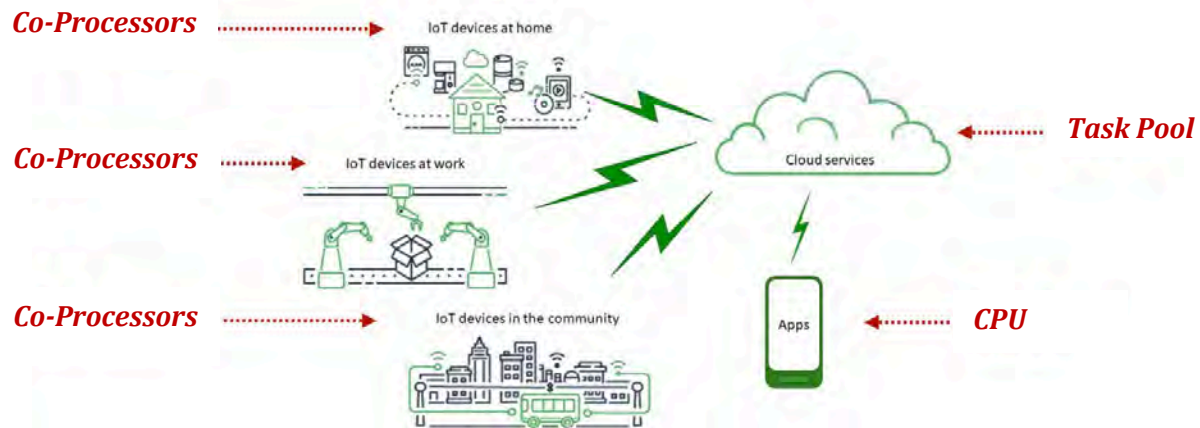


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777

AWS Claim Chart

U.S. Patent 9,146,777

Parallel processing with solidarity cells by proactively retrieving from a task pool a matching task for the solidarity cell to process

References

- [Reference 1] AWS IoT Core Developer Guide
 [Reference 5] AWS IoT Core Features
 [Reference 6] Designing MQTT Topics for AWS IoT Core
 [Reference 10] IP Version 4 (IPv4) Datagram General Format
 [Reference 11] IP Version 6 (IPv6) Datagram Main Header Format
 [Reference 12] AWS Management Console Overview
 [Reference 13] Getting Started with AWS IoT Core

Patent '777 Claim Elements	AWS
14. An apparatus for parallel processing of a large computing requirement, the apparatus comprising:	The apparatus is illustrated on Appendix A.
a central processing unit ("CPU");	"You can use the AWS IoT console, the Jobs HTTPS API, the AWS Command Line Interface, or the AWS SDKs to create and manage jobs." [Reference 1, page 501] "The AWS Management Console brings the unmatched breadth and depth of AWS right to your computer or mobile phone with a secure, easy-to-access, web-based portal" [Reference 12, page 1]
a task pool in electronic communication with the CPU;	"The AWS Management Console provides a web-based interface for accessing and managing all your AWS IoT resources. Programmatic access to AWS IoT Core is enabled by the AWS CLI and AWS SDK." [Reference 13, page 1] "Job management and control data types The following data types are used by management and control applications to communicate with the AWS IoT Jobs service." [Reference 1, page 570]
first solidarity cell in electronic communication with the task pool, the first solidarity cell comprising a first agent configured to proactively retrieve, from the task pool, a matching task for the solidarity cell to process;	"Using the MQTT protocol Communication between the AWS IoT Jobs service and your devices can occur over the MQTT protocol. Devices subscribe to MQTT topics to be notified of new jobs and to receive responses from the AWS IoT Jobs service. Devices publish on MQTT topics to query or update the state of a job execution. Each device has its own general MQTT topic." [Reference 1, page 511] "Fleet provisioning for AWS IoT Core makes it easy to onboard any number of manufactured devices to the cloud at scale. It provides an end-to-end managed device onboarding experience that sets up devices with unique digital identities and performs the

	device-side and cloud-side configuration needed for each device to connect and operate with AWS IoT automatically upon its first connection to AWS IoT Core” [Reference 5, page 1] “Devices can: ... • Retrieve the next pending job execution by calling the DescribeJobExecution (p. 586) API with jobId \$next.” [Reference 1, page 512] “Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails.” [Reference 1, page 514] “MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
and a second solidarity cell comprising a second agent that proactively retrieves matching tasks from the task pool for the second solidarity cell to process, wherein the matching task for each solidarity cell is a task in the task pool that is ready to be processed and can be performed by the solidarity cell;	“Using the MQTT protocol Communication between the AWS IoT Jobs service and your devices can occur over the MQTT protocol. Devices subscribe to MQTT topics to be notified of new jobs and to receive responses from the AWS IoT Jobs service. Devices publish on MQTT topics to query or update the state of a job execution. Each device has its own general MQTT topic.” [Reference 1, page 511] “Fleet provisioning for AWS IoT Core makes it easy to onboard any number of manufactured devices to the cloud at scale. It provides an end-to-end managed device onboarding experience that sets up devices with unique digital identities and performs the device-side and cloud-side configuration needed for each device to connect and operate with AWS IoT automatically upon its first connection to AWS IoT Core” [Reference 5, page 1] “Devices can: ... 1. Retrieve the next pending job execution by calling the DescribeJobExecution (p. 586) API with jobId \$next.” [Reference 1, page 512] “Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails.

	5. Call the UpdateJobExecution (p. 591) MQTT API to update the job status. If the includeJobDocument field is set to true in this command, the device can skip the previous step and retrieve the job document at this point. 6. Optionally, you can add a step timer by setting a value for stepTimeoutInMinutes when you call UpdateJobExecution (p. 591). 7. <u>Perform</u> the actions specified by the job document using the UpdateJobExecution (p. 591) MQTT API to report on the progress of the job.” [Reference 1, page 514] “MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target.” [Reference 1, page 499]
wherein each solidarity cell sends its agent to the task pool when the solidarity cell does not have a matching task to process;	“Option a: Get the next job 1. When a device first comes online, it should subscribe to the device's notify-next topic. 2. Call the DescribeJobExecution MQTT API with jobId \$next to get the next job, its job document, and other details, including any state saved in statusDetails. If the job document has a code file signature, you must verify the signature before proceeding with processing the job request. ... If the device remains online, it continues to receive a notifications of the next pending job execution, including its job execution data, when it completes a job or a new pending job execution is added. When this occurs, the device continues as described in step 2.” [Reference 1, page 514] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “Using the MQTT protocol Communication between the AWS IoT Jobs service and your devices can occur over the MQTT protocol. Devices subscribe to MQTT topics to be notified of new jobs and to receive responses from the AWS IoT Jobs service. Devices publish on MQTT topics to query or update the state of a job execution. Each device has its own general MQTT topic.” [Reference 1, page 511]

and wherein each agent comprises a source address, a destination address, and a payload, and wherein each agent retrieves a matching task by:

“When creating a security profile, you can specify your IoT device's expected behavior by configuring behaviors and thresholds for metrics generated by IoT devices. The following are device-side metrics, which are metrics from agents that you install on your devices ...

... Destination IPs (aws:destination-ip-addresses)

A set of IP destinations.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) Classless Inter-Domain Routings (CIDR) from which each device must or must not connect to AWS IoT...

... Source IP (aws:source-ip-address)

The IP address from which a device has connected to AWS IoT.

Use this metric to specify a set of allowed (formerly referred to as whitelisted) or denied (formerly referred to as blacklisted) Classless Inter-Domain Routings (CIDR) from which each device must or must not connect to AWS IoT.”

[Reference 1, page 803-816]

“MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload:

- Who is the originator of the command?
- Who is the intended receiver of the command?
- Was the command processed successfully?
- What is the current status of the command?
- If the command was not processed successfully, what is the error?”

[Reference 6, page 13]

“In the payload of this request, the IoT application includes a field that denotes where the device (device-1) should send its response and a session identifier for tracking”

[Reference 6, page 15]

“AWS IoT Core supports devices and clients that use the MQTT and the MQTT over WebSocket Secure (WSS) protocols to publish and subscribe to messages, and devices and clients that use the HTTPS protocol to publish messages. All protocols support IPv4 and IPv6.”

[Reference 1, page 72]

“Table 56: Internet Protocol Version 4 (IPv4) Datagram Format

Source Address: The 32-bit IP address of the originator of the datagram.

Destination Address: The 32-bit IP address of the intended recipient of the datagram”

[Reference 10, page 2]

“Table 68: IPv6 Main Header Format

Source Address: The 128-bit IP address of the originator of the datagram. As with IPv4, this is always the device that originally sent the datagram.

Destination Address: The 128-bit IP address of the intended recipient of the datagram; unicast, anycast or multicast. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.”

[Reference 11, page 2]

“Devices can:

...

- Retrieve the next pending job execution by calling the DescribeJobExecution (p. 586) API with jobId \$next.”

[Reference 1, page 512]

	“Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails.” [Reference 1, page 514] “thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19]
being dispatched by its solidarity cell to the task pool, during which the source address is its solidarity cell's address, the destination address is the task pool's address, and the payload comprises a list of functions the agent's solidarity cell is configured to perform;	“Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. 5. Call the UpdateJobExecution (p. 591) MQTT API to update the job status. If the includeJobDocument field is set to true in this command, the device can skip the previous step and retrieve the job document at this point. 6. Optionally, you can add a step timer by setting a value for stepTimeoutInMinutes when you call UpdateJobExecution (p. 591). 7. Perform the actions specified by the job document using the UpdateJobExecution (p. 591) MQTT API to report on the progress of the job.” [Reference 1, page 514] “MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload: 1. Who is the originator of the command? 2. Who is the intended receiver of the command? 3. Was the command processed successfully? 4. What is the current status of the command? 5. If the command was not processed successfully, what is the error?” [Reference 6, page 13] “AWS IoT Jobs is a service that allows you to define a set of remote operations that are sent to and executed on one or more things connected to AWS IoT. For command use cases, Jobs allows applications to run tasks that require executing multiple steps. An

	AWS IoT Job contains instructions that the thing must run to complete its transaction. AWS IoT Jobs are the recommended feature for fleet-wide operational tasks, such as software updates, that are only executed by trusted administrators of the entire IoT application.” [Reference 6, page 12] “The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its specific command topic.” [Reference 6, page 19] “A job execution is an instance of a job on a target device. The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT. An execution number is a unique identifier of a job execution on a specific target.” [Reference 1, page 499]
searching the task pool for a task that is ready to be processed and has a function that the agent's solidarity cell can perform;	“thingGroupId Optional. A filter that lets you search for jobs that have the specified thing group ID as a target.” [Reference 1, page 561] “AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a matching mechanism between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.” [Reference 6, page 1] “Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. 5. Call the UpdateJobExecution (p. 591) MQTT API” [Reference 1, page 514] “The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT.” [Reference 1, page 499]
and returning to its solidarity cell, during which the source address is the task pool's address, the destination address is the agent's solidarity cell's address, and the payload comprises a descriptor of the matching task.	“Option b: Pick from available jobs 1. When a device first comes online, it should subscribe to the thing's notify topic. 2. Call the GetPendingJobExecutions (p. 578) MQTT API to get a list of pending job executions. 3. If the list contains one or more job executions, pick one. 4. Call the DescribeJobExecution (p. 586) MQTT API to get the job document and other details, including any state saved in statusDetails. 5. Call the UpdateJobExecution (p. 591) MQTT API” [Reference 1, page 514] “The target starts an execution of a job by downloading the job document. It then performs the operations specified in the document, and reports its progress to AWS IoT.” [Reference 1, page 499]

“MQTT command topics should be designed in a way that can answer operational questions based on the MQTT topic and payload:

- Who is the **originator of the command**?
- Who is the **intended receiver of the command**?
- Was the command processed successfully?
- What is the current status of the command?
- If the command was not processed successfully, what is the error?”

[Reference 6, page 13]

“To create a job, you must first create a **job document** that is a description of the remote operations to be performed by the devices.

Job documents are UTF-8 encoded JSON documents and should contain **information that your devices need to perform a job**. A job document contains one or more URLs where the device can download an update or some other data.”

[Reference 1, page 499]

“The smart lock receives the command message from the MQTT topic. By leveraging a consistent naming schema for commands in this application, the smart lock can ensure that it only receives commands on its **specific** command topic.”

[Reference 6, page 19]

“AWS IoT Core supports MQTT, a widely adopted lightweight messaging protocol designed for constrained devices. MQTT participants receive information organized through MQTT topics. An MQTT topic acts as a **matching mechanism** between publishers and subscribers. Conceptually, an MQTT topic behaves like an ephemeral notification channel.”

[Reference 6, page 1]

“thingGroupId

Optional. A filter that lets you search for jobs that have the **specified thing group ID** as a target.”

[Reference 1, page 561]

Appendix A

The IoT universe

In general, the Internet of Things (IoT) consists of the key components shown in this diagram.

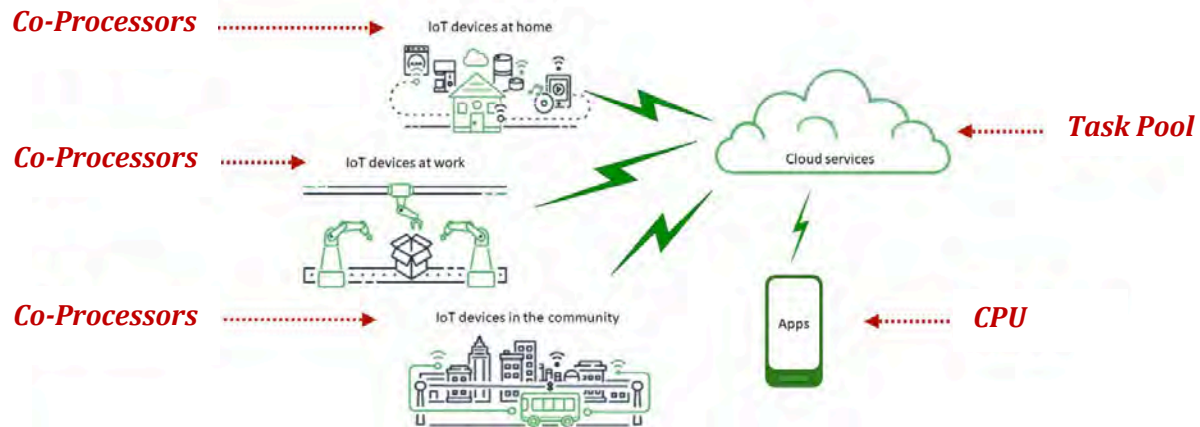


Figure 1. Key components of The IoT Universe

Source: Reference 1, page 4

Note: Added the terms *Co-processors*, *Task Pool*, and *CPU*, to map to Patent '777