SCTE® CABLE-TEC EXPO® FAST FORWARD 2021

ATLANTA, GA
OCTOBER 11-14

SCTE
a subsidiary of CableLabs®

UNLEASH THE POWER OF LIMITLESS CONNECTIVITY

2021 Fall
Technical Forum
SCTE® • NCTA • CABLELABS®

SCTE
a subsidiary of CableLabs®

Internet of Things, Home Networking, Smart Cities, and Emerging Services

Message Queuing Telemetry Transport (MQTT) For IoT Devices: Less is More

Sweety Bertilla

Sr. Android Engineer
Comcast Cable

2021 Fall Technical Forum
SCTE® • NCTA • CABLELABS®

SCTE CABLE-TEC EXPO® FAST FORWARD 2021
VIRTUAL EXPERIENCE
OCTOBER 11-14

Introduction

- Well-designed, lightweight messaging protocol used for communication between IoT devices.
- Low bandwidth, low latency alternative for IoT device transmissions
- Uses publish/subscribe operations to exchange data between client and server
- Unlike HTTP, this method saves a substantial amount of time previously spent on polling, which makes updates occur more quickly and smoothly.
- Key elements – Connect, Disconnect, Subscribe, Un-subscribe, Publish, and Topic
- Quality of Service(QoS) feature supported by MQTT helps application client to opt the level of service based on network reliability.
- Communicates through MQTT message broker

History

- Created in 1999 by Andy Standford-Clark (IBM) and Alen Nipper (Arcom, now Eurotech) as part of the IBM MQ series of products .
- MQTT is not a message queue, but can queue messages for clients
- As a lightweight messaging protocol, it has been widely adopted for MTM (machine to machine) communications for industrial, messenger, and IoT applications
- Ability to keep bandwidth requirements to a minimum, deal with high latency, unreliable networks, small footprint devices and low power consumption, has made it an excellent choice for communication.
- Unlike other protocols like HTTP, MQTT continues to edge out other options due to its nature as a lightweight, asynchronous, bi-directional, secure, efficient, reliable, publish/subscribe, and data agnostic messaging protocol.

Advantages of MQTT

Asynchronous bi-directional communication

- Provides significant performance and responsiveness in client-server communications
- Inherently *non-blocking,* which means that an application does not need to wait for a response in order to proceed.
- Messages can also be pushed directly to the client without an explicit request.
- Downstream microservices can choose to directly send updates to a client application

MQTT Features

Pub/Sub

- Publisher / subscriber pattern which provides a framework for exchanging messages between publishers (producers) and subscribers (consumers).
- A publisher can send a message and it will automatically be routed – via the MQTT message broker – to any client subscribed to that topic.

MQTT Features

Topic structure

- MQTT uses a topic hierarchy in order to send and receive messages.
- MQTT topics can consist of wildcards and topic separators.
  - + A plus represents a single topic level wildcard.
  - # A hash is a multi-level wildcard that can only be at the end of a topic subscription.
- Example:
  - platform / customerId4 / house1 / #
  - platform / customerId4 / + / lights / #

MQTT Features

Quality of Service (QoS)

- MQTT Client gets an option to select the level of service that works for their application based on network reliability.
- MQTT manages the re-transmission of messages and guarantees delivery, even when the underlying transport is not reliable
- QoS 0 is also known as Fire and Forget service. MQTT client sends the message to MQTT broker and doesn't wait for an acknowledgement.
- QoS 1 makes sure that the MQTT message is delivered at least once to the end client. The publisher send the message to MQTT broker and waits for an acknowledgement.
- QoS 2 is also known as the safest and slowest message service. There is a four-way handshake that happens between publisher and MQTT broker to make sure the end client receives the message only once.

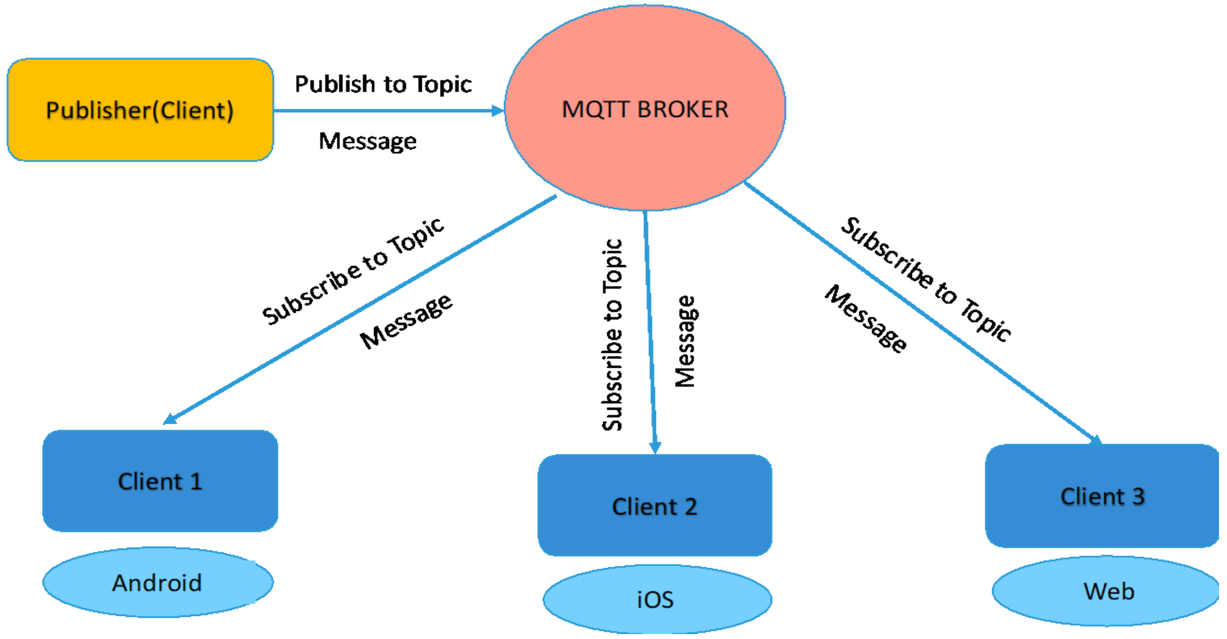
MQTT Features

Last Will and Testament

- Last Will and Testament (LWT) is a feature of MQTT that can notify other clients about an ungracefully disconnected client.
- LWT contains several parameters:
    - Last Will Topic (topic that the LWT message will be published to)
    - Last Will QoS (QoS of the LWT message)
    - Last Will Message (LWT message itself)
    - Last Will Retained (boolean) (whether the last lost connection information will be retained)

MQTT for application client

- MQTT uses publish/subscribe operations to exchange data between clients. The MQTT message broker acts as medium for communication between the clients.
- Key components include the MQTT broker, client(s) and topic(s).



MQTT for application client

MQTT Connect

- Client can establish secure persistent connection with MQTT broker
- AWS IoT core is the MQTT broker used in Xfinity app.
- Different types of connection can be established using AWS IoT core
  - Connect with keystore and port number
  - Connect with AWS credentials provider
  - Connect with proxy host and proxy port.
  - Connect using custom authorizer. The IoT client of Xfinity Application uses custom authorizer to connect with MQTT broker. This custom authorizer is used to validate the customer against an internal identity provider.

MQTT for application client

Publish and Subscribe to Topic

- Client connected to MQTT broker can publish or subscribe to topics
- Receive async messages when subscribed to a topic
- Client can set different levels of Quality of Service either QoS0, QoS1 or QoS2(AWS IoT broker doesn't support QoS2)
- To stop receiving messages client can unsubscribe to topics
- The messages published can be a simple string or json string

MQTT for application client

MQTT and AWS shadow state

- AWS Shadow State service, provided by AWS IoT Core, is a persistent cache that can be used to store IoT device state Receive async messages when subscribed to a topic
- The two types of state in shadow state messages are *desired state* and *reported state*
- Desired state lets the client send a request through MQTT broker
- The reported state is returned via the device's MQTT topic to which the client is subscribed

MQTT for application client

Message payload

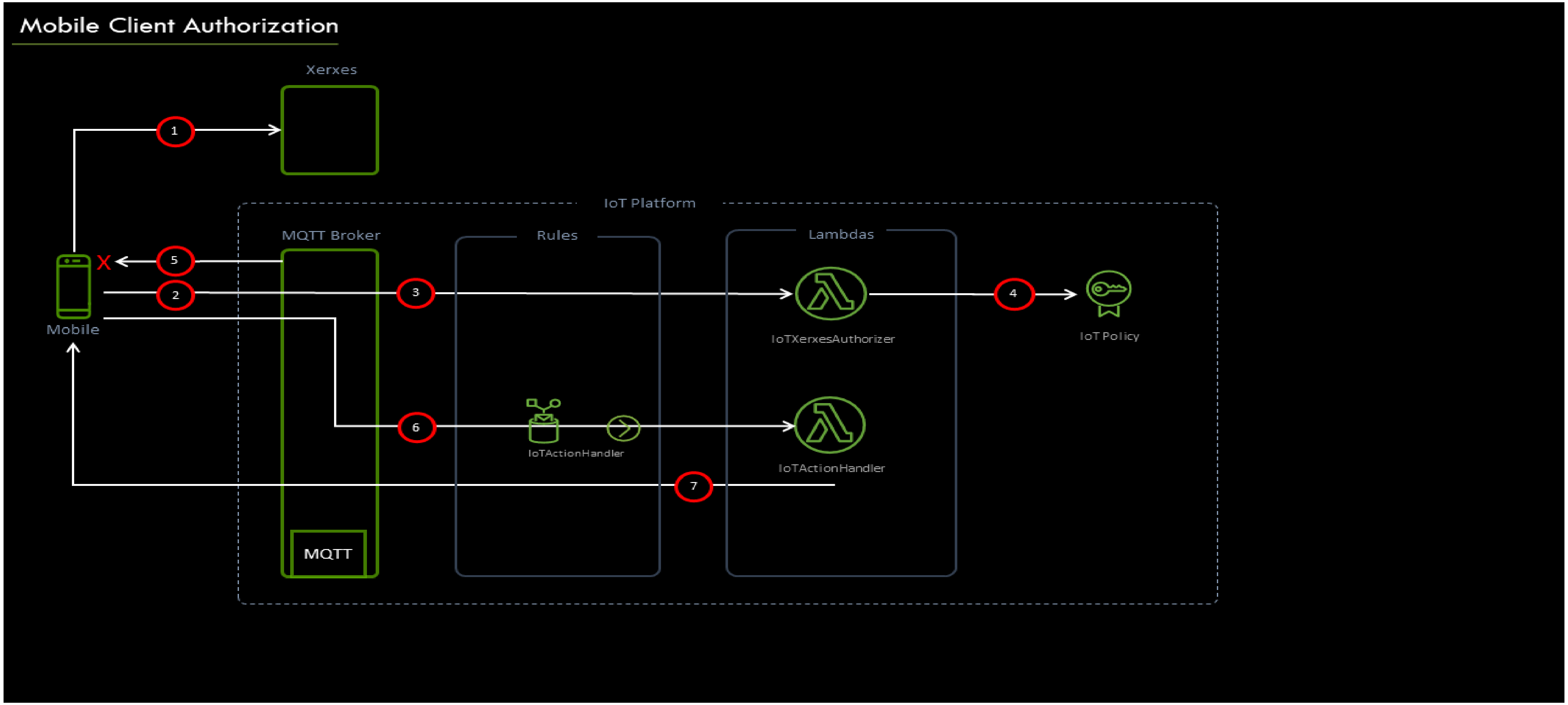
- Since MQTT is a lightweight protocol, the message payloads cannot extend in size
- AWS IoT Core supports a maximum payload of 128kb
- MQTT messages also have no guaranteed ordering. Due to these limitations, we included *sequence number* and *total messages expected* to our MQTT message payloads to ensure all messages are received and ordered properly.
- Shadow State Reported state message payloads contain a property named *version*, which gets incremented with each change in shadow state

Security

Mobile Client

- Authentication and authorization via a Java Web Token (JWT) based Identity Token
- Connect – allow the MQTT connection with a unique MQTT Client ID (CID) generated from the principle information within the JWT
- Publish – allows the MQTT client to publish messages to MQTT topics which are name spaced based on the PID, AID and JTI
- Subscribe/receive – allows the MQTT client to subscribe/receive messages to/from MQTT topic(s) which are name spaced based on the PID, AID and JTI.

Message Queuing Telemetry Transport (MQTT)

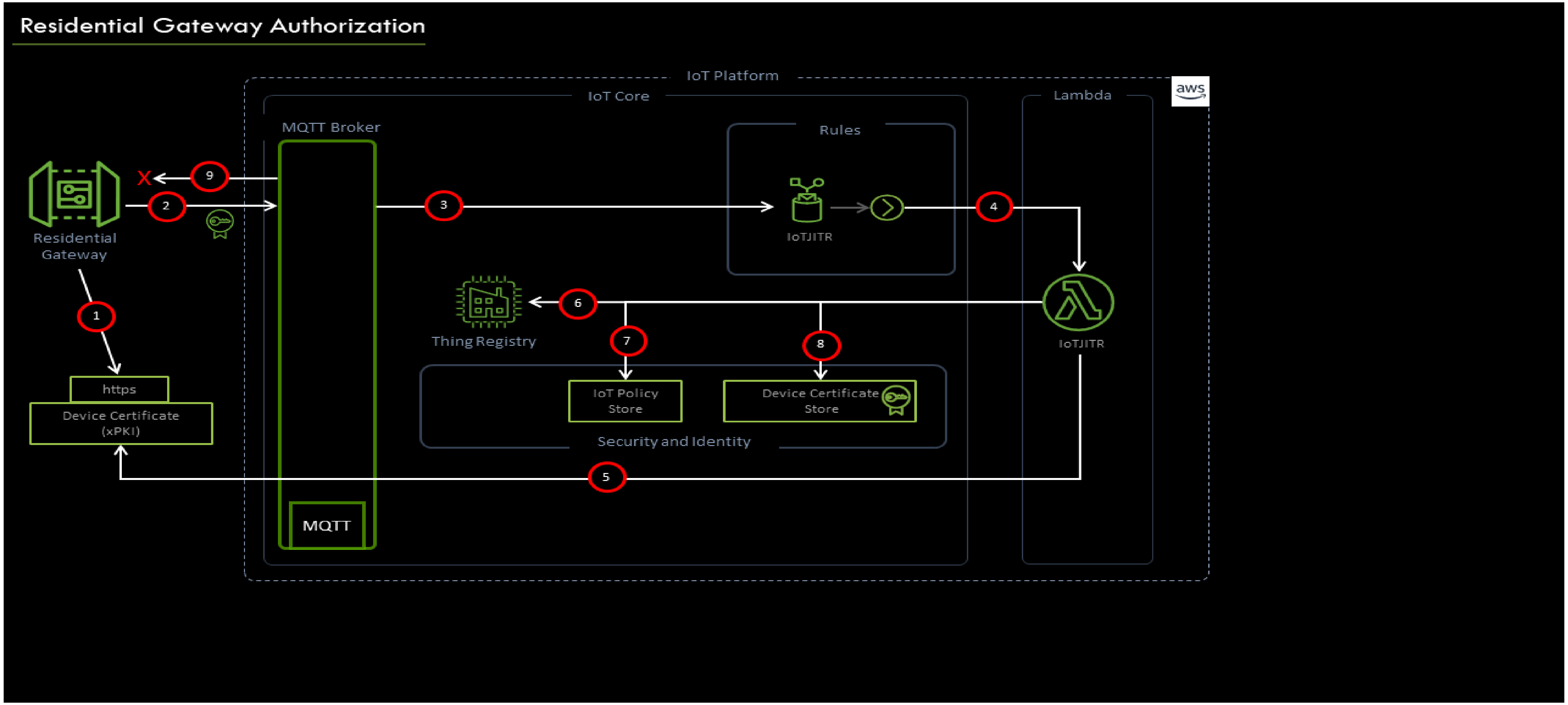


Security

Residential Gateway

- The Xfinity Residential Gateway enables advanced IoT Bridge capabilities for Zigbee, Thread, and WIFI based devices.
- To provide a secure MQTT connection to the IoT Platform, the Residential Gateway performs a CSR (Certificate Signing Request) and is issued a device certificate by Xfinity Public Key Infrastructure (xPKI).
- Within the Certificate there are critical principle attributes that provide for a fine-grained security model. These attributes are:
  - Partner Identifier - syndicated partner identifier for the Multi System Operator (MSO)
  - Account Identifier (AID) - identifies the subscriber's opaque account id
  - Installation Identifier – the device identifier as a UUID (Universal Unique Identifier)
  - Mac Address – Device CMAC address

Message Queuing Telemetry Transport (MQTT)



MQTT for client-platform communication

Evaluation Criteria

- Asynchronous Bi-directional Messaging Support
- Data Model Support - support for exchanging messages based on a JSON Schema data model
- Platform Support - Android, IOS, and Web Clients must be supported
- Secure - provide a secure communication channel between the mobile/web application clients, gateways, and the IoT Platform
- Efficient - provide a lightweight implementation
- Reliable - offers a way to mitigate errors that may arise from unreliable mobile/home networks
- Performant - minimizes latency due to network protocol negotiation and message broker and router traversal

MQTT vs other messaging protocols - HTTP

- *Pros*
  - Supports primary request/response
  - reasonably small - HPACK for header compression required
  - IETF standard - Preferred by some vendors
  - Gateway supports millions of connected clients
  - Used in mobile applications and most web applications not requiring pub/sub
- *Cons*
  - No guaranteed deliver (retry required)
  - No last will & testament
  - Slightly larger message size due to headers
  - Does not have a wide adoption
  - Short polling is not real-time (request timer driven)
  - Long polling is immediate but is more complex

MQTT vs other messaging protocols - Websockets

- *Pros*
  - Works over port 443
  - Supported by numerous mobile clients and web browsers (modern browsers implementation less of a concern)
  - Supported by many web servers such as NGINX and Apache
- *Cons*
  - Scaling of web servers to meet needs of mobile client persistent connections
  - Client reconnection implementation on top of WebSockets required
  - bit too raw; while it supports two-way client/server communication
  - Support for request/response and pub/sub message exchange patterns
  - Requires additional work

MQTT vs other messaging protocols - AMQP

- *Pros*
  - Richest set of message scenarios (patterns)
  - Asynchronous
  - Supports queuing
  - Mobile client support; web browser support via web sockets
- *Cons*
  - Uses port 5672 not 443 (potential router and firewall issues)
  - Requires RabbitMQ on EC2 instances
  - Not a managed AWS service
  - Larger protocol
  - Used in IoT space by very few vendors

MQTT vs other messaging protocols - XMPP-IoT

- *Pros*
  - Support for message read, write, and consume
  - Extended messaging and presence protocols
  - Used for two way chat and push notifications
- *Cons*
  - Uses port 5222 not 443 (potential router and firewall issues)
  - Requires an XMPP/Jabber server on EC2 instances
  - Not a managed AWS service
  - It's XML based

MQTT vs other messaging protocols - MQTT

- *Pros*
  - JWT and X.509 certificate-based authorization support
  - Support for port 443
  - Supports asynchronous message patterns (i.e. pub/sub)
  - Assured delivery (3 QoS levels) and retained messages which provide flexible options for Client/Server
  - Last will & testament - notifies other clients of an ungraceful disconnected of gateway-based devices
- *Cons*
  - OASIS standard - not preferred by some vendors
  - Fixed headers and options if extensibility is required

MQTT vs other messaging protocols - MQTT

- *Pros[Continued]*
  - Multiple subscriptions 'multiplexed' over one connection
  - Smaller size on the wire - minimum compressed header
  - Desired for a resource-constrained device, low bandwidth, and high latency networks
  - Brokers support millions of connected devices
  - Easy to route messages to topic subscriber(s)
  - Lightweight implementation for embedded clients
  - Power-efficient due to no-polling, shorter messages
  - Less resources consumed compared to long polling situation on Server.
  - Support for many programming languages
  - MQTT is mature and stable.

SCTE® CABLE-TEC EXPO® FAST FORWARD 2021

ATLANTA, GA
OCTOBER 11-14

SCTE®
a subsidiary of CableLabs®

Thank You!

Sweety Bertilla

Sr. Android Engineer
Comcast Cable
Sweetybertilla_francisxavier@cable.comcast.com

Robert Farnum

Principal II Engineer
Comcast Cable
robert_farnum@comcast.com

Kristopher Linquist

Principal II Engineer
Comcast Cable
kris_linquist@comcast.com