This section explains how devices can use the MQTT bridge to communicate with Cloud IoT Core. For general information about HTTP and MQTT, see Protocols (/iot/docs/concepts/protocols).

Be sure to refer to the API documentation (/iot/docs/reference/rest) for full details about each method described in this section. See also the sample MQTT clients (/iot/docs/samples/mqtt-samples).

To publish over the MQTT bridge:

1. Install an MQTT client on your device.
2. Download an MQTT server certificate (#downloading_mqtt_server_certificates) onto your device.
3. Configure the MQTT client (#configuring_mqtt_clients) to authenticate the device to Cloud IoT Core.
4. Initiate a TLS handshake over `mqtt.googleapis.com` or a long-term support domain (#using_a_long-term_mqtt_domain).
5. Publish telemetry events (#publishing_telemetry_events) or set the device state (#setting_device_state).

Cloud IoT Core supports the MQTT protocol by running a managed broker that listens to the port `mqtt.googleapis.com:8883`. Port 8883 is the standard TCP port reserved with IANA (http://iana.org/) for secure MQTT connections. Connections to this port must use TLS transport (/iot/docs/requirements), which is supported by open source clients like Eclipse Paho (https://www.eclipse.org/paho/).

If port 8883 is blocked by your firewall, you can also use port 443: `mqtt.googleapis.com:443`.

The MQTT standard is defined for implementing a full publish/subscribe broker. However, the managed MQTT bridge ud IoT Core does not support all publish/subscribe operations, such as creating arbitrary topics that devices can use nessages between them. (Filtering can be accomplished with downstream processes running on Cloud Pub/Sub.) Cl re uses a predefined set of topics and specific topic formats.

The MQTT specification describes three Quality of Service (QoS) levels:

- QoS 0, delivered at most once
- QoS 1, delivered at least once
- QoS 2, delivered exactly once

Cloud IoT Core does not support QoS 2. Publishing QoS 2 messages closes the connection. Subscribing to a predefined topic with QoS 2 downgrades the QoS level to QoS 1.

QoS 0 and 1 function as follows in Cloud IoT Core:

- A `PUBLISH` message with QoS 1 will be acknowledged by the `PUBACK` message after it has been successfully sent to Cloud Pub/Sub.
- `PUBLISH` messages with QoS 0 do not require `PUBACK` responses, and may be dropped if there is any jitter along the message delivery path (for example, if Cloud Pub/Sub is temporarily unavailable).
- The Cloud IoT Core MQTT bridge maintains a small buffer of undelivered messages in order to retry them. If the buffer becomes full, the message with QoS 1 may be dropped and a `PUBACK` message will not be sent to the client. The client is expected to resend the message.

For device configurations (/iot/docs/how-tos/config/configuring-devices), QoS levels are as follows:

- When QoS is 0, a given configuration version will be published to the device only once. If the device does not receive the configuration, it must resubscribe. A QoS of 0 is thus useful when a configuration is frequently updated (on the order of seconds or minutes) and it's not necessary for the device to receive every update.
- When QoS is 1, the latest configuration update will be retried until the device acknowledges it with a PUBACK. If a newer configuration is pushed before the older one is acknowledged, the older one will not be redelivered; instead, the new one will be delivered (and redelivered). This level is the safest mode for device configurations: it guarantees that the device will eventually get the latest configuration.

To use TLS transport, devices must verify Cloud IoT Core server certificates to ensure they're communicating with Cloud IoT Core rather than an impersonator. The following certificate packages support verification:

- The complete (https://pki.goog/roots.pem) Google root CA certification package (128 KB) for `mqtt.googleapis.com`.
    - This package establishes the chain of trust to communicate with Google products and services, including Cloud IoT Core.
    - Devices with the complete root CA certification package communicate directly with the MQTT server.
    - This package is regularly updated.
- Google's minimal root CA set (<1 KB) for `mqtt.2030.ltsapis.goog`. The minimal root CA set includes a primary (https://pki.goog/gtsltsr/gtsltsr.crt) and backup (https://pki.goog/gsr4/GSR4.crt) certificate.
    - This set is for devices with memory constraints, like microcontrollers, and establishes the chain of trust to communicate with Cloud IoT Core only.
    - Devices with the minimal root CA set communicate with the Cloud IoT Core via long-term support domains (#using_a_long-term_mqtt_domain).
    - This set is fixed through 2030 (the primary and backup certificates won't change). For added security, Google Trust Services (https://pki.goog/) may switch between the primary and backup certificates at any time without notice.

After downloading Google root CA certificates to your device, you can configure an MQTT client to authenticate the device (#configuring_device_authentication), connect to the MQTT server, and communicate over the MQTT bridge.

MQTT clients authenticate devices by connecting to the MQTT bridge. To configure an MQTT client to authenticate a device:

1. Set the MQTT client ID to the full device path:

2. Associate the MQTT client with MQTT server certificates (#downloading_mqtt_server_certificates).
3. Set the MQTT host name to `mqtt.googleapis.com` or a long-term support domain (#using_a_long-term_mqtt_domain) (if you used the minimal root CA set).

4. Specify a username. The MQTT bridge ignores the username field, but some MQTT client libraries will not send the password field unless the username field is specified. For best results, supply an arbitrary username like `unused` or `ignored`.
5. Set the password. The password field must contain the JWT (/iot/docs/how-tos/credentials/jwts).

The following sample shows how to configure the MQTT client to authenticate a device:

Long-term support (LTS) domains let you use one TLS configuration for an extended period of time. You can set up an MQTT client once, configure the MQTT client to publish messages through an LTS domain, and then communicate over the MQTT bridge continuously during the supported time frame.

The current active LTS domain is `mqtt.2030.ltsapis.goog`. This LTS domain is supported through 2030.

To use the LTS domain:

1. Configure an MQTT client to publish messages through an LTS domain.
    a. Configure the MQTT client (#configuring_mqtt_clients) to authenticate the device to Cloud IoT Core.
    b. When configuring the device, associate the minimal root CA set's primary (https://pki.goog/gtsltsr/gtsltsr.crt) and backup (https://pki.goog/gsr4/GSR4.crt) certificates

with the MQTT client.

2. Initiate a TLS handshake over `mqtt.2030.ltsapis.goog` on port 8883 or 443. Use at least the following TLS features:
    - TLS 1.2 (https://www.ietf.org/rfc/rfc5246.txt)
    - P-256 with SHA-256 as the certificate key and hash algorithm
    - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 using P-256 and uncompressed points for the cipher suite
    - Server Name Indication
    - DNS over TCP or UDP

For more information on securing MQTT traffic, including messages sent to LTS domains, see Device security recommendations (/iot/docs/concepts/device-security#device_security_recommendations).

After the device is configured with an MQTT client and connected to the MQTT bridge, it can publish a telemetry event by issuing a `PUBLISH` message to an MQTT topic in the following format:

The device ID (/iot/docs/concepts/devices#device_identifiers) is the string ID of the device specified in the MQTT client ID (#configuring_mqtt_clients). The device ID is case sensitive.

Messages published to this MQTT topic are forwarded to the corresponding registry's default telemetry topic. The default telemetry topic is the Cloud Pub/Sub topic specified in the `eventNotificationConfigs` (/iot/docs/reference/cloudiot/rest/v1/projects.locations.registries#resource-deviceregistry) `[i].pubsubTopicName` (/iot/docs/reference/cloudiot/rest/v1/projects.locations.registries#eventnotificationconfig) field in the registry resource (/iot/docs/reference/cloudiot/rest/v1/projects.locations.registries#resource-deviceregistry). If no default Pub/Sub topic exists, published telemetry data will be lost. To publish messages to other Cloud Pub/Sub topics, see Publishing telemetry events to separate Pub/Sub topics (#publishing_telemetry_events_to_separate_pubsub_topics).

The forwarded message data field contains a copy of the message published by the device, and the following message attributes (/pubsub/docs/overview#data_model) are added to each message in the Cloud Pub/Sub topic:

Attribute	Description
`deviceId`	The user-defined string identifier for the device, for example, **`thing1`**. The device ID must be unique within the registry.
`deviceNumId`	The server-generated numeric ID of the device. When you create a device (https://cloud.google.com/iot/docs/how-tos/devices), Cloud IoT Core automatically generates the device numeric ID; it's globally unique and not editable.
`deviceRegistryLocation`	The Google Cloud Platform region of the device registry, for example, **`us-central1`**.
`deviceRegistryId`	The user-defined string identifier for the device registry, for example, **`registry1`**.
`projectId`	The string ID of the cloud project that owns the registry and device.
`subFolder`	The subfolder can be used as an event category or classification. For MQTT clients, the subfolder is the subtopic after ***`DEVICE_ID`*****`/events`**, which is copied directly. For example, if the client publishes to the MQTT topic **`/devices/`*****`DEVICE_ID`*****`/events/alerts`**, the subfolder is the string **`alerts`**.

If you try to publish a device telemetry event without specifying a Cloud Pub/Sub topic for the device's registry, the M
ction closes automatically. To verify why the connection closed, get the device details
docs/reference/cloudiot/rest/v1/projects.locations.registries.devices) and check the **`"lastErrorStatus"`** field in th
se. This applies only to telemetry events, not state data.

The following sample shows how to send `PUBLISH` messages through the MQTT connection:

Devices can publish data to additional Cloud Pub/Sub topics. By default, MQTT messages published to `/devices/DEVICE_ID/events` are forwarded to the corresponding registry's default telemetry topic. You can specify a subfolder in the MQTT topic to forward data to additional Cloud Pub/Sub topics. The subfolder The subfolder is the subtopic after `/devices/DEVICE_ID/events`.

Messages published to a subfolder are forwarded to the Cloud Pub/Sub topic with the same name. The corresponding registry must be configured with the Cloud Pub/Sub topic (/iot/docs/how-tos/devices#creating_a_device_registry_with_multiple_pubsub_topics); otherwise, messages are forwarded to the default (/iot/docs/how-tos/devices#creating_a_device_registry) Cloud Pub/Sub topic.

Messages are forwarded to the default Cloud Pub/Sub topic instead of the additional Cloud Pub/Sub topic in the following cases:

- No subfolder is specified in the MQTT topic
- A subfolder is specified in the MQTT topic, but it doesn't have a matching Pub/Sub topic in the device registry

For example, if the device publishes to the MQTT topic `/devices/DEVICE_ID/events/alerts`, the subfolder is the string `alerts`. Messages are forwarded to the additional Cloud Pub/Sub topic if the `eventNotificationConfigs[i].subfolderMatches` (/iot/docs/reference/cloudiot/rest/v1/projects.locations.registries#eventnotificationconfig) and `eventNotificationConfigs[i].pubsubTopicName`

(/iot/docs/reference/cloudiot/rest/v1/projects.locations.registries#eventnotificationconfig) fields are both set to `alerts`. Otherwise, messages are forward to the default Cloud Pub/Sub topic.

Connected devices can report device state by issuing a `PUBLISH` message to the following MQTT topic:

To categorize and retrieve state messages, configure the registry (/iot/docs/how-tos/devices#creating_a_device_registry) with a device state topic. The device state topic is the Cloud Pub/Sub topic specified in the `StateNotificationConfig.pubsubTopicName` (/iot/docs/reference/cloudiot/rest/v1/projects.locations.registries#statenotificationconfig) field. If the registry is configured with a device state topic, these messages are forwarded to the matching Cloud Pub/Sub topic on a best-effort basis.

Subfolders are not supported for device state messages. Devices that attempt to publish state messages to an MQT with a subfolder will be automatically disconnected.

For more details on retrieving state messages, see Getting device state (https://cloud.google.com/iot/docs/how-tos/config/getting-state).

Cloud IoT Core limits projects that generate excessive load. When devices retry failed operations without waiting, they can trigger limits that affect all devices in the same Google Cloud project.

For retries, you are strongly encouraged to implement a truncated exponential backoff algorithm (/iot/docs/how-tos/exponential-backoff) with introduced jitter.

When sending the initial MQTT `CONNECT` message from a client, you can supply an optional "keep-alive" value. This value is a time interval, measured in seconds, during which the broker expects a client to send a message, such as a `PUBLISH` message. If no message is sent from the client to the broker during the interval, the broker automatically closes the connection. Note that the keep-alive value you specify is multiplied by 1.5, so setting a 10-minute keep-alive actually results in a 15 minute interval.

For more information, see the MQTT specification (http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html#_Keep_Alive).

Cloud IoT Core does not supply its own default keep-alive value; if you choose to specify a keep-alive interval, you must set it in the client.

For best results, set the client's keep-alive interval to a minimum of 60 seconds. Many open source client libraries, including the Paho MQTT libraries for C (https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/_m_q_t_t_client_8h.html#aefd7c865f2641c8155b763fdf3061c25), Python (https://pypi.python.org/pypi/paho-mqtt/1.1#connect-reconnect-disconnect), Node.js (https://www.npmjs.com/package/mqtt#mqttclientstreambuilder-options), and Java (https://www.eclipse.org/paho/files/javadoc/constant-values.html#org.eclipse.paho.client.mqttv3.MqttConnectOptions.html#KEEP_ALIVE_INTERVAL_DEFAULT), use 60 seconds by default.

Separate from the keep-alive interval, Cloud IoT Core has its own idle time limit of 20 minutes (/iot/quotas). Based on this limit, a client connection will automatically be terminated if the client doesn't send any messages for 20 minutes, even if the keep-alive interval is longer. If a keep-alive value isn't supplied, the default idle timeout of 20 minutes still takes effect.

If you have trouble connecting, see Troubleshooting (/iot/docs/troubleshooting).