

# Lecture #8

## Cloud IoT

Spring 2024

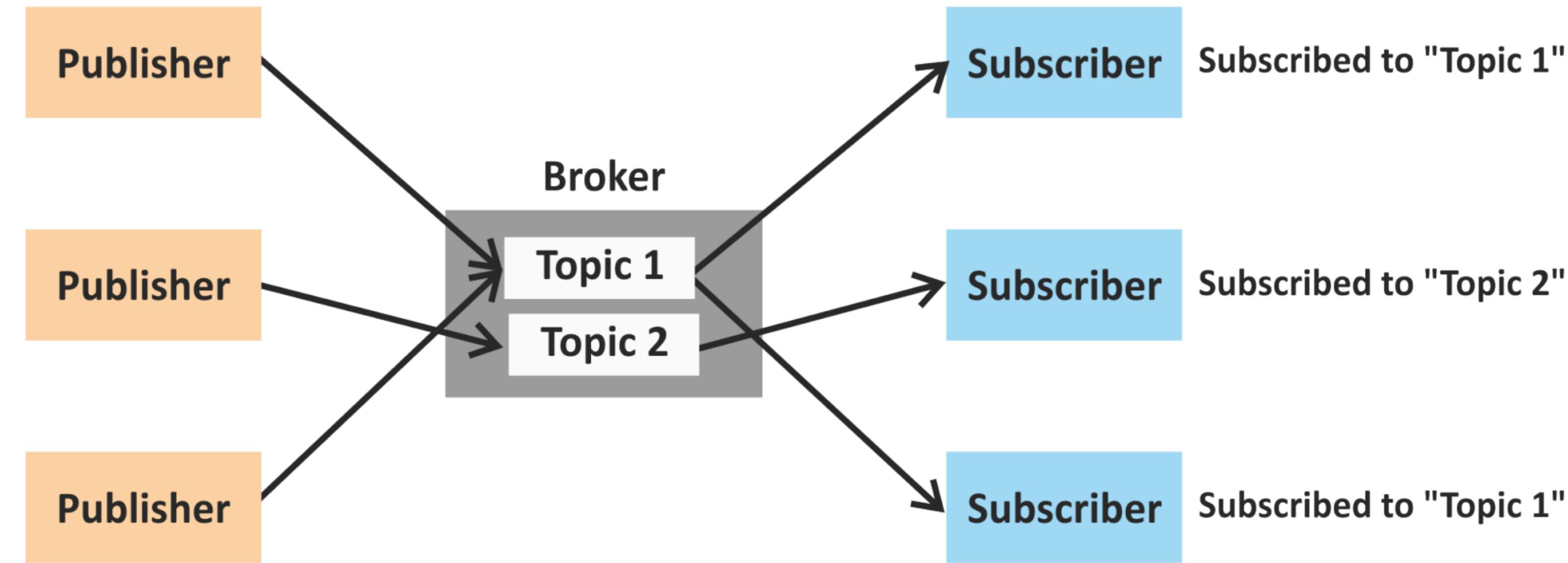
# MQTT

Message Queue Telemetry Transport



- Publish-subscribe.
- A message **broker** is required.
- Standard: ISO/IEC PRF 20922.
- Developed in 1999 (and released royalty free in 2010).
- Small code footprint.
- Limited network bandwidth / constrained environments.
- Data agnostic.

# Model



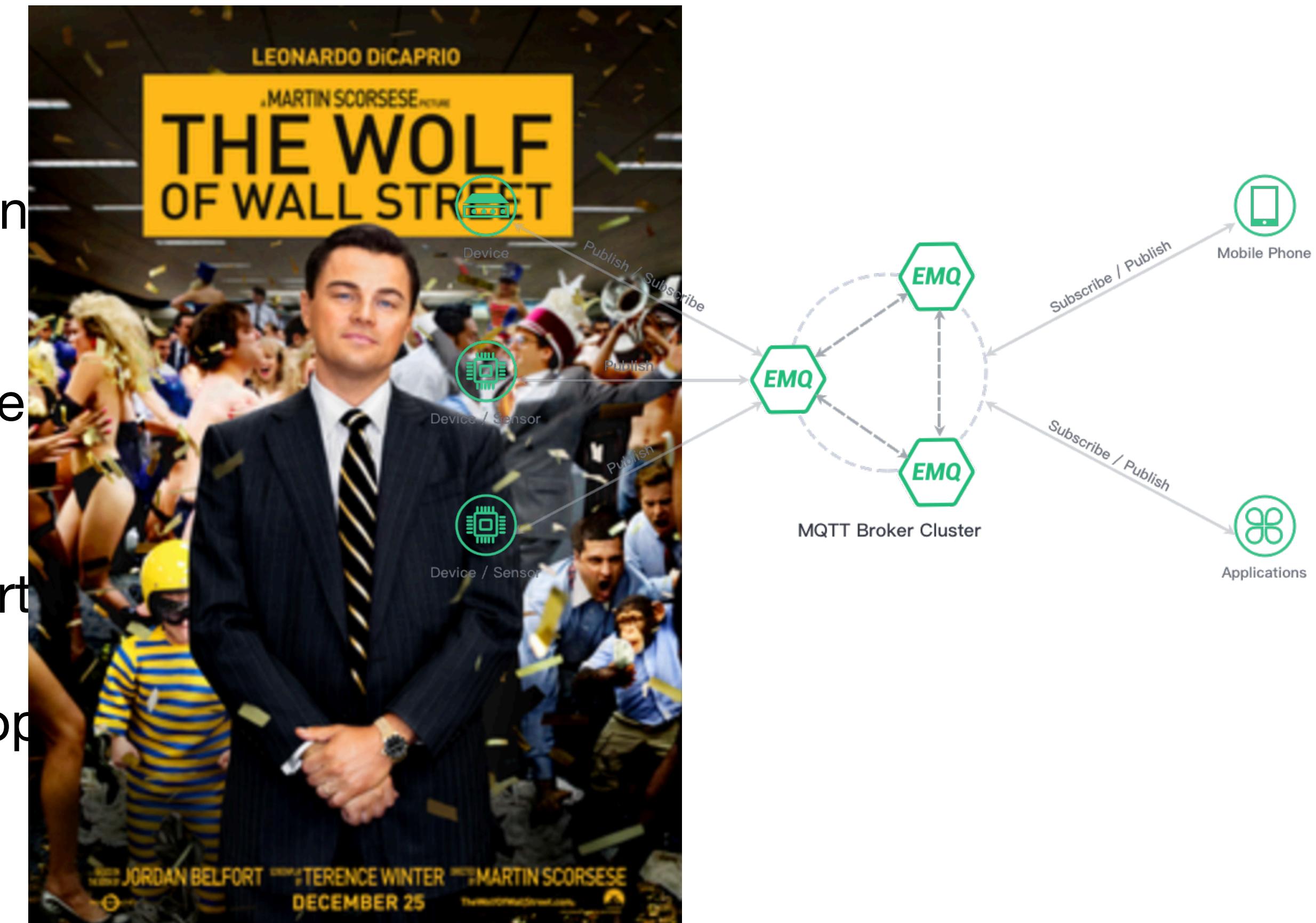
# Broker Benefits

- Eliminates insecure connections.
- Easily scales.
- Manages client connect states.
- Reduce network strain.

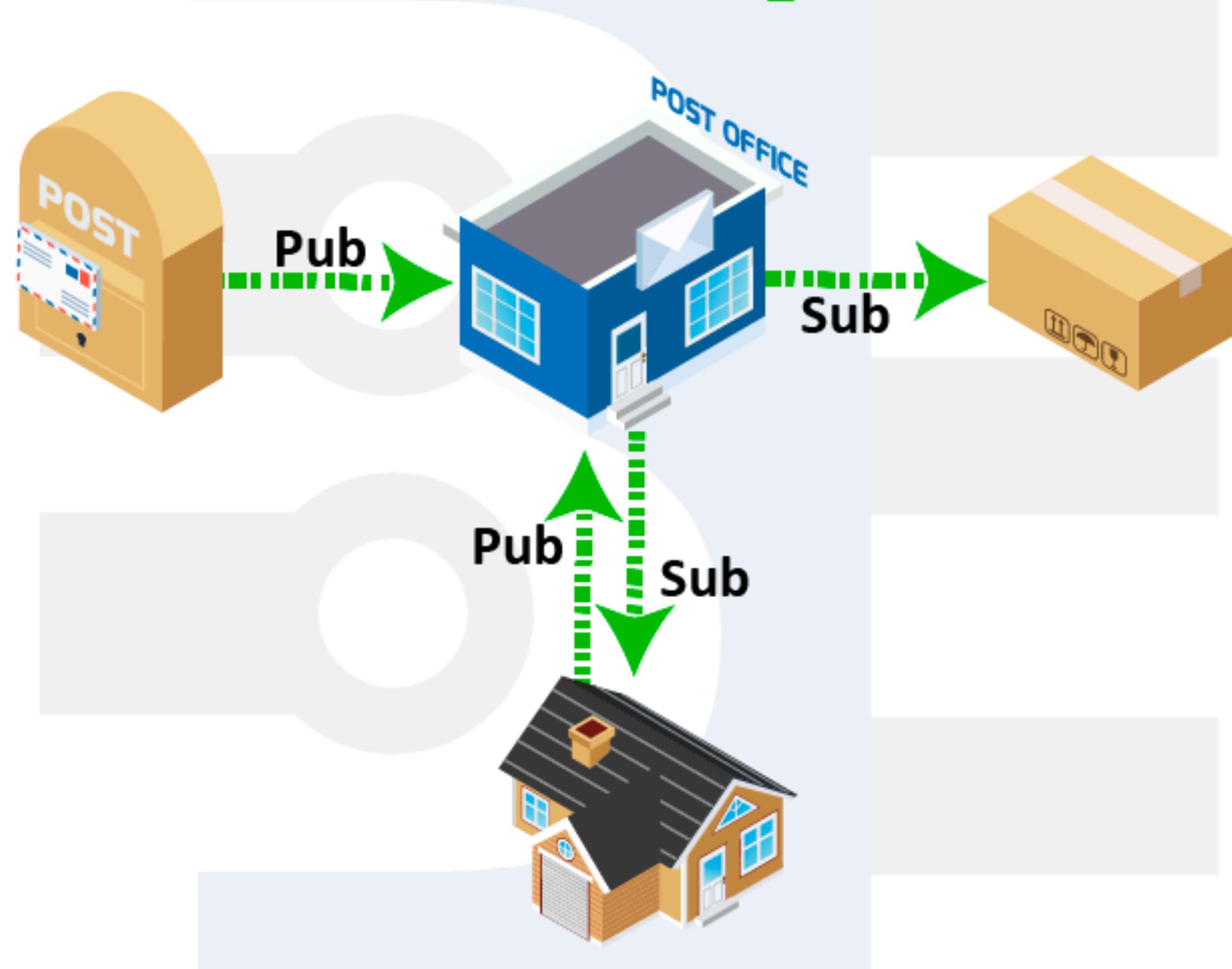


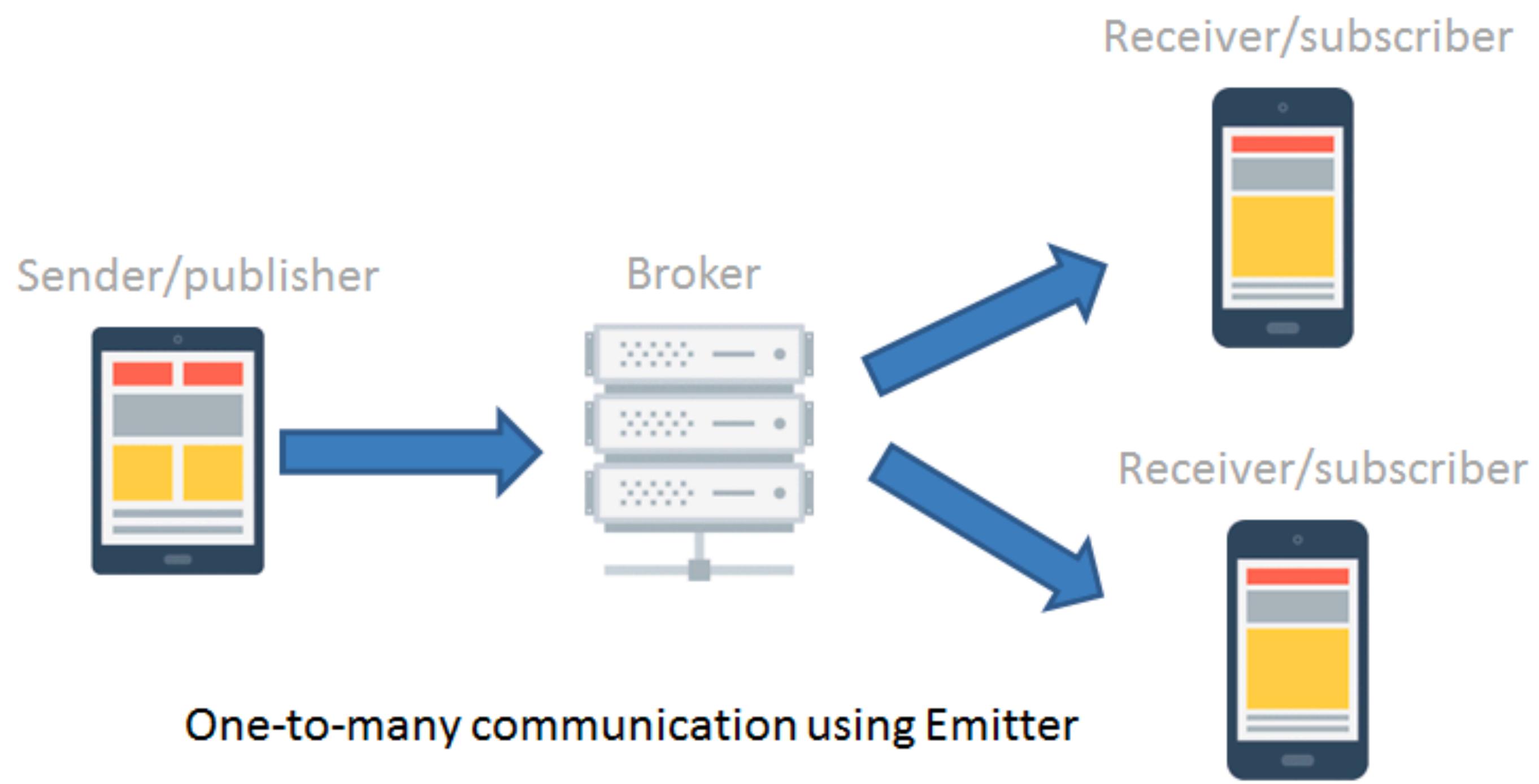
# What is a broker

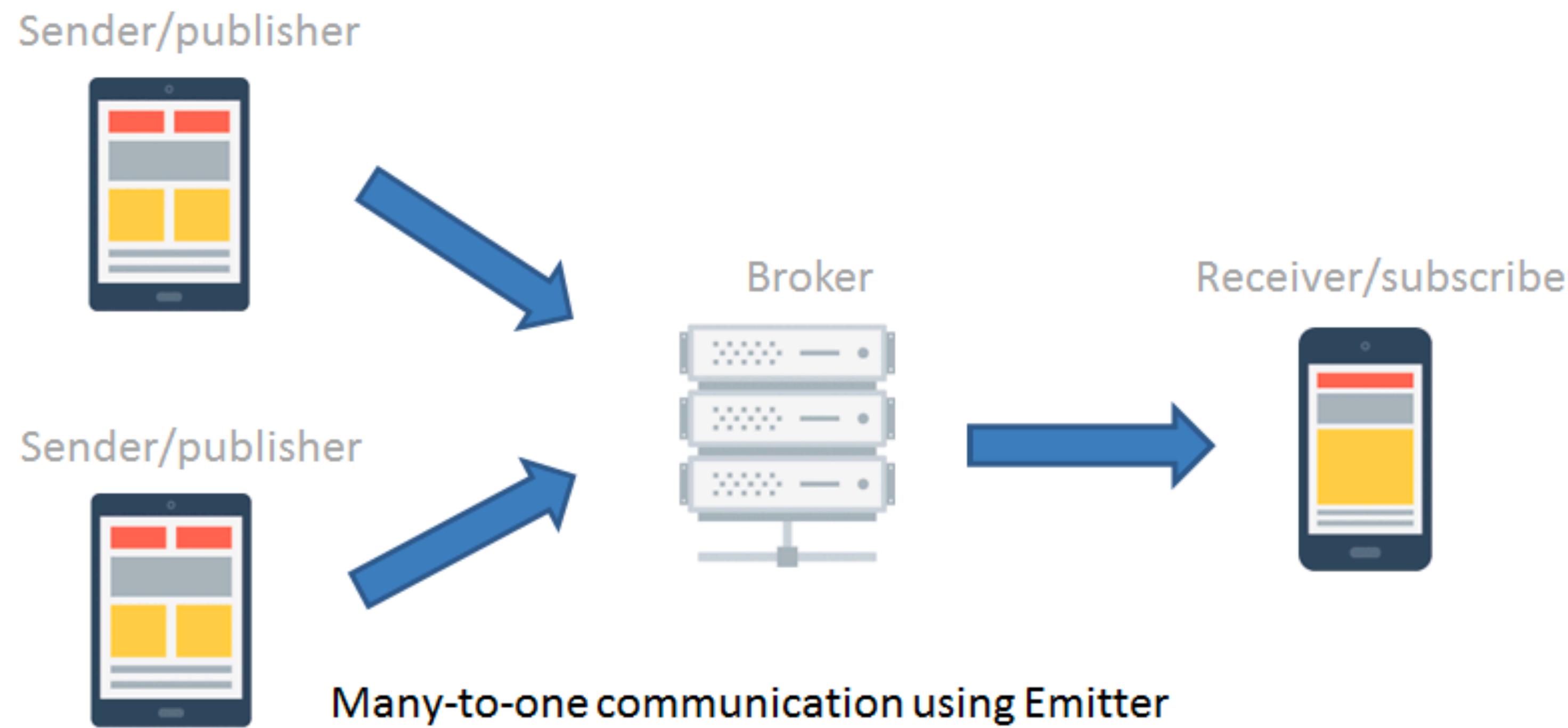
- Software running on computer.
- Running on premise cloud.
- Self-Built or 3rd party.
- Open source or propri

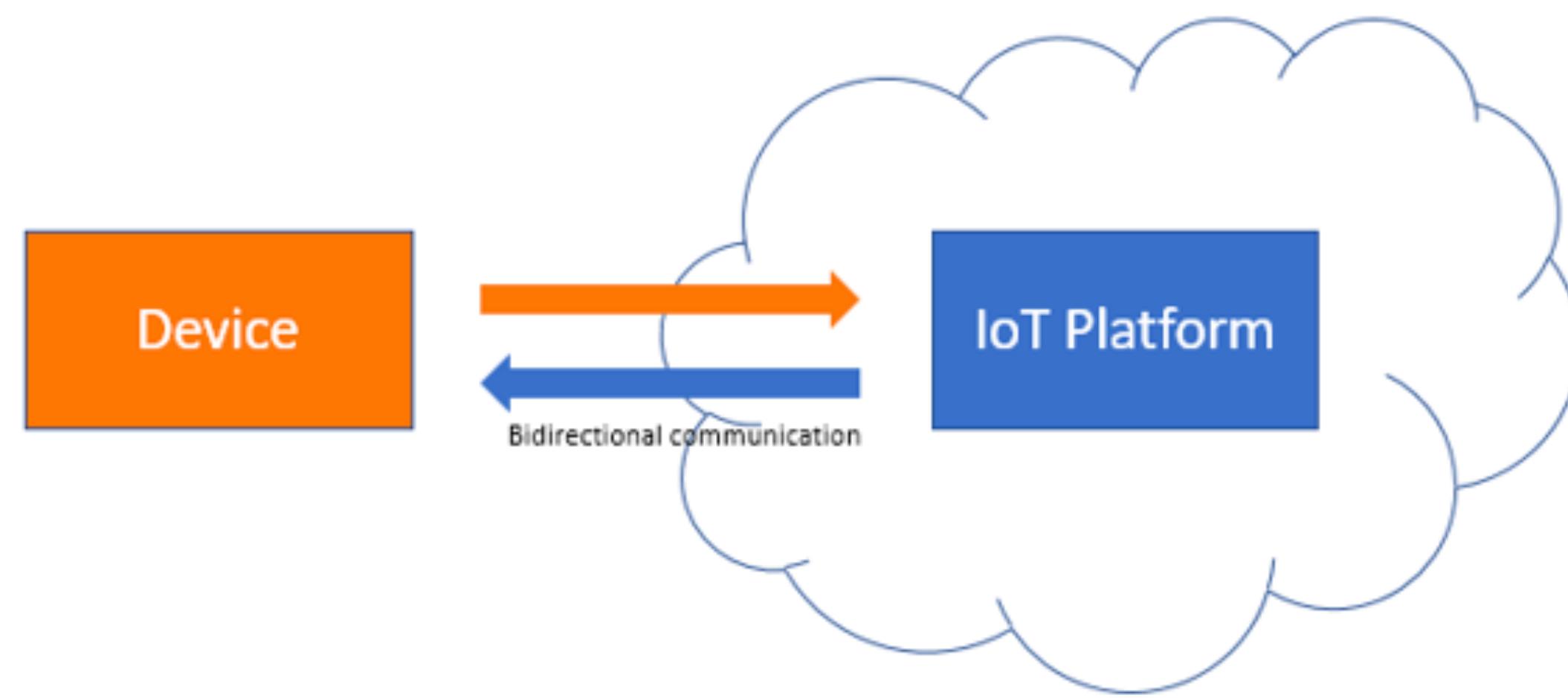


`mqtt://broker/topic/message`









# Connecting to the broker

<b>Value</b>	<b>Return Code Response</b>	<b>Description</b>
0	0x00 Connection Accepted	Connection accepted
1	0x01 Connection Refused, unacceptable protocol version	The Server does not support the level of the MQTT protocol requested by the Client
2	0x02 Connection Refused, identifier rejected	The Client identifier is correct UTF-8 but not allowed by the Server
3	0x03 Connection Refused, Server unavailable	The Network Connection has been made but the MQTT service is unavailable
4	0x04 Connection Refused, bad user name or password	The data in the user name or password is malformed
5	0x05 Connection Refused, not authorized	The Client is not authorized to connect
6-255		Reserved for future use

# Publishing to a topic



# Subscribing to a topic

- Example:
  - Topic #1: home/groundfloor/kitchen/temperature
  - Topic #2: office/conferenceroom/luminance
- Wild cards
  - Single-level:
    - home/groundfloor/+/temperature (to subscribe to **all the temperature readings** in all the rooms of the ground floor)
  - Multi-level:
    - home/groundfloor/# (to subscribe to **all the readings** in all the rooms of the ground floor, **not only the temperature**)

# Quality of Service

- **0:** The broker/client will deliver the message once, with no confirmation.
- **1:** The broker/client will deliver the message at least once, with confirmation required.
- **2:** The broker/client will deliver the message exactly once by using a four step handshake.



# Last will and testament

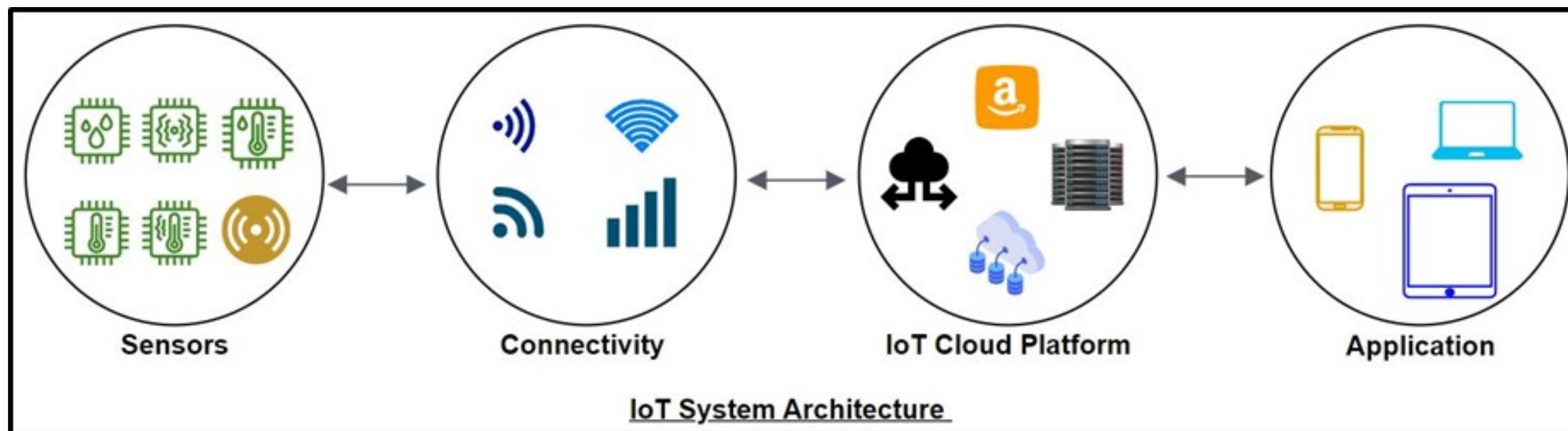
MQTT-Packet:	
CONNECT	
contains:	Example
<code>clientId</code>	"client-1"
<code>cleanSession</code>	true
<code>username</code> (optional)	"hans"
<code>password</code> (optional)	"letmein"
<code>lastWillTopic</code> (optional)	"/hans/will"
<code>lastWillQos</code> (optional)	2
<code>lastWillMessage</code> (optional)	"unexpected exit"
<code>keepAlive</code>	60

# Learn More

- Learn more: [mqtt.org](https://mqtt.org)
- Software: [mqtt.org/software](https://mqtt.org/software)
- Recommended broker (C):  
[Mosquitto \(mosquitto.org\)](https://mosquitto.org)
- Lots of good tutorials out there  
on Android Things, Python,  
Java and Mobile.



# Architecture of an IoT System



# Top 7 IoT Cloud Platforms

- Amazon Web Services (AWS) IoT Platform
- Microsoft Azure IoT
- Google IoT
- IBM Watson IoT
- Cisco IoT Cloud Connect
- ThingsBoard Open-Source IoT Platform
- Oracle IoT Intelligent Applications

# AWS IoT

Monitor

## Connect

Connect one device

▶ Connect many devices

Test

▶ Device Advisor

MQTT test client

Device Location [New](#)

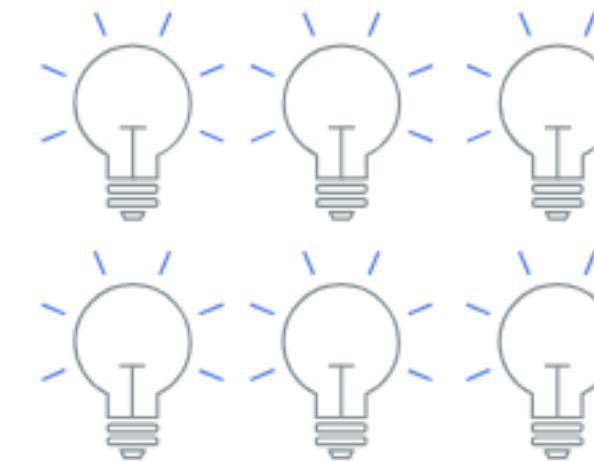
Manage

▶ All devices

▶ Greengrass devices

## How it works

Connect devices to AWS IoT so they can send and receive data. **Bold** text refers to an entry in the **Connect** menu of the navigation pane.



### Connect one device

The **Quick connect** wizard walks you through the steps to create the resources and download the software required to connect your IoT device to AWS IoT.

### Connect many devices

**Fleet provisioning templates** define security policies and registry settings when a device connects to AWS IoT for the first time.

# Create a thing object

AWS IoT

Monitor

Connect

▶ Connect one device **Connect one device**

▶ Connect many devices

Test

▶ Device Advisor

MQTT test client

Manage

▶ All devices

▶ Greengrass devices

▶ LPWAN devices

▶ Remote actions

▶ Message Routing

Retained messages

▶ Security

▶ Fleet Hub

Device Software

Billing groups

Settings

Feature spotlight

AWS IoT > Connect > Connect one device

Step 1  
**Prepare your device**

Step 2  
Register and secure your device

Step 3  
Choose platform and SDK

Step 4  
Download connection kit

Step 5  
Run connection kit

## Prepare your device Info

### How it works



In this wizard, we'll be creating a thing resource in AWS IoT. A thing resource is a digital representation of a physical device or logical entity.

A thing resource uses certificates to secure communication between your device and AWS IoT. AWS IoT policies control access to the AWS IoT resources. This wizard creates the certificate and policy for your device.

When a device connects to AWS IoT, policies enable it to subscribe and publish MQTT messages with AWS IoT message broker.

### Prepare your device

1. Turn on your device and make sure it's connected to the internet.
2. Choose how you want to load files onto your device.
  1. If your device supports a browser, open the AWS IoT console on your device and run this wizard. You can download the files directly to your device from the browser.
  2. If your device doesn't support a browser, choose the best way to transfer files from the computer with the browser to your device. Some options to transfer files include using the file transfer protocol (FTP) and using a USB memory stick.
3. Make sure that you can access a command-line interface on your device.
  1. If you're running this wizard on your IoT device, open a terminal window on your device to access a command-line interface.

# Register and secure your device

AWS IoT

- Monitor
- Connect
  - Connect one device**
  - ▶ Connect many devices
- Test
  - ▶ Device Advisor
  - MQTT test client
- Manage
  - ▶ All devices
  - ▶ Greengrass devices
  - ▶ LPWAN devices
  - ▶ Remote actions
  - ▶ Message Routing
  - Retained messages
  - ▶ Security
  - ▶ Fleet Hub
- Device Software
- Billing groups
- Settings
- Feature spotlight
- Documentation 

AWS IoT > Connect > Connect one device

Step 1  
Prepare your device

Step 2  
**Register and secure your device**

Step 3  
Choose platform and SDK

Step 4  
Download connection kit

Step 5  
Run connection kit

## Register and secure your device Info

### Represent your device in the cloud

A thing resource is a digital representation of a physical device or logical entity in AWS IoT. A thing resource lets your device use AWS IoT features such as Device Shadows, events, jobs, and other device management features. Certificates authenticate your device, and policies authorize access to other AWS resources and actions.

This wizard helps you create the thing resource, policy, and certificate resources necessary to connect your device to AWS IoT so that it can publish simple messages. After you complete this wizard, you can edit the resources to explore AWS IoT features further.

### Thing properties

Create a new thing  Choose an existing thing

Thing name  
  
Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

### Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

- ▶ Thing type - *optional*
- ▶ Searchable thing attributes - *optional*

# Choose platform and SDK

AWS IoT > Connect > Connect one device

Step 1  
Prepare your device

Step 2  
Register and secure your device

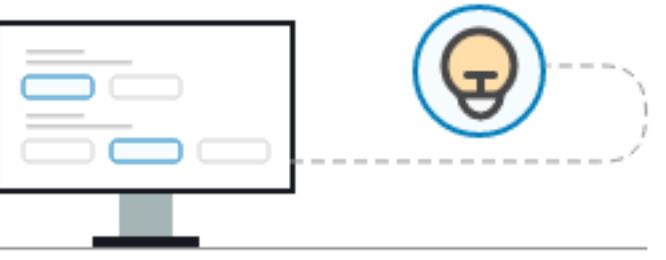
Step 3  
**Choose platform and SDK**

Step 4  
Download connection kit

Step 5  
Run connection kit

## Choose platform and SDK Info

### Choose the software for your device



This wizard helps you download a software development kit (SDK) to your device. AWS IoT supports Device SDKs that run on your device and include a sample program that publishes and subscribes to MQTT messages. AWS IoT supports Device SDKs in the languages shown below.

### Platform and SDK

Choose the platform OS and AWS IoT Device SDK that you want to use for your device.

**Device platform operating system**  
This is the operating system installed on the device that will connect to AWS.

**Linux / macOS**  
Linux version: any  
macOS version: 10.13+

**Windows**  
Version 10

**AWS IoT Device SDK**  
Choose a Device SDK that's in a language your device supports.

**Node.js**  
Version 10+  
Requires Node.js and npm to be installed

**Python**  
Version 3.6+  
Requires Python and Git to be installed

**Java**  
Version 8  
Requires Java JDK, Maven, and Git to be installed

# Download files to your device

AWS IoT > Connect > Connect one device

Step 1  
Prepare your device

Step 2  
Register and secure your device

Step 3  
Choose platform and SDK

Step 4  
Download connection kit

Step 5  
Run connection kit

## Download connection kit Info

**Install the software on your device**

We created the AWS IoT resources that your device needs to connect to AWS IoT. We also created a connection kit that includes the resources in a zipped file that you need to install on your device. The resources in the connection kit are listed below. In this step, you'll install them on your device.

 → 

**Connection kit**

Certificate TutorialTestThing.cert.pem	Private key TutorialTestThing.private.key	AWS IoT Device SDK Python
Script to send and receive messages start.sh	Policy TutorialTestThing-Policy <a href="#">View policy</a>	

**Download**

If you are running this from a browser on the device, after you download the connection kit, it will be in the browser's download folder.

If you are not running this from a browser on your device, you'll need to transfer the connection kit from your browser's download folder to your device using the method you tested when you prepared your device in step 1.

 [Download connection kit](#)

# Download files to your device

AWS IoT > Connect > Connect one device

Step 1  
[Prepare your device](#)

Step 2  
[Register and secure your device](#)

Step 3  
[Choose platform and SDK](#)

Step 4  
**Download connection kit**

Step 5  
[Run connection kit](#)

## Download connection kit Info

**Install the software on your device**

We created the AWS IoT resources that your device needs to connect to AWS IoT. We also created a connection kit that includes the resources in a zipped file that you need to install on your device. The resources in the connection kit are listed below. In this step, you'll install them on your device.

 → 

**Connection kit**

Certificate <a href="#">TutorialTestThing.cert.pem</a>	Private key <a href="#">TutorialTestThing.private.key</a>	AWS IoT Device SDK Python
Script to send and receive messages <a href="#">start.sh</a>	Policy <a href="#">TutorialTestThing-Policy</a> <a href="#">View policy</a>	

**Download**

If you are running this from a browser on the device, after you download the connection kit, it will be in the browser's download folder.

If you are not running this from a browser on your device, you'll need to transfer the connection kit from your browser's download folder to your device using the method you tested when you prepared your device in step 1.

 [Download connection kit](#)

# Run the sample

AWS IoT > Connect > Connect one device

Step 1  
Prepare your device

Step 2  
Register and secure your device

Step 3  
Choose platform and SDK

Step 4  
Download connection kit

Step 5  
Run connection kit

## Run connection kit Info

### How to display messages from your device

**Step 1: Add execution permissions**  
On the device, launch a terminal window to copy and paste the command to add execution permissions.

```
chmod +x start.sh
```

**Step 2: Run the start script**  
On the device, copy and paste the command to the terminal window and run the start script.

```
./start.sh
```

**Step 3: Return to this screen to view your device's messages**  
After running the start script, return to this screen to see the messages between your device and AWS IoT. The messages from your device appear in the following list.

Subscriptions	sdk/test/Python	Pause	Clear
sdk/test/Python	Waiting for messages		

# Output

```
Running pub/sub sample application...
Connecting to a13hikvzkye6lx-ats.iot.us-east-1.amazonaws.com with client ID 'basicPubSub'...
Connected!
Subscribing to topic 'sdk/test/Python'...
Subscribed with QoS.AT LEAST_ONCE
Sending messages until program killed
Publishing message to topic 'sdk/test/Python': Hello World! [1]
Received message from topic 'sdk/test/Python': b'Hello World! [1]'
Publishing message to topic 'sdk/test/Python': Hello World! [2]
Received message from topic 'sdk/test/Python': b'Hello World! [2]'
Publishing message to topic 'sdk/test/Python': Hello World! [3]
Received message from topic 'sdk/test/Python': b'Hello World! [3]'
```

# Output

AWS IoT > Connect > Connect one device

Step 1  
Prepare your device

Step 2  
Register and secure your device

Step 3  
Choose platform and SDK

Step 4  
Download connection kit

Step 5  
Run connection kit

## Run connection kit Info

### How to display messages from your device

**Step 1: Add execution permissions**  
On the device, launch a terminal window to copy and paste the command to add execution permissions.

```
chmod +x start.sh
```

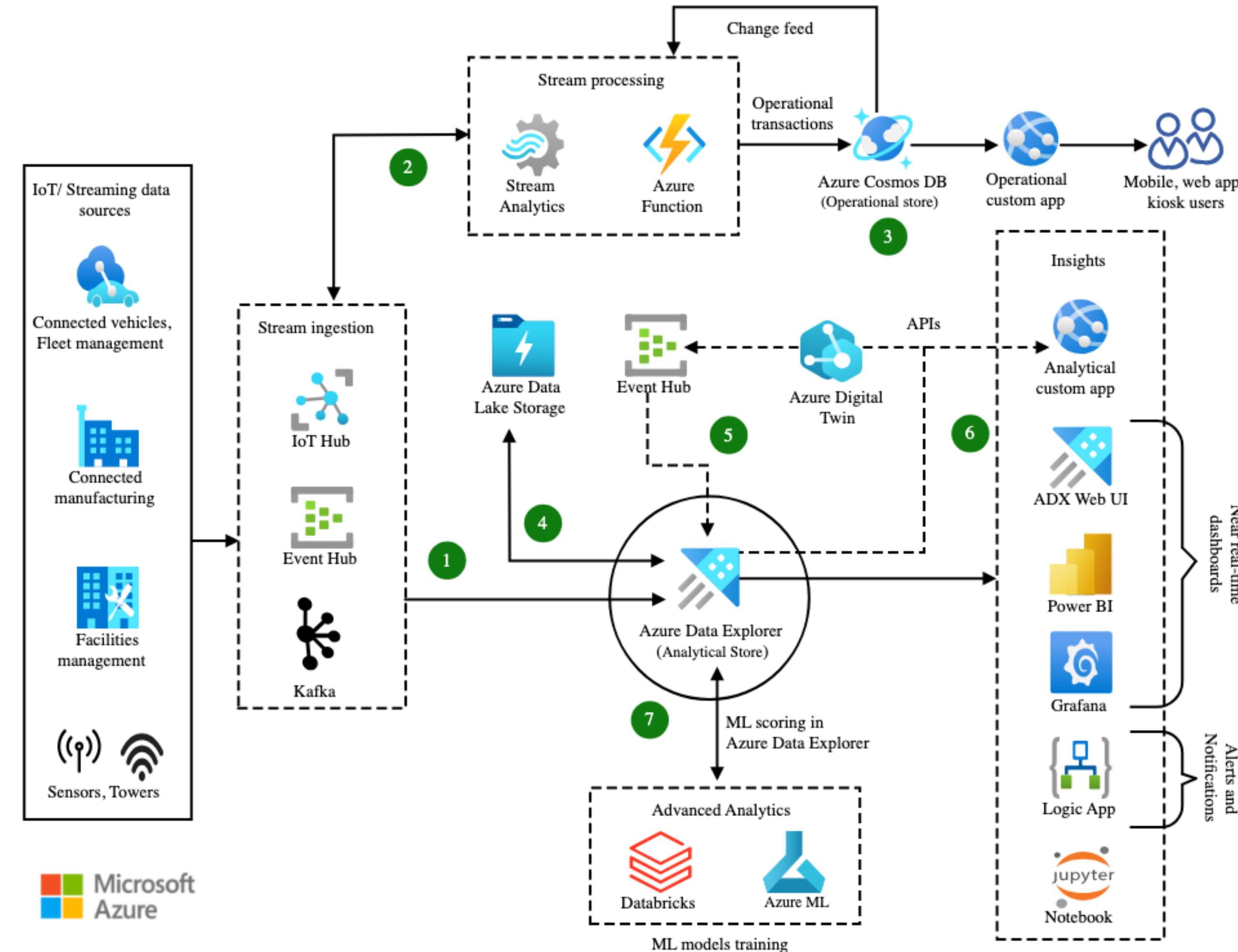
**Step 2: Run the start script**  
On the device, copy and paste the command to the terminal window and run the start script.

```
./start.sh
```

**Step 3: Return to this screen to view your device's messages**  
After running the start script, return to this screen to see the messages between your device and AWS IoT. The messages from your device appear in the following list.

Subscriptions	sdk/test/Python	Resume	Clear
sdk/test/Python	<p>▼ sdk/test/Python September 14, 2022, 10:47:44 (UTC-0700)</p> <p>"Hello World! [3]"</p>		
	<p>▼ sdk/test/Python September 14, 2022, 10:47:43 (UTC-0700)</p> <p>"Hello World! [2]"</p>		

# Azure IoT



# Launch the Cloud Shell

To launch the Cloud Shell:

1. Select the **Cloud Shell** button on the top-right menu bar in the Azure portal.



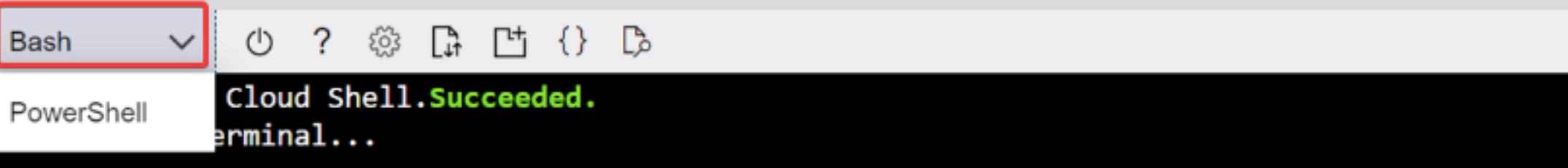
ⓘ Note

If this is the first time you've used the Cloud Shell, it prompts you to create storage, which is required to use the Cloud Shell. Select a subscription to create a storage account and Microsoft Azure Files share.

2. Select your preferred CLI environment in the **Select environment** dropdown. This quickstart uses the **Bash** environment. You can also use the **PowerShell** environment.

ⓘ Note

Some commands require different syntax or formatting in the **Bash** and **PowerShell** environments. For more information, see [Tips for using the Azure CLI successfully](#).



Bash

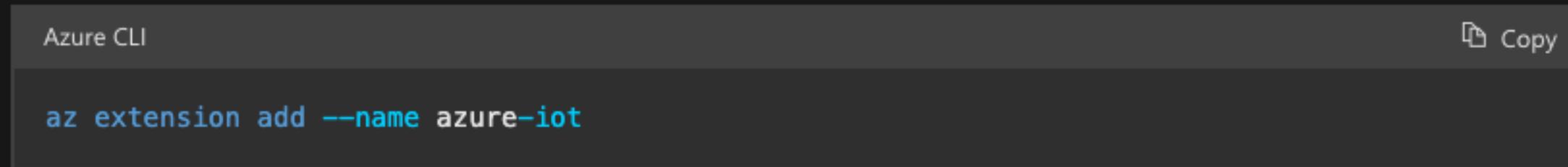
PowerShell

Cloud Shell.Succeeded.

terminal...

# Prepare CLI sessions

- In the first CLI session, run the `az extension add` command. The command adds the Microsoft Azure IoT Extension for Azure CLI to your CLI shell. The IOT Extension adds IoT Hub, IoT Edge, and IoT Device Provisioning Service (DPS) specific commands to Azure CLI.



Azure CLI

az extension add --name azure-iot

Copy

After you install the Azure IOT extension, you don't need to install it again in any Cloud Shell session.

## ⓘ Note

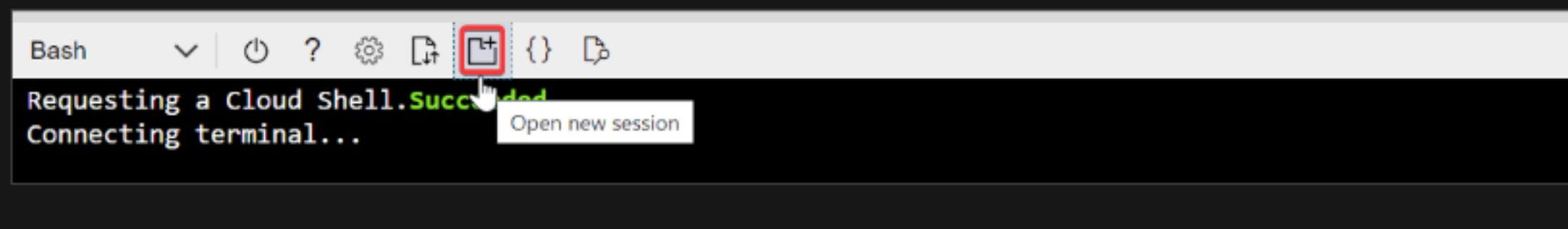
This article uses the newest version of the Azure IoT extension, called `azure-iot`. The legacy version is called `azure-cli-iot-ext`. You should only have one version installed at a time. You can use the command `az extension list` to validate the currently installed extensions.

Use `az extension remove --name azure-cli-iot-ext` to remove the legacy version of the extension.

Use `az extension add --name azure-iot` to add the new version of the extension.

To see what extensions you have installed, use `az extension list`.

- Open the second CLI session. If you're using the Cloud Shell in a browser, use the **Open new session** button. If using the CLI locally, open a second CLI instance.



# Create an IoT hub

1. In the first CLI session, run the `az group create` command to create a resource group. The following command creates a resource group named *MyResourceGroup* in the *eastus* location.

Azure CLI

 Copy

```
az group create --name MyResourceGroup --location eastus
```

2. In the first CLI session, run the Az PowerShell module `iot hub create` command to create an IoT hub. It takes a few minutes to create an IoT hub.

*YourIoTHubName*. Replace this placeholder and the surrounding braces in the following command, using the name you chose for your IoT hub. An IoT hub name must be globally unique in Azure. Use your IoT hub name in the rest of this quickstart wherever you see the placeholder.

Azure CLI

 Copy

```
az iot hub create --resource-group MyResourceGroup --name {YourIoTHubName}
```

# Create and monitor a device

1. In the first CLI session, run the `az iot hub device-identity create` command. This command creates the simulated device identity.

*YourIoTHubName*. Replace this placeholder below with the name you chose for your IoT hub.

*simDevice*. You can use this name directly for the simulated device in the rest of this quickstart. Optionally, use a different name.

Azure CLI

 Copy

```
az iot hub device-identity create -d simDevice -n {YourIoTHubName}
```

2. In the first CLI session, run the `az iot device simulate` command. This command starts the simulated device. The device sends telemetry to your IoT hub and receives messages from it.

*YourIoTHubName*. Replace this placeholder below with the name you chose for your IoT hub.

Azure CLI

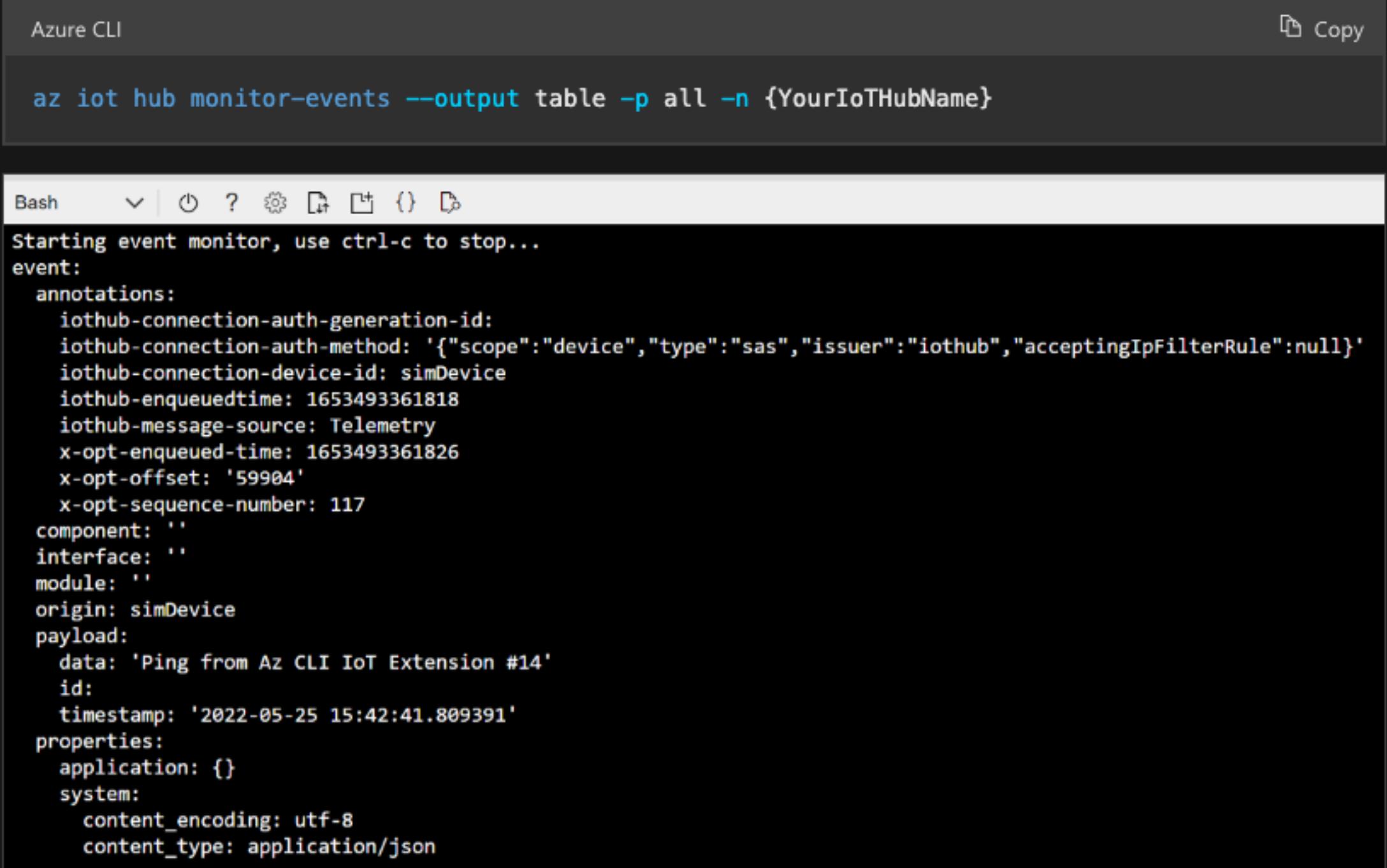
 Copy

```
az iot device simulate -d simDevice -n {YourIoTHubName}
```

# To monitor a device

1. In the second CLI session, run the `az iot hub monitor-events` command. This command continuously monitors the simulated device. The output shows telemetry such as events and property state changes that the simulated device sends to the IoT hub.

*YourIoTHubName*. Replace this placeholder below with the name you chose for your IoT hub.



Azure CLI

```
az iot hub monitor-events --output table -p all -n {YourIoTHubName}
```

Bash Starting event monitor, use ctrl-c to stop...  
event:  
annotations:  
iohub-connection-auth-generation-id:  
iohub-connection-auth-method: '{"scope":"device","type":"sas","issuer":"iohub","acceptingIpFilterRule":null}'  
iohub-connection-device-id: simDevice  
iohub-enqueuedtime: 1653493361818  
iohub-message-source: Telemetry  
x-opt-enqueued-time: 1653493361826  
x-opt-offset: '59904'  
x-opt-sequence-number: 117  
component: ''  
interface: ''  
module: ''  
origin: simDevice  
payload:  
data: 'Ping from Az CLI IoT Extension #14'  
id:  
timestamp: '2022-05-25 15:42:41.809391'  
properties:  
application: {}  
system:  
content\_encoding: utf-8  
content\_type: application/json

2. After you monitor the simulated device in the second CLI session, press **Ctrl+C** to stop monitoring. Keep the second CLI session open to use in later steps.

# Use the CLI to send a message

1. In the first CLI session, confirm that the simulated device is still running. If the device stopped, run the following command to restart it:

*YourIoTHubName*. Replace this placeholder below with the name you chose for your IoT hub.

```
Azure CLI Copy  
az iot device simulate -d simDevice -n {YourIoTHubName}
```

2. In the second CLI session, run the `az iot device c2d-message send` command. This command sends a cloud-to-device message from your IoT hub to the simulated device. The message includes a string and two key-value pairs.

*YourIoTHubName*. Replace this placeholder below with the name you chose for your IoT hub.

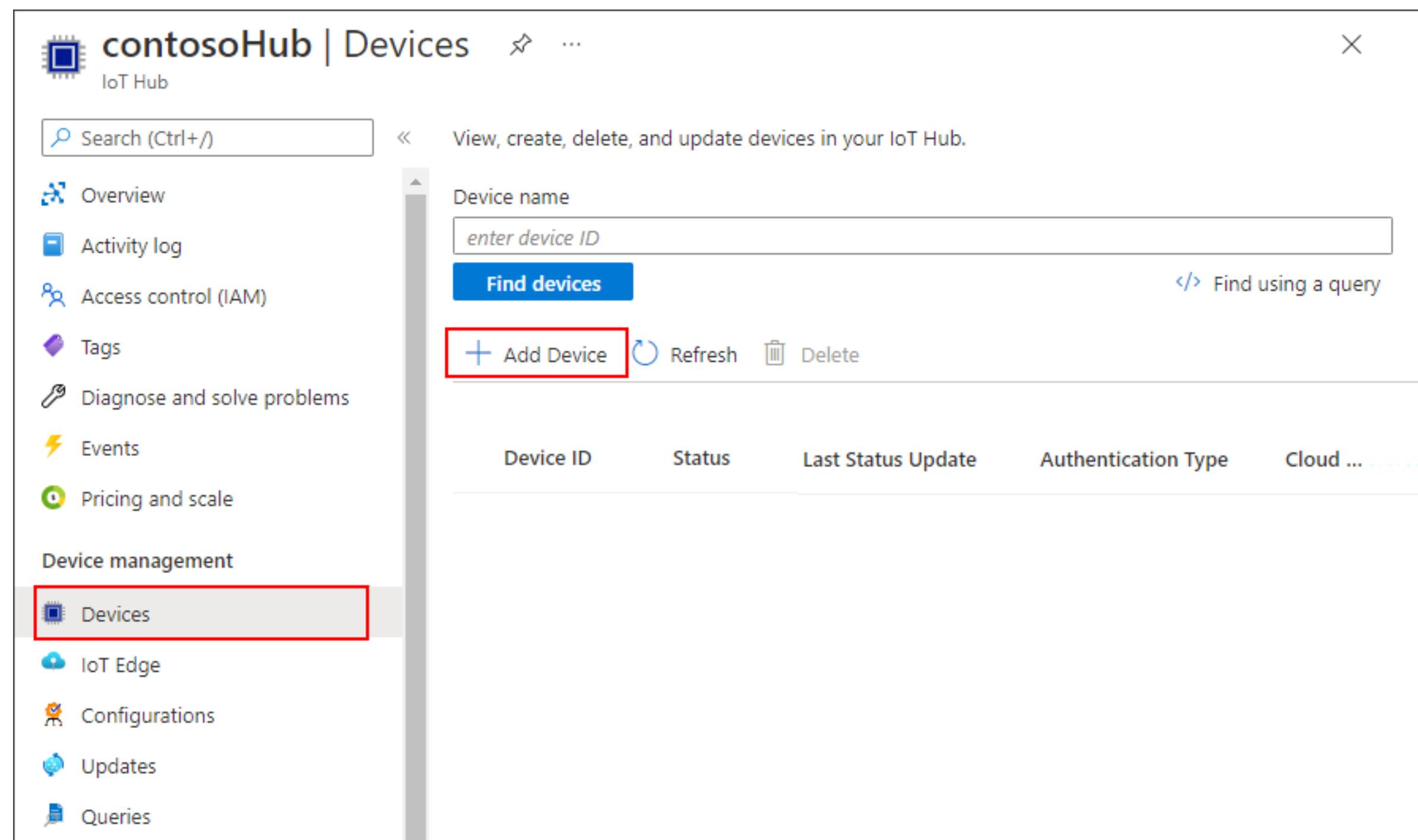
```
Azure CLI Copy  
az iot device c2d-message send -d simDevice --data "Hello World" --props "key0=value0;key1=value1"
```

Optionally, you can send cloud-to-device messages by using the Azure portal. To do this, browse to the overview page for your IoT Hub, select **IoT Devices**, select the simulated device, and select **Message to Device**.

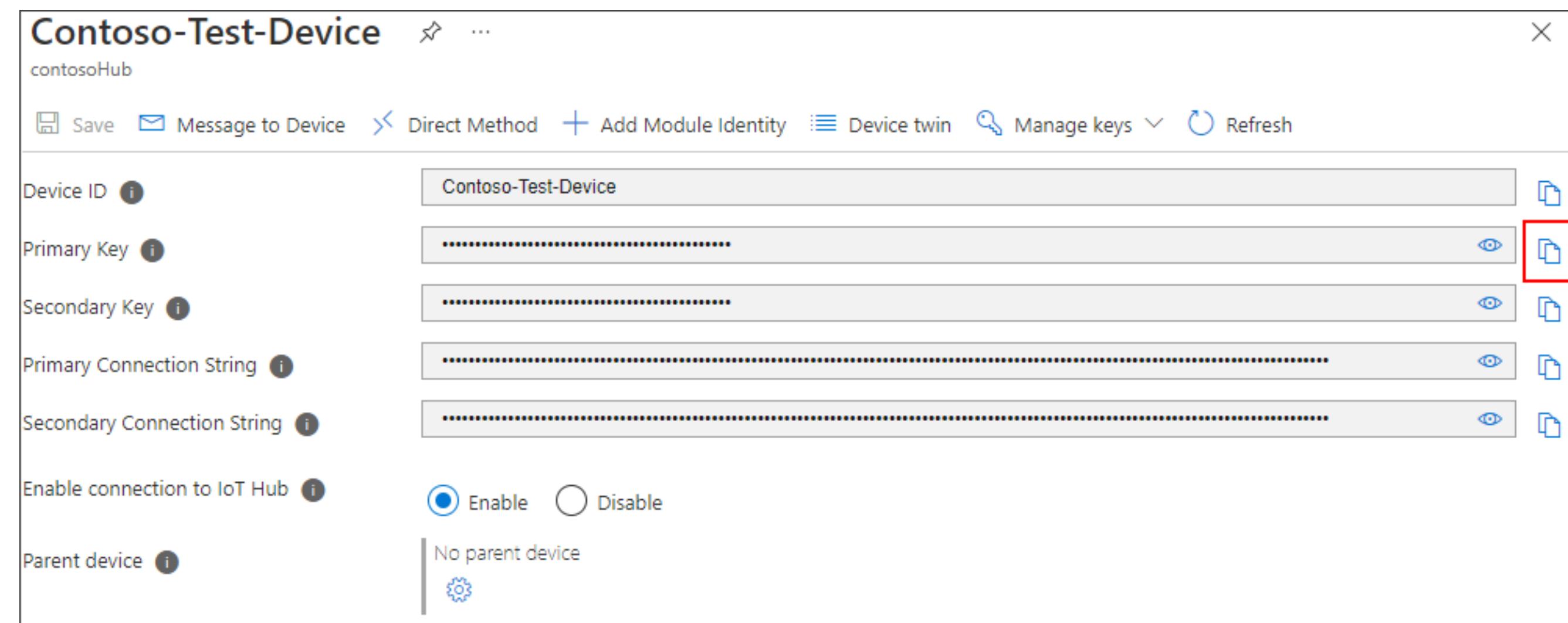
3. In the first CLI session, confirm that the simulated device received the message.

```
C2D Message Handler [Received C2D message]:  
{ 'Message Properties': { 'content_encoding': 'utf-8',  
                           'key0': 'value0',  
                           'key1': 'value1',  
                           'message_id':  
                             'Payload': 'Hello World',  
                             'Topic': '/devices/simDevice/messages/devicebound'}  
Device simulation in progress: 19%|#####
```

# Register a device and send messages to IoT Hub



# Credentials



<https://github.com/Azure/azure-iot-sdk-csharp>

# Google IoT

≡ Google Cloud Platform

## New Project

---

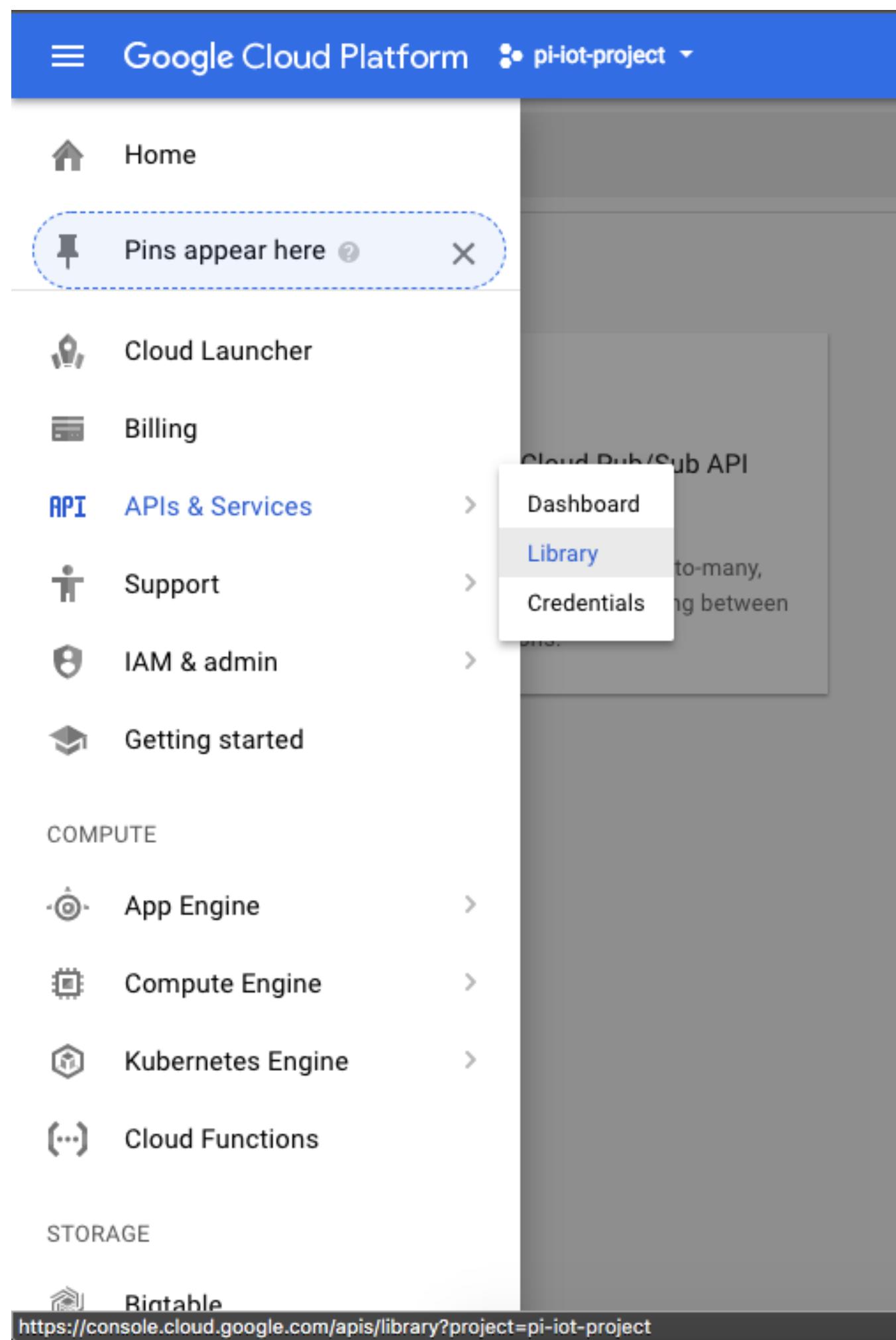
i You have 11 projects remaining in your quota. [Learn more.](#)

Project name ?  
pi-iot-project

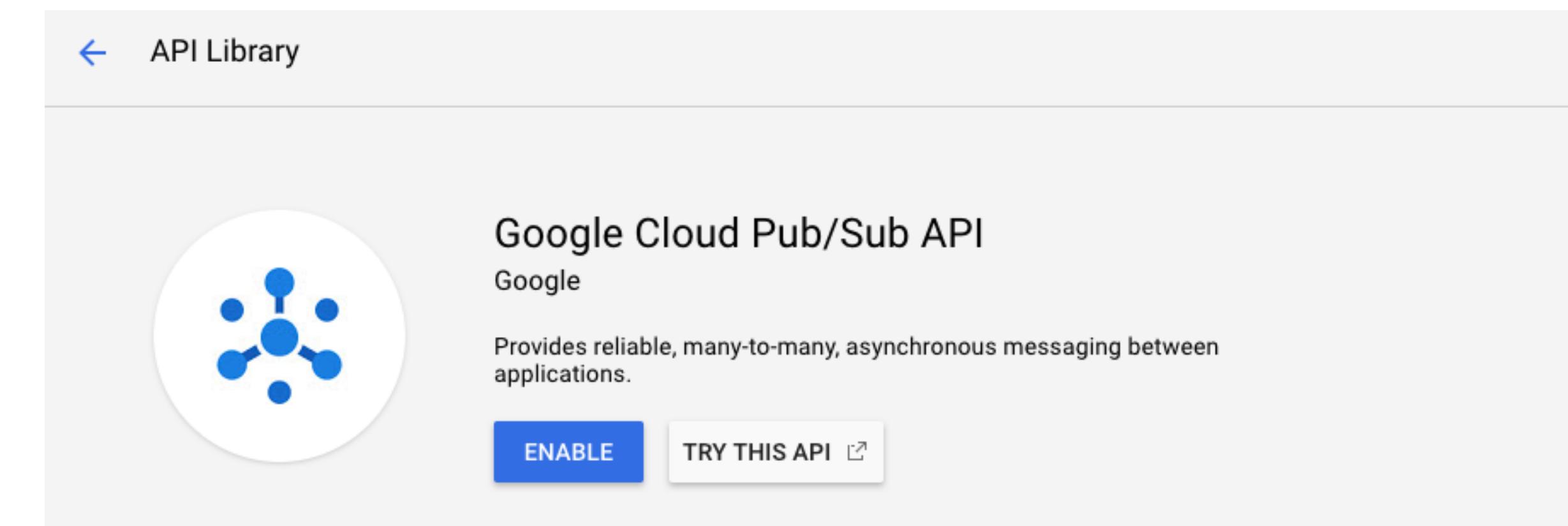
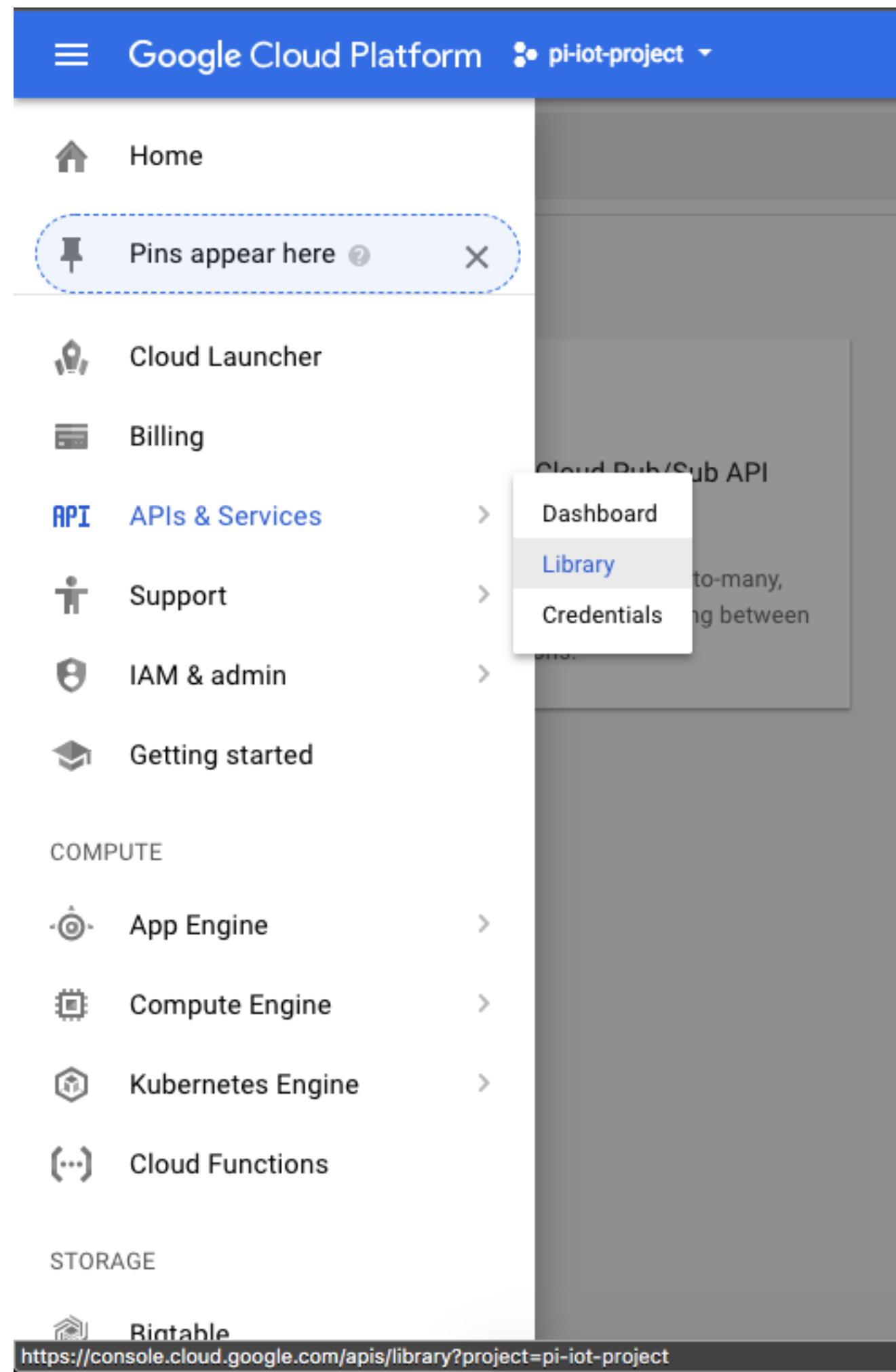
Project ID ?  
pi-iot-project ↻

Create Cancel

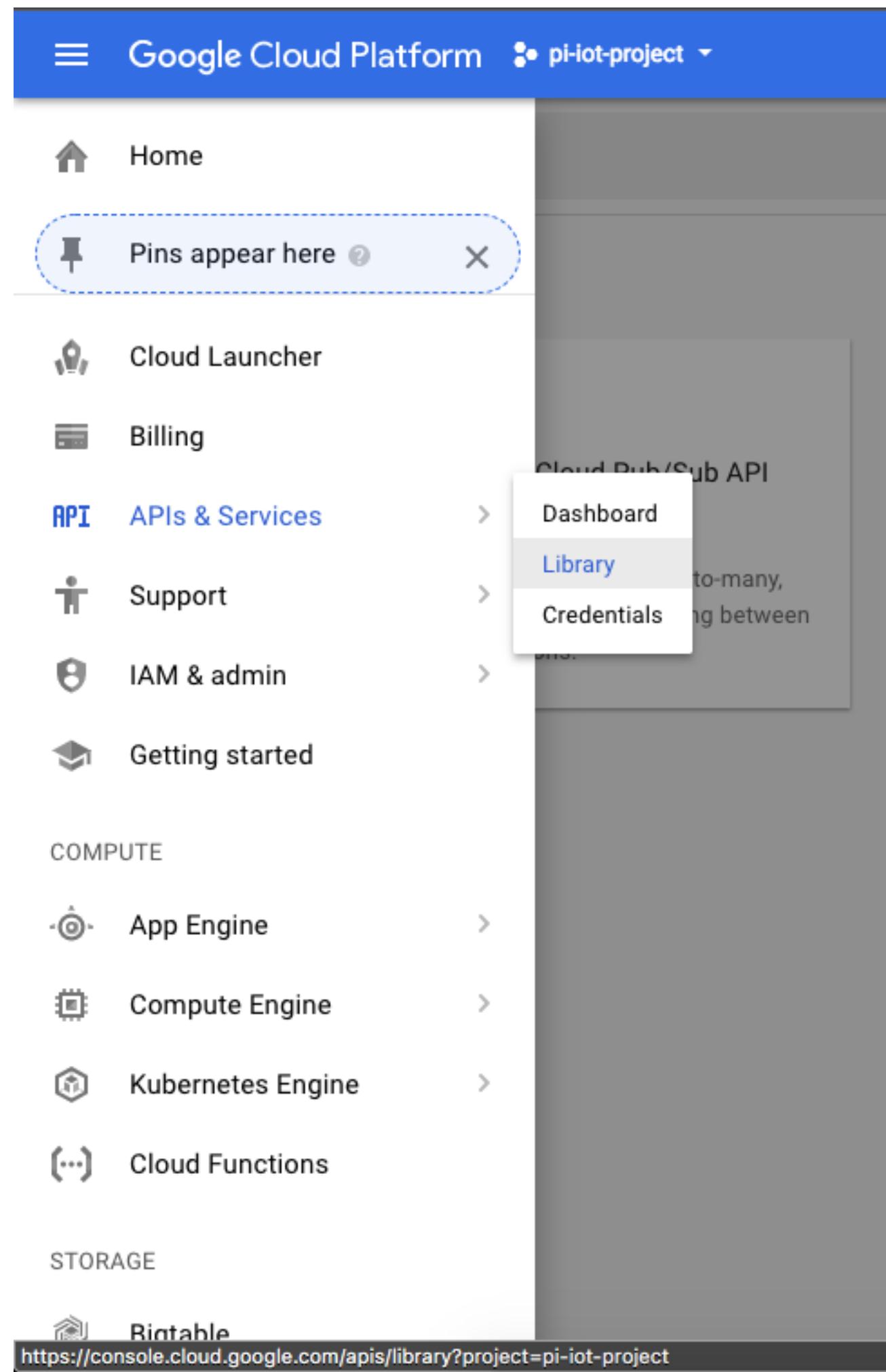
# Enabling APIs



# Enabling APIs



# Enabling APIs



This screenshot shows the "Google Cloud Pub/Sub API" page in the API Library. It features a circular icon with a network-like symbol. The title is "Google Cloud Pub/Sub API" and it is provided by "Google". A description states: "Provides reliable, many-to-many, asynchronous messaging between applications." Below the description are two buttons: "ENABLE" and "TRY THIS API".

This screenshot shows the "Google Realtime API" page in the API Library. It features a circular icon with a person icon. The title is "Google Realtime API" and it is provided by "Google". A description states: "The Google Realtime API makes it easy to build realtime collaboration into your application. Note..." Below the description is a single "ENABLE" button.

# Enabling device registry and devices

IoT Core **BETA**

## Device registries

A device registry allows you to group devices and set properties that they all share, such as connection protocol, data storage location, and Pub/Sub topics. To start connecting devices to Cloud IoT, first create a device registry to place them in. [Learn more](#)

**Create device registry**

Field	Value
Registry ID	Pi3-DHT11-Nodes
Cloud region	us-central1
Protocol	MQTT HTTP
Default telemetry topic	device-events
Default state topic	dht11

# Enabling device registry and devices

IoT Core **BETA**

## Device registries

A device registry allows you to group devices and set properties that they all share, such as connection protocol, data storage location, and Pub/Sub topics. To start connecting devices to Cloud IoT, first create a device registry to place them in. [Learn more](#)

**Create device registry**

IoT Core | [← Create device registry](#)

Set shared properties for devices in this registry.

**Registry ID** [?](#)  
Pi3-DHT11-Nodes

**Cloud region** [?](#)  
us-central1

**Protocol** [?](#)  
 MQTT  
 HTTP

**Pub/Sub topics**

**Default telemetry topic** [?](#)  
projects/pi-iot-project/topics/device-events

**Device state topic (Optional)** [?](#)  
projects/pi-iot-project/topics/dht11

[▼ Add CA certificate](#)

**Create** **Cancel**

# Public key

IoT Core    ← Device details    EDIT DEVICE    UPDATE CONFIG    BLOCK COMMUNICATION    DELETE

Device ID: Pi3-DHT11-Node  
Numeric ID: 2771385318201261    Registry: Pi3-DHT11-Nodes  
Device communication: Allowed

Details    Configuration & state history

Latest activity

Heartbeat (MQTT only)	—
Telemetry event received	—
Device state event received	—
Config sent	—
Config ACK (MQTT only)	—
Error	—

Device metadata  
None

Authentication

Add public key    Delete

No authentication keys have been added for this device.

The device won't be able to connect to Google Cloud Platform without a valid key.

```
openssl req -x509 -newkey rsa:2048 -keyout rsa_private.pem -nodes -out rsa_cert.pem -subj "/CN=unused"
```

# Public key

## Add authentication key

Specify a public key that will be used to authenticate this device. [Learn more](#)

**Input method**

- Enter manually
- Upload

**Public key format**

- RS256 (?)
- ES256 (?)
- RS256\_X509 (?)
- ES256\_X509 (?)

**Public key value**

```
-----BEGIN CERTIFICATE-----  
6IC/Derrh2DPiHoSbjc/dDhyqQDtIohWxQvV+LL8a1GfVXBW36SDVAT5y4mKm4C  
HidaNliePxsaerQ5xTStoLC+nCFauGxqic9vwZWFDaJhcCtNmXP5l7Z1i85d+gRt  
tMqW2TzIUm+BReAkxGZFxg32gcflumf5SblaEUs/hdMk/kBz7ULJ8N/dQguTchUd  
bFRHW9tMC8JMwya0Sb0CAwEAAaQME4wHQYDVR00BBYEFKQQ2Fj6ZFS0q7Ms280G  
VA955MKLMB8GA1UDIwQYMBaAFKQQ2Fj6ZFS0q7Ms200GVA955MKLMawGA1UdEwQF  
MAMBAf8wDQYJKoZIhvCNQELBQAQDggEBAf0RmIDvpzoHSFyMHnD78Xrm6amGxRNe  
LnRebkivFs+XZsgLj2hob151+QFyzp6+rNtMTfh+xTzx3bGIg1QIBIu3E96Kz+M  
b+sb/ADDD8wtmk7/cMrsftvCMiVtKhvKaSyAs1SizG5zVV9dwWEbj503yZ+tkS0  
a6YDwKfxpuNMk4prOwljJXJJP1W04d89NnSCH0orKqut6NpRsQbx+XoOyZi7KB  
nddi1T0muB460Ucklfzl1fy6YkjU/DUgeSdb+1MkpEnX1rQ+D8PH+71Ba3+isVT  
a79018bYa04Gu8e6bVeK08mbWH0gb5U/Pp07o6u1YajAqJKVU4dJscg=  
-----END CERTIFICATE-----
```

**Public key expiration date (Optional)**

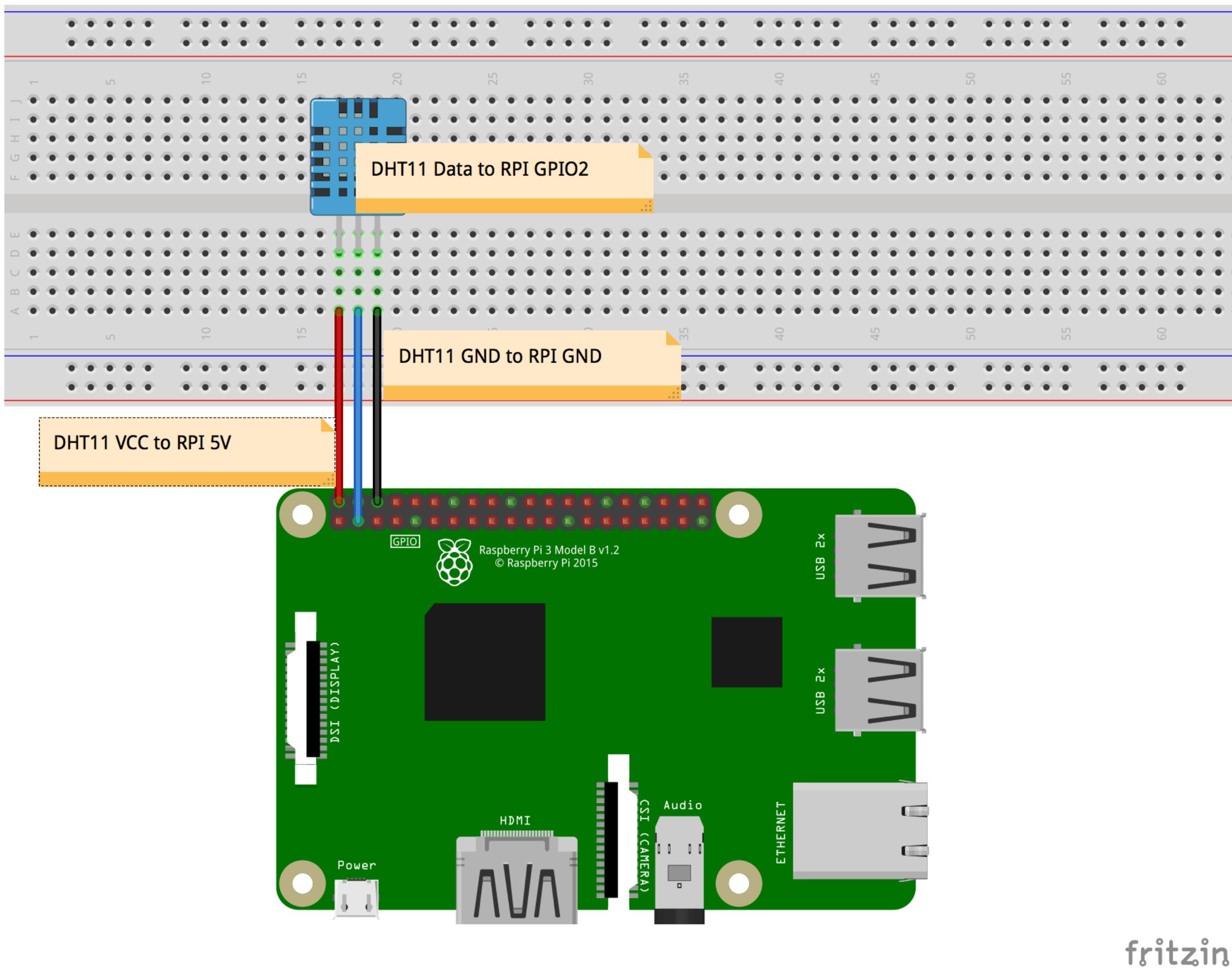
- Expires on:

2/17/19, 8:58 AM IST ▾

CANCEL ADD

```
openssl req -x509 -newkey rsa:2048 -keyout rsa_private.pem -nodes -out rsa_cert.pem -subj "/CN=unused"
```

# Setting up Raspberry Pi 3 with DHT11 node



# Setting up Node.js

1. Open a new Terminal and run the following commands:

```
$ sudo apt update  
$ sudo apt full-upgrade
```

2. This will upgrade all the packages that need upgrades. Next, we will install the latest version of Node.js. We will be using the Node 7.x version:

```
$ curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -  
$ sudo apt install nodejs
```

3. This will take a moment to install, and once your installation is done, you should be able to run the following commands to see the version of Node.js and npm:

```
$ node -v  
$ npm -v
```

# Developing the Node.js device app

1. From the Terminal, once you are inside the Google-IoT-Device folder, run the following command:

```
$ npm init -y
```

2. Next, we will install jsonwebtoken (<https://www.npmjs.com/package/jsonwebtoken>) and mqtt (<https://www.npmjs.com/package/mqtt>) from npm. Execute the following command:

```
$ npm install jsonwebtoken mqtt --save
```

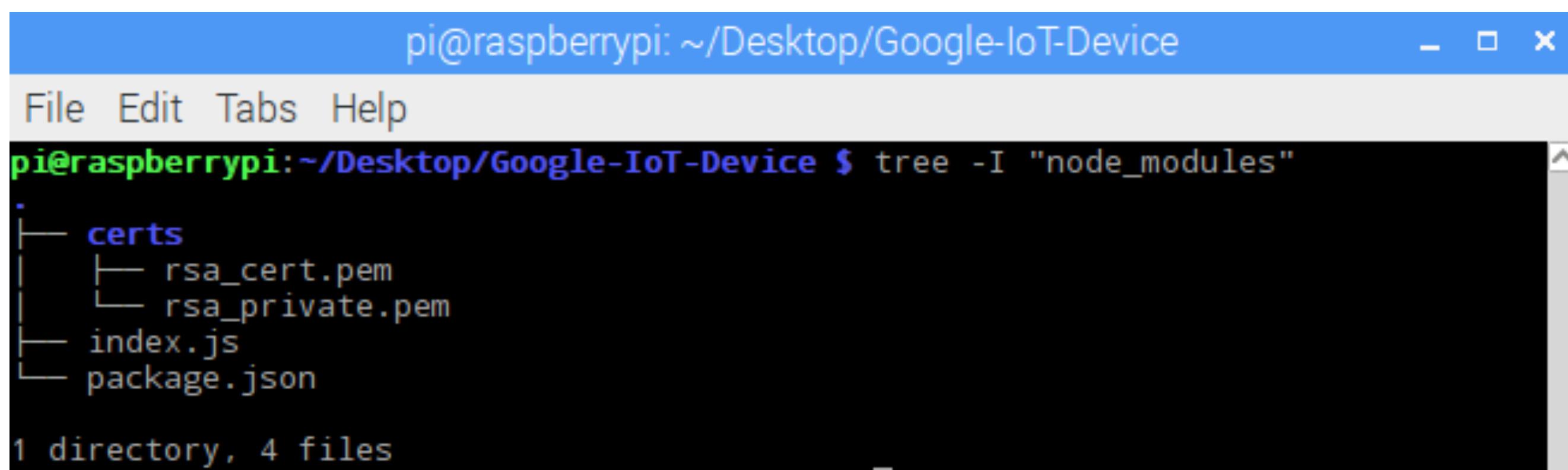
3. Next, we will install rpi-dht-sensor (<https://www.npmjs.com/package/rpi-dht-sensor>) from npm. This module will help in reading the DHT11 temperature and humidity values:

```
$ npm install rpi-dht-sensor --save
```

# package.json

```
{  
  "name": "Google-IoT-Device",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "jsonwebtoken": "^8.1.1",  
    "mqtt": "^2.15.3",  
    "rpi-dht-sensor": "^0.1.1"  
  }  
}
```

# Project Structure



A screenshot of a terminal window titled "pi@raspberrypi: ~/Desktop/Google-IoT-Device". The window shows a file tree command being run:

```
pi@raspberrypi:~/Desktop/Google-IoT-Device $ tree -I "node_modules"
.
+- certs
|   +- rsa_cert.pem
|   +- rsa_private.pem
+- index.js
+- package.json

1 directory, 4 files
```

The output shows a directory named "certs" containing two files: "rsa\_cert.pem" and "rsa\_private.pem". It also lists "index.js" and "package.json". A status message at the bottom indicates "1 directory, 4 files".

```
var fs = require('fs');
var jwt = require('jsonwebtoken');
var mqtt = require('mqtt');
var rpiDhtSensor = require('rpi-dht-sensor');

var dht = new rpiDhtSensor.DHT11(2); // `2` => GPIO2

var projectId = 'pi-iot-project';
var cloudRegion = 'us-central1';
var registryId = 'Pi3-DHT11-Nodes';
var deviceId = 'Pi3-DHT11-Node';

var mqttHost = 'mqtt.googleapis.com';
var mqttPort = 8883;
var privateKeyFile = '../certs/rsa_private.pem';
var algorithm = 'RS256';
var messageType = 'state'; // or event

var mqttClientId = 'projects/' + projectId + '/locations/' + cloudRegion + '/registries/' + registryId + '/devices/' + deviceId;
var mqttTopic = '/devices/' + deviceId + '/' + messageType;

var connectionArgs = {
  host: mqttHost,
  port: mqttPort,
  clientId: mqttClientId,
  username: 'unused',
  password: createJwt(projectId, privateKeyFile, algorithm),
```

```
var connectionArgs = {
  host: mqttHost,
  port: mqttPort,
  clientId: mqttClientId,
  username: 'unused',
  password: createJwt(projectId, privateKeyFile, algorithm),
  protocol: 'mqtts',
  secureProtocol: 'TLSv1_2_method'
};
```

```
console.log('connecting...');

var client = mqtt.connect(connectionArgs);
```

```
// Subscribe to the /devices/{device-id}/config topic to receive config updates.

client.subscribe('/devices/' + deviceId + '/config');
```

```
client.on('connect', function(success) {
  if (success) {
    console.log('Client connected...');
    sendData();
  } else {
    console.log('Client not connected...');
  }
});
```

```
client.on('close', function() {
  console.log('close');
});
```

```
client.on('error', function(err) {
  console.log('error', err);
});

client.on('message', function(topic, message, packet) {
  console.log(topic, 'message received: ', Buffer.from(message, 'base64').toString('ascii'));
});

function createJwt(projectId, privateKeyFile, algorithm) {
  var token = {
    'iat': parseInt(Date.now() / 1000),
    'exp': parseInt(Date.now() / 1000) + 86400 * 60, // 1 day
    'aud': projectId
  };
  var privateKey = fs.readFileSync(privateKeyFile);
  return jwt.sign(token, privateKey, {
    algorithm: algorithm
  });
}

function fetchData() {
  var readout = dht.read();
  var temp = readout.temperature.toFixed(2);
  var humd = readout.humidity.toFixed(2);

  return {
    'temp': temp,
    'humd': humd,
  }
}
```

```
algorithm: algorithm
});

}

function fetchData() {
  var readout = dht.read();
  var temp = readout.temperature.toFixed(2);
  var humd = readout.humidity.toFixed(2);

  return {
    'temp': temp,
    'humd': humd,
    'time': new Date().toISOString().slice(0, 19).replace('T', ' ')
  };
}

function sendData() {
  var payload = fetchData();

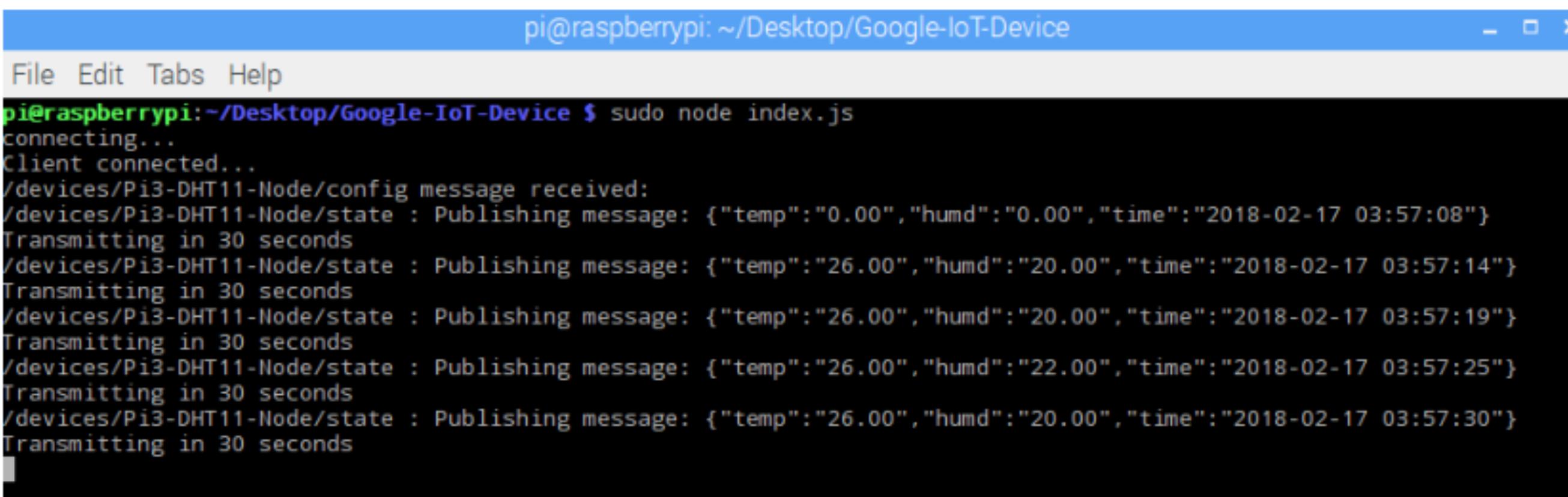
  payload = JSON.stringify(payload);
  console.log(mqttTopic, ': Publishing message:', payload);
  client.publish(mqttTopic, payload, { qos: 1 });

  console.log('Transmitting in 30 seconds');
  setTimeout(sendData, 30000);
}
```

# Running

```
$ sudo node index.js
```

And we should see something like this:



A screenshot of a terminal window titled "pi@raspberrypi: ~/Desktop/Google-IoT-Device". The window shows the command "sudo node index.js" being run and its output. The output indicates the script is connecting to a client and publishing messages every 30 seconds. Each message contains temperature, humidity, and timestamp data.

```
pi@raspberrypi:~/Desktop/Google-IoT-Device $ sudo node index.js
connecting...
Client connected...
/devices/Pi3-DHT11-Node/config message received:
/devices/Pi3-DHT11-Node/state : Publishing message: {"temp": "0.00", "humd": "0.00", "time": "2018-02-17 03:57:08"}
Transmitting in 30 seconds
/devices/Pi3-DHT11-Node/state : Publishing message: {"temp": "26.00", "humd": "20.00", "time": "2018-02-17 03:57:14"}
Transmitting in 30 seconds
/devices/Pi3-DHT11-Node/state : Publishing message: {"temp": "26.00", "humd": "20.00", "time": "2018-02-17 03:57:19"}
Transmitting in 30 seconds
/devices/Pi3-DHT11-Node/state : Publishing message: {"temp": "26.00", "humd": "22.00", "time": "2018-02-17 03:57:25"}
Transmitting in 30 seconds
/devices/Pi3-DHT11-Node/state : Publishing message: {"temp": "26.00", "humd": "20.00", "time": "2018-02-17 03:57:30"}
Transmitting in 30 seconds
```

# Console

Device ID: Pi3-DHT11-Node

Numeric ID: 2771385318201261 Registry: Pi3-DHT11-Nodes

Device communication:  Allowed

Details [Configuration & state history](#)

Configuration history  State history [Compare](#) 

Cloud update: February 17, 2018

Device State

Format  
 Base64  
 Text

{"temp": "26.00", "humd": "20.00", "time": "2018-02-17 03:57:41"}



Cloud update 9:27 AM

 STATE	Cloud Update: 9:27 AM	eyJ0ZW1wljoiMjYuMDAiLCJodW1kljoiMjAuMDAiLCJ0aW1lljoiMjAxOC0wMi0xNyAwMzo1NzozNiJ9
 STATE	Cloud Update: 9:27 AM	eyJ0ZW1wljoiMjYuMDAiLCJodW1kljoiMjAuMDAiLCJ0aW1lljoiMjAxOC0wMi0xNyAwMzo1NzozMCJ9
 STATE	Cloud Update: 9:27 AM	eyJ0ZW1wljoiMjYuMDAiLCJodW1kljoiMjIuMDAiLCJ0aW1lljoiMjAxOC0wMi0xNyAwMzo1NzoyNSJ9
 STATE	Cloud Update: 9:27 AM	eyJ0ZW1wljoiMjYuMDAiLCJodW1kljoiMjAuMDAiLCJ0aW1lljoiMjAxOC0wMi0xNyAwMzo1NzoxOSJ9
 STATE	Cloud Update: 9:27 AM	eyJ0ZW1wljoiMjYuMDAiLCJodW1kljoiMjAuMDAiLCJ0aW1lljoiMjAxOC0wMi0xNyAwMzo1NzoxNCJ9
 STATE	Cloud Update: 9:27 AM	eyJ0ZW1wljoiMC4wMCIsImh1bWQiOiwLjAwliwidGltZSI6ljlwMTgtMDItMTcgMDM6NTc6MDgifQ==

# Lecture outcomes

- MQTT Protocol
- Cloud
  - AWS
  - Azure
  - Google

