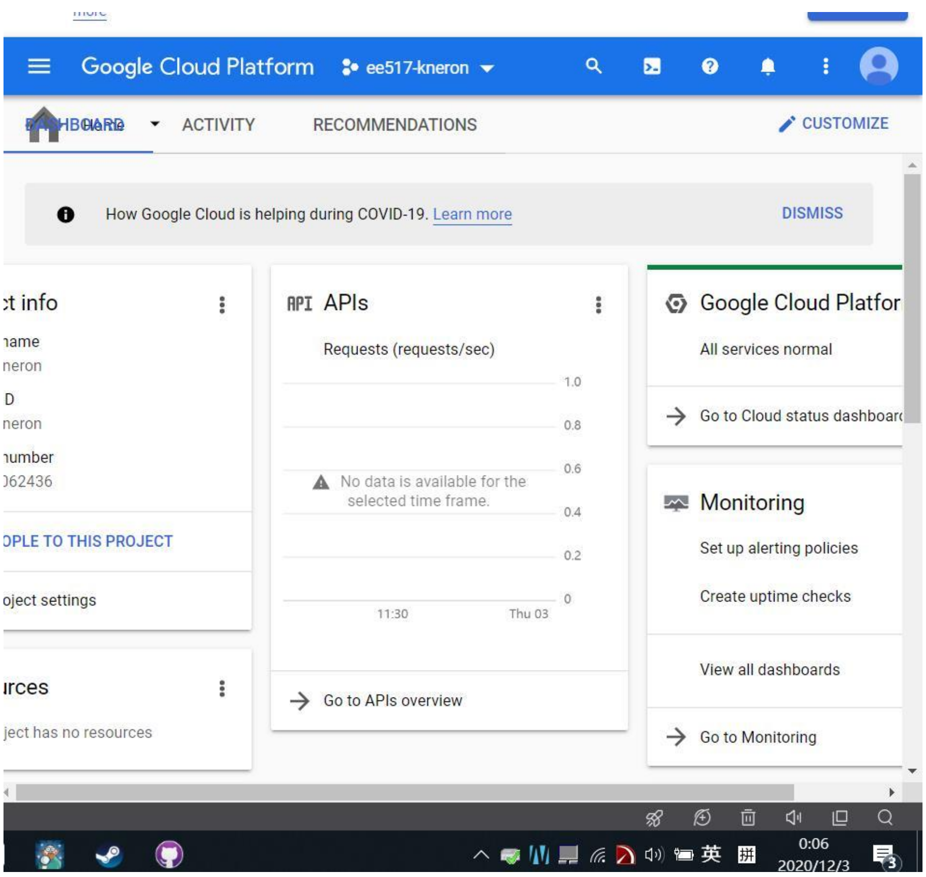
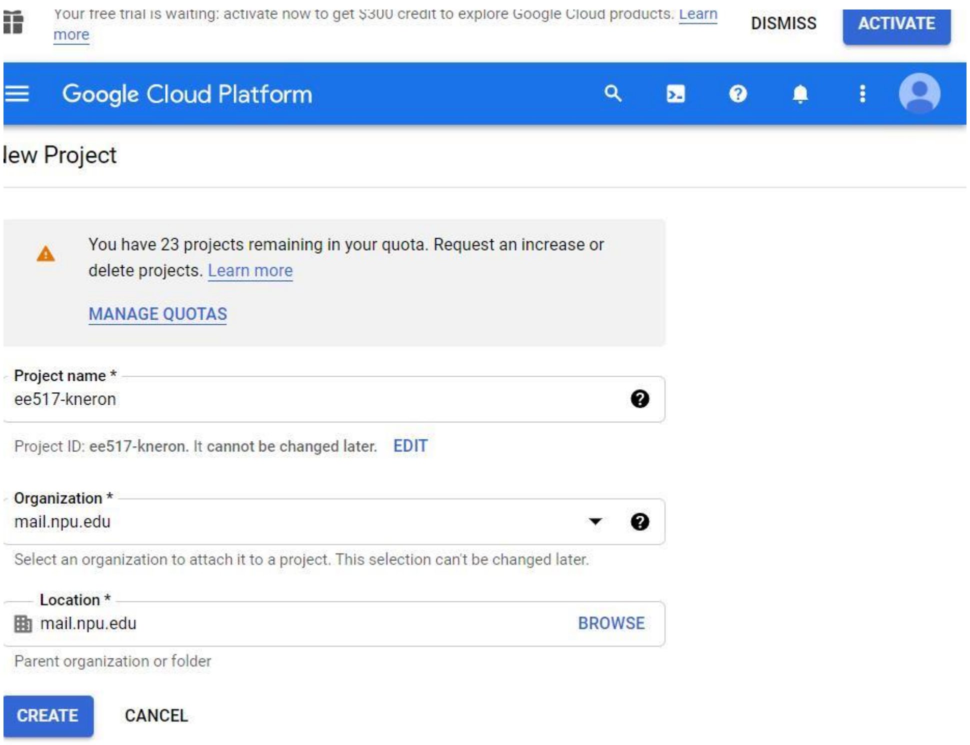
Open the Google Cloud Platform page



Create a new project as below


Your free trial is waiting: activate now to get $300 credit to explore Google Cloud products. Learn more

You have 23 projects remaining in your quota. Request an increase or delete projects. Learn more

MANAGE QUOTAS

Project name *
ee517-kneron

Project ID: ee517-kneron. It cannot be changed later. EDIT

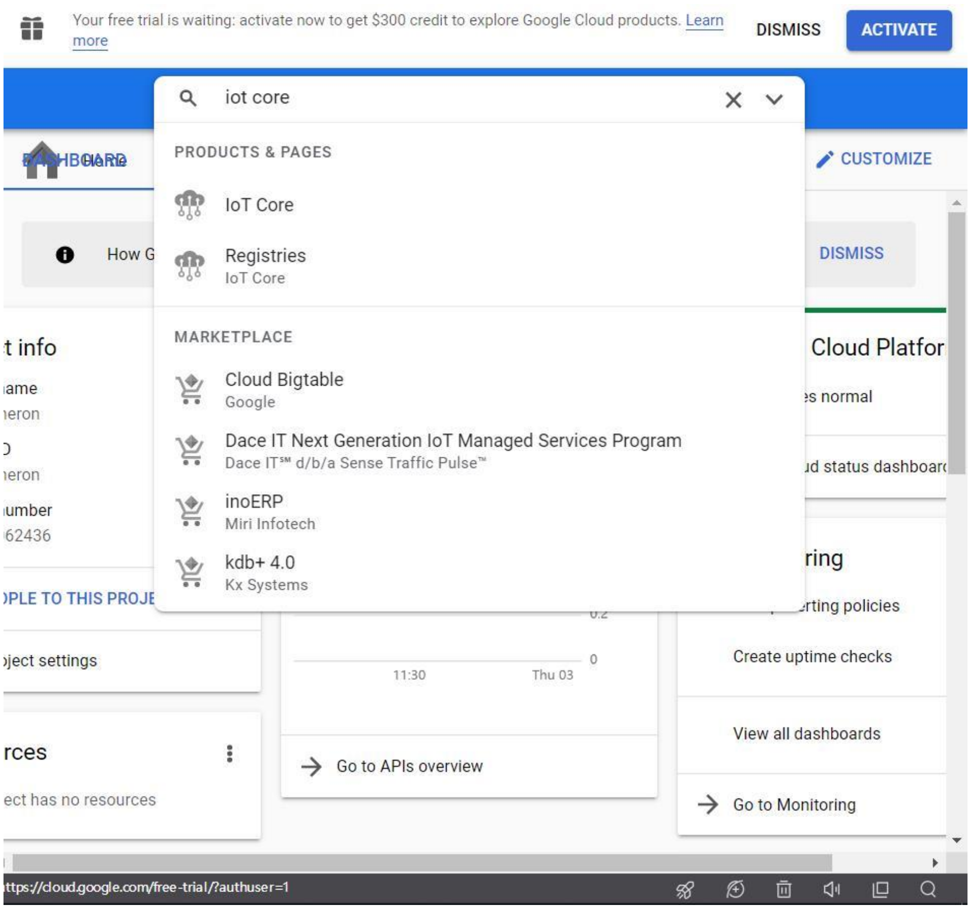
Organization *
mail.npu.edu

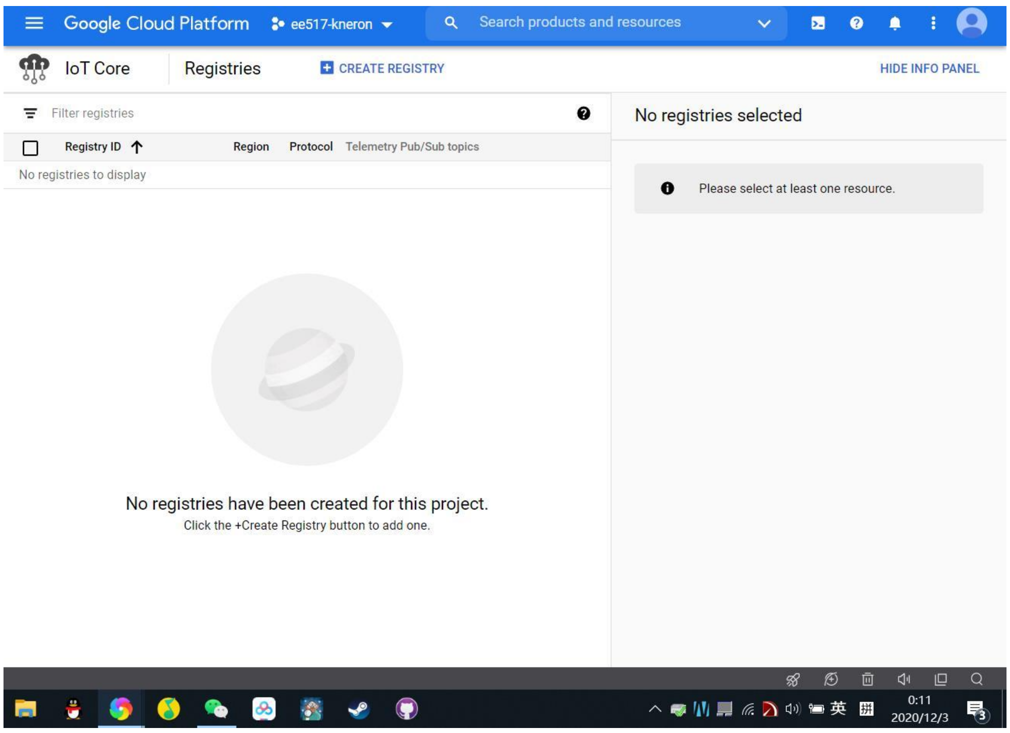
Select an organization to attach it to a project. This selection can't be changed later.

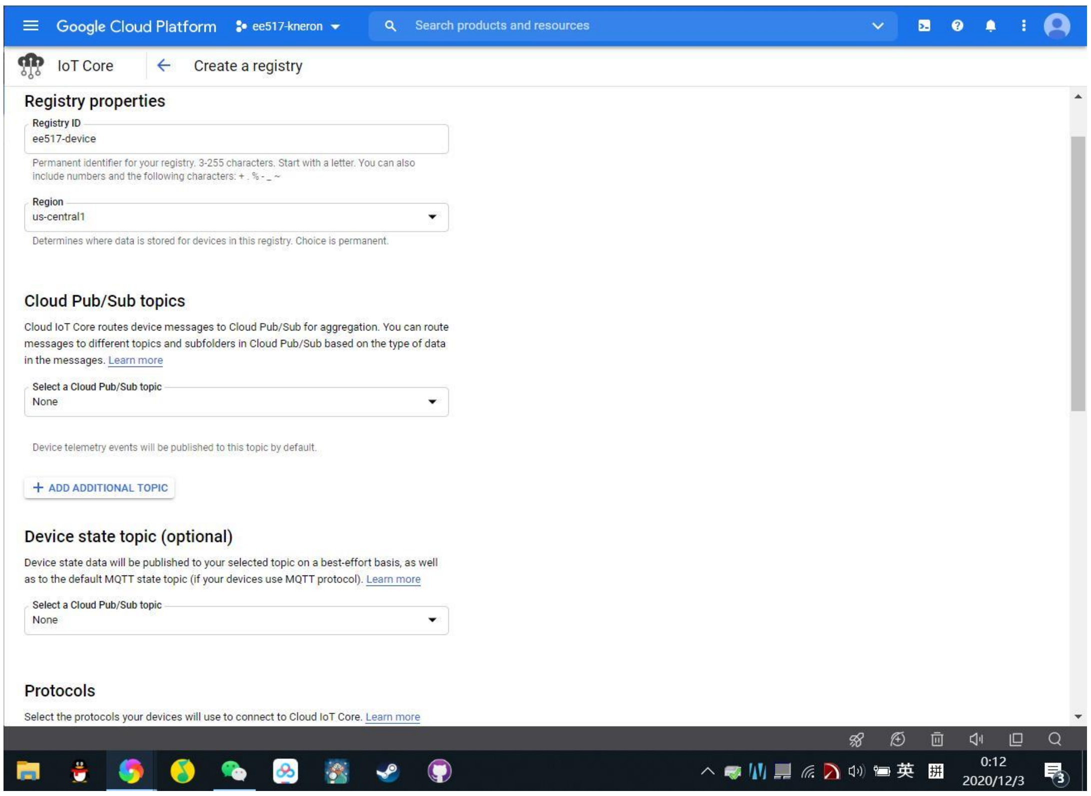
Location *
mail.npu.edu BROWSE

Parent organization or folder

Go to the IoT Core and create a new registry as below


Your free trial is waiting: activate now to get $300 credit to explore Google Cloud products. Learn more
DISMISS
ACTIVATE
iot core
PRODUCTS & PAGES
IoT Core
Registries
IoT Core
MARKETPLACE
Cloud Bigtable
Google
Dace IT Next Generation IoT Managed Services Program
Dace IT℠ d/b/a Sense Traffic Pulse™
inoERP
Miri Infotech
kdb+ 4.0
Kx Systems
CUSTOMIZE
DISMISS
Go to APIs overview
Create uptime checks
View all dashboards
Go to Monitoring
ttps://cloud.google.com/free-trial/?authuser=1





Registry properties

Registry ID: ee517-device

Permanent identifier for your registry. 3-255 characters. Start with a letter. You can also include numbers and the following characters: + . % - _ ~

Region: us-central1

Determines where data is stored for devices in this registry. Choice is permanent.

Cloud Pub/Sub topics

Cloud IoT Core routes device messages to Cloud Pub/Sub for aggregation. You can route messages to different topics and subfolders in Cloud Pub/Sub based on the type of data in the messages. Learn more

Select a Cloud Pub/Sub topic: None

Device telemetry events will be published to this topic by default.

+ ADD ADDITIONAL TOPIC

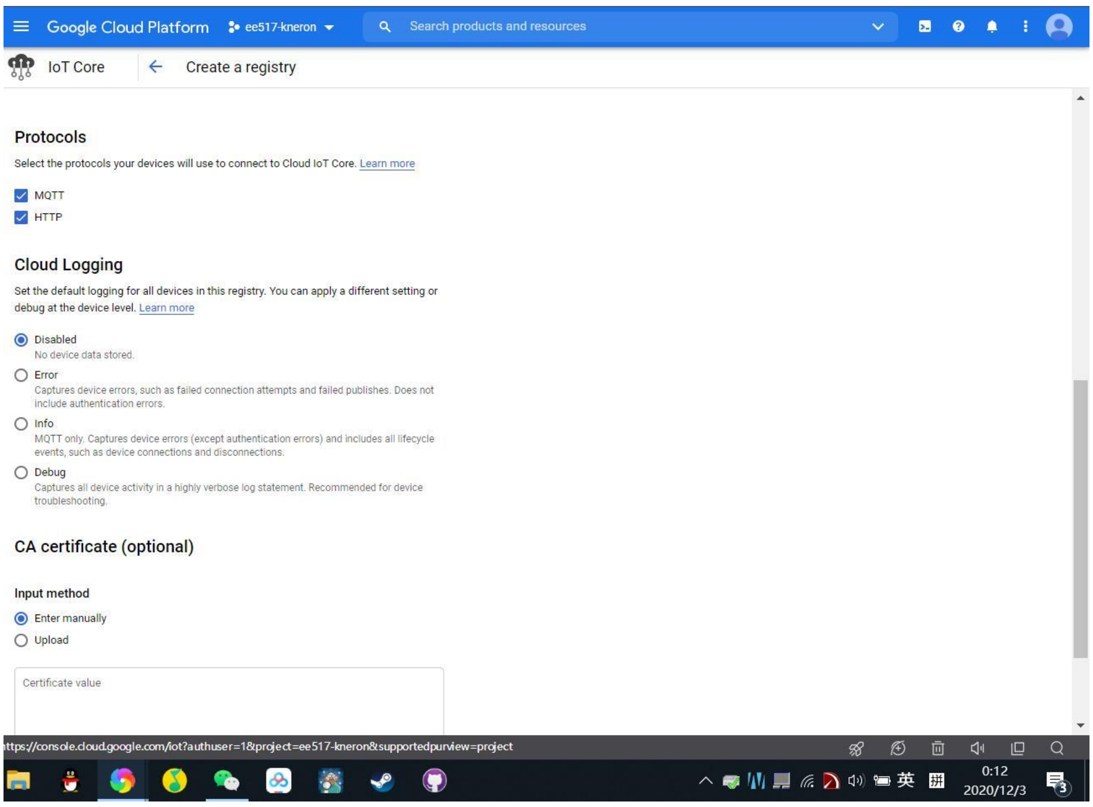
Device state topic (optional)

Device state data will be published to your selected topic on a best-effort basis, as well as to the default MQTT state topic (if your devices use MQTT protocol). Learn more

Select a Cloud Pub/Sub topic: None

Protocols

Select the protocols your devices will use to connect to Cloud IoT Core. Learn more



Protocols

Select the protocols your devices will use to connect to Cloud IoT Core. Learn more

[✓ MQTT]
[✓ HTTP]

Cloud Logging

Set the default logging for all devices in this registry. You can apply a different setting or debug at the device level. Learn more

[✓ Disabled]
No device data stored.

Error
Captures device errors, such as failed connection attempts and failed publishes. Does not include authentication errors.

Info
MQTT only. Captures device errors (except authentication errors) and includes all lifecycle events, such as device connections and disconnections.

Debug
Captures all device activity in a highly verbose log statement. Recommended for device troubleshooting.

CA certificate (optional)

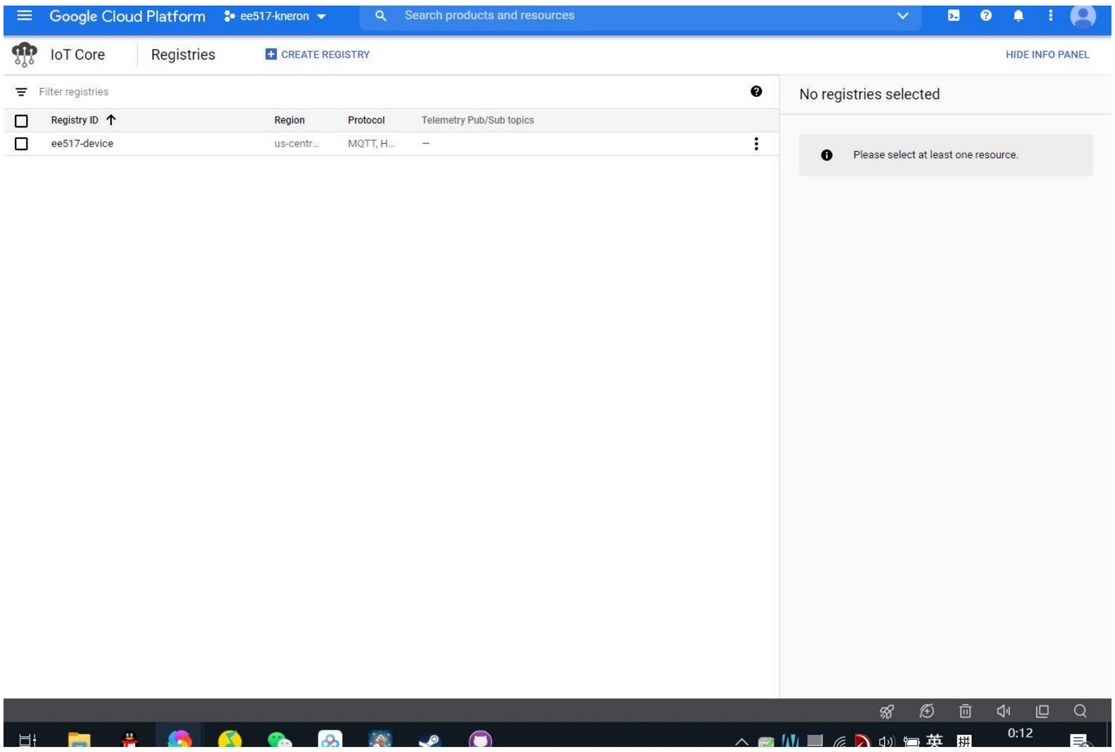
Input method

[✓ Enter manually]
Upload

Certificate value

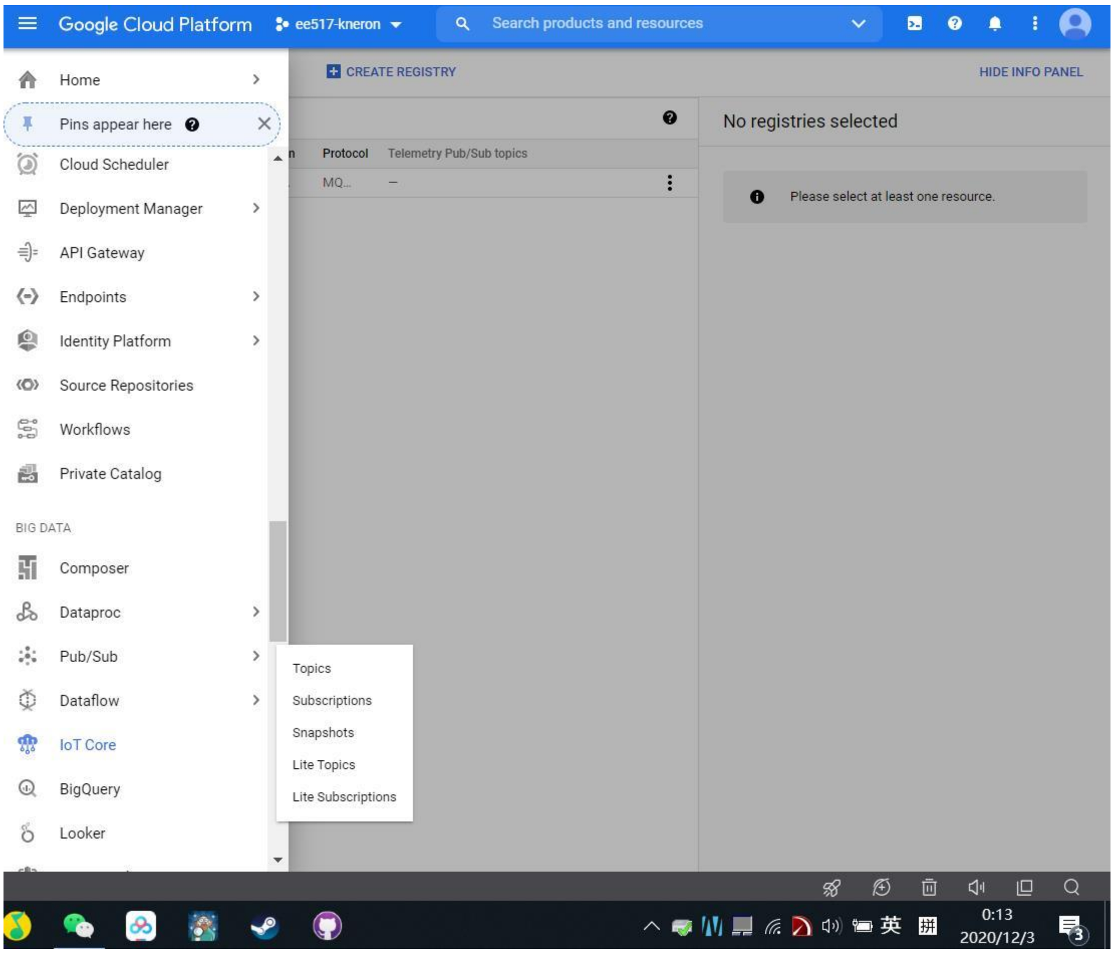
ttps://console.cloud.google.com/iot?authuser=1&project=ee517-kneron&supportedpurview=project

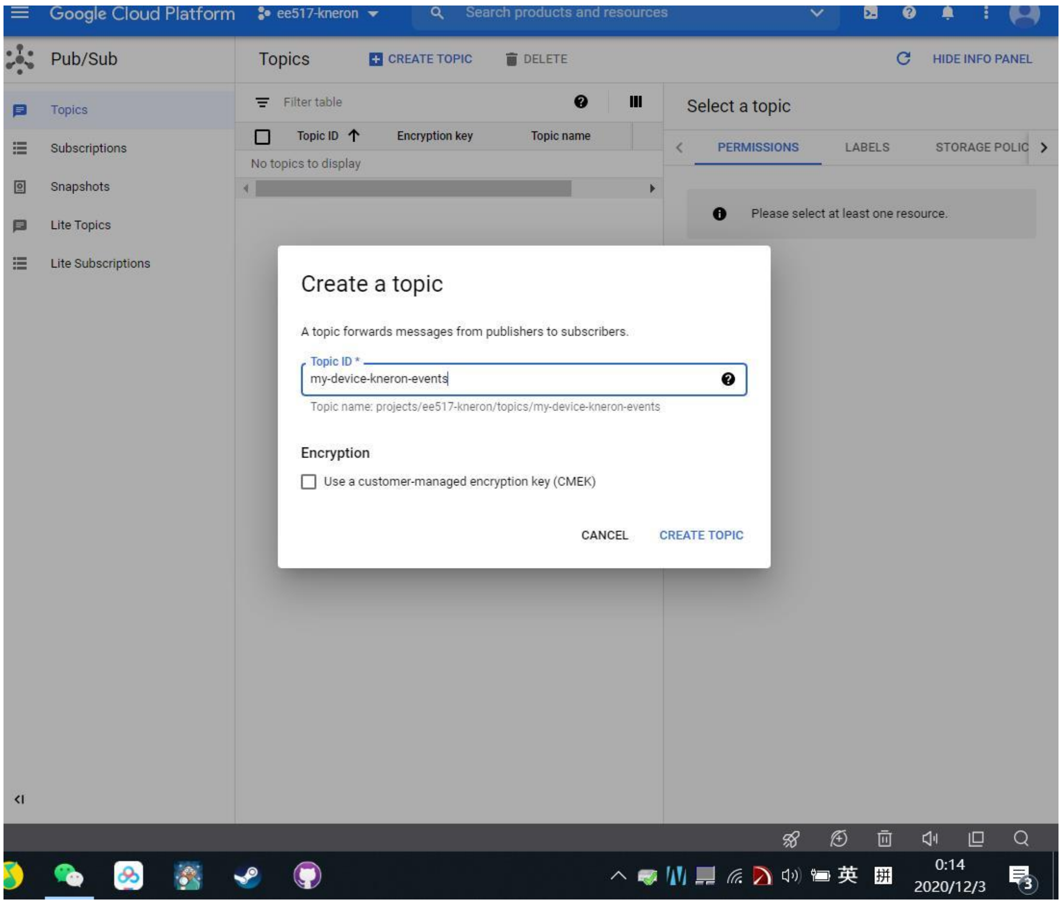
It will look like this after creating



Registry ID	Region	Protocol	Telemetry Pub/Sub topics
ee517-device	us-centr...	MQTT, H...	—

Go to the Pub/Sub , create a new Topic as below





Create a topic

A topic forwards messages from publishers to subscribers.

Topic ID * my-device-kneron-events

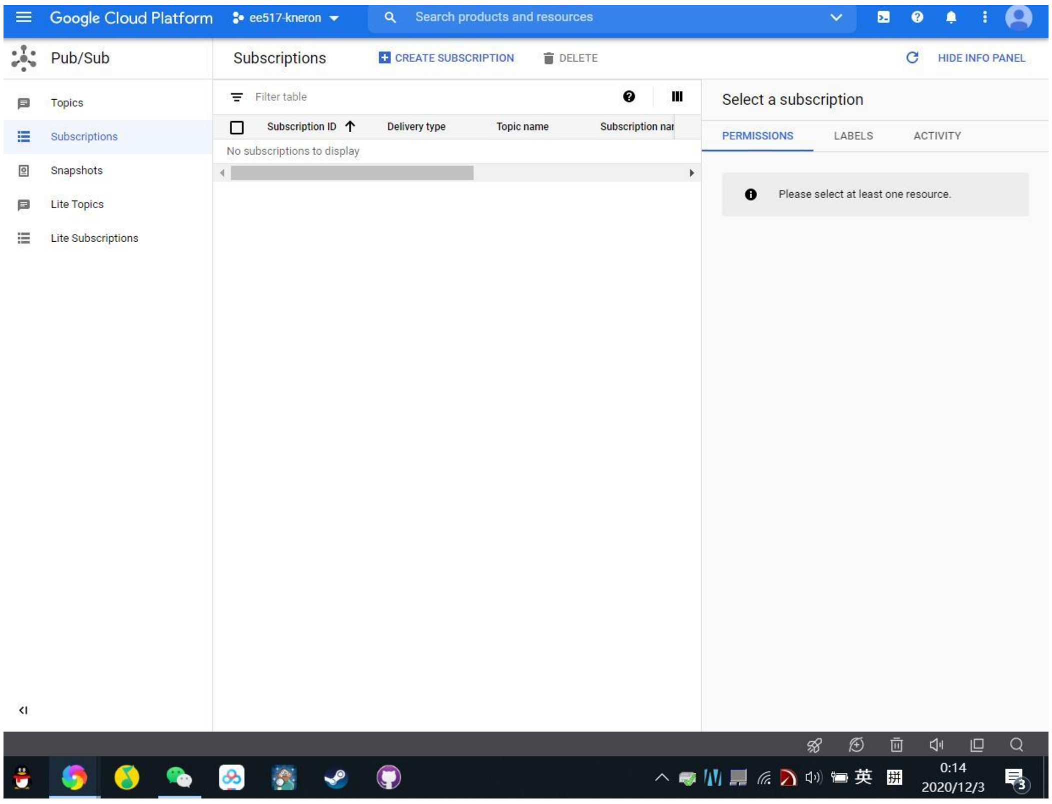
Topic name: projects/ee517-kneron/topics/my-device-kneron-events

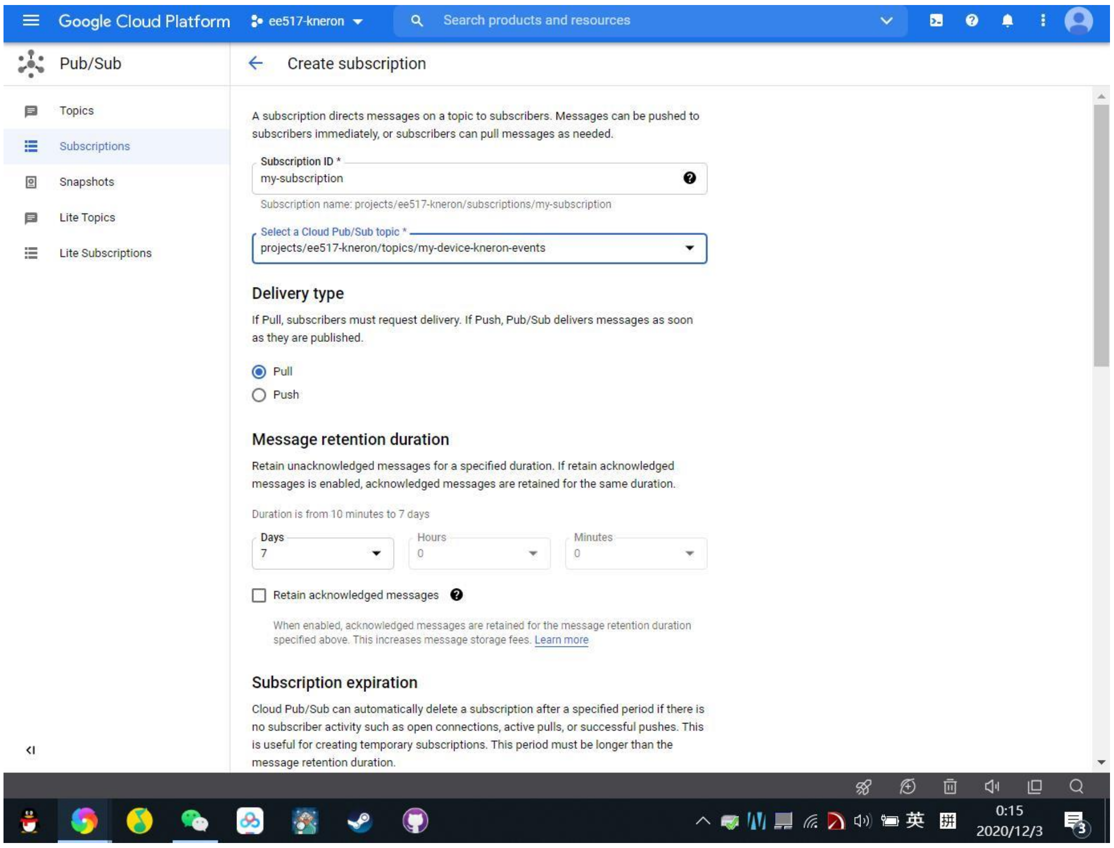
Encryption

☐ Use a customer-managed encryption key (CMEK)

CANCEL CREATE TOPIC

Create a Subscriptions as below




Google Cloud Platform · ee517-kneron · Search products and resources

Pub/Sub · Topics · Subscriptions · Snapshots · Lite Topics · Lite Subscriptions

Create subscription

A subscription directs messages on a topic to subscribers. Messages can be pushed to subscribers immediately, or subscribers can pull messages as needed.

Subscription ID *: my-subscription

Subscription name: projects/ee517-kneron/subscriptions/my-subscription

Select a Cloud Pub/Sub topic *: projects/ee517-kneron/topics/my-device-kneron-events

Delivery type

If Pull, subscribers must request delivery. If Push, Pub/Sub delivers messages as soon as they are published.

[✓ Pull] Push

Message retention duration

Retain unacknowledged messages for a specified duration. If retain acknowledged messages is enabled, acknowledged messages are retained for the same duration.

Duration is from 10 minutes to 7 days

Days: 7 · Hours: 0 · Minutes: 0

☐ Retain acknowledged messages

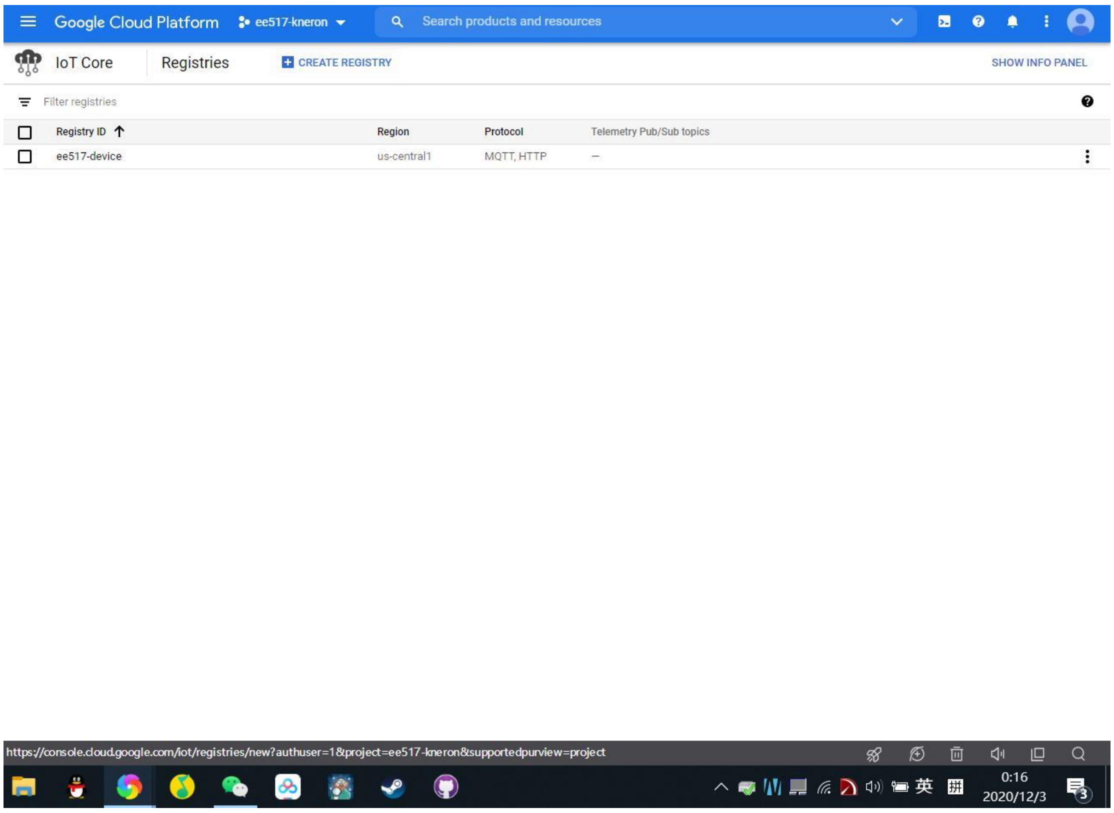
When enabled, acknowledged messages are retained for the message retention duration specified above. This increases message storage fees. Learn more

Subscription expiration

Cloud Pub/Sub can automatically delete a subscription after a specified period if there is no subscriber activity such as open connections, active pulls, or successful pushes. This is useful for creating temporary subscriptions. This period must be longer than the message retention duration.

0:15 2020/12/3

Go to the IoT Core to create a Device as below


Google Cloud Platform · ee517-kneron · Search products and resources

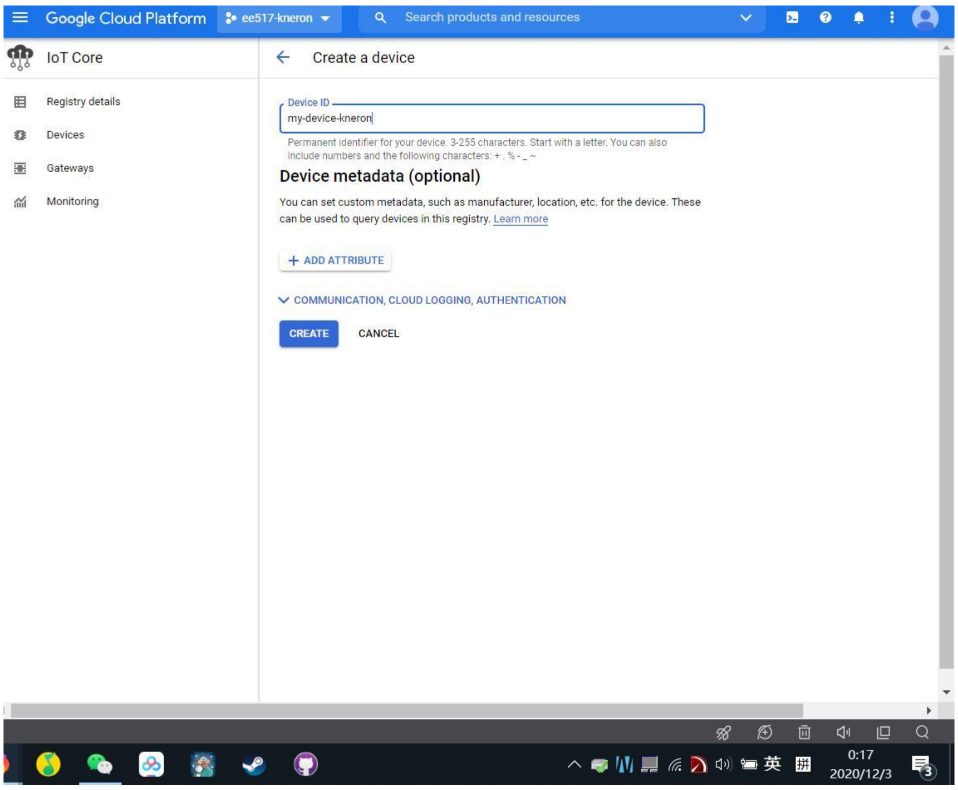
IoT Core · Registries · CREATE REGISTRY · SHOW INFO PANEL

Filter registries

Registry ID	Region	Protocol	Telemetry Pub/Sub topics
ee517-device	us-central1	MQTT, HTTP	—

https://console.cloud.google.com/iot/registries/new?authuser=1&project=ee517-kneron&supportedpurview=project

0:16 2020/12/3


Google Cloud Platform   ee517-kneron   Search products and resources

IoT Core

Registry details
Devices
Gateways
Monitoring

Create a device

Device ID: my-device-kneron

Permanent identifier for your device. 3-255 characters. Start with a letter. You can also include numbers and the following characters: + . % - _ ~

Device metadata (optional)

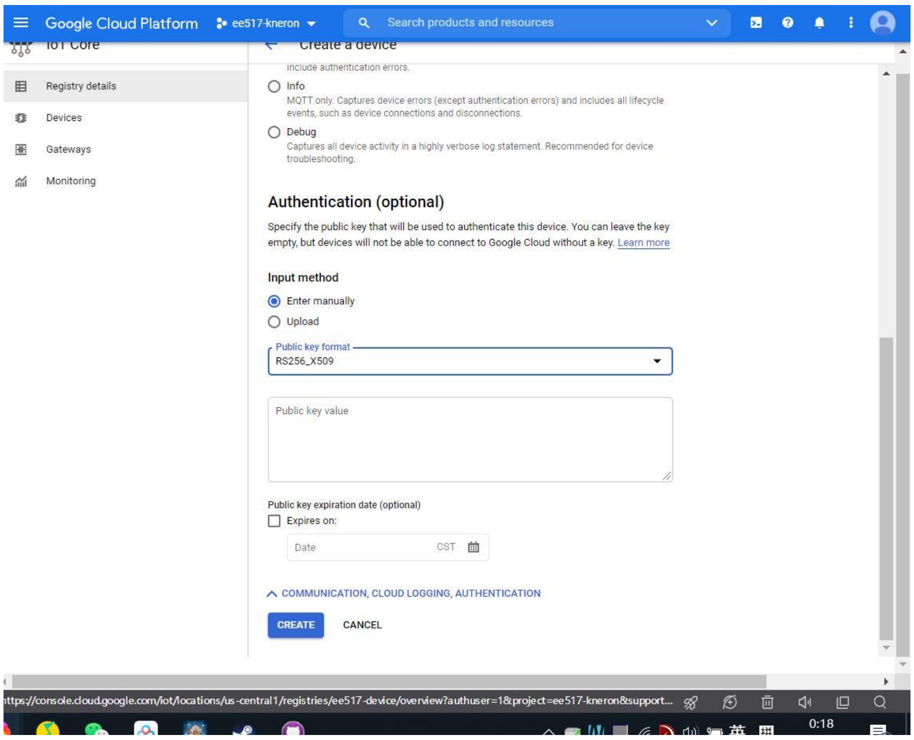
You can set custom metadata, such as manufacturer, location, etc. for the device. These can be used to query devices in this registry. Learn more

+ ADD ATTRIBUTE

COMMUNICATION, CLOUD LOGGING, AUTHENTICATION

CREATE   CANCEL

0:17 2020/12/3


Google Cloud Platform   ee517-kneron   Search products and resources

IoT Core   Create a device

Registry details
Devices
Gateways
Monitoring

include authentication errors.

Info
MQTT only. Captures device errors (except authentication errors) and includes all lifecycle events, such as device connections and disconnections.

Debug
Captures all device activity in a highly verbose log statement. Recommended for device troubleshooting.

Authentication (optional)

Specify the public key that will be used to authenticate this device. You can leave the key empty, but devices will not be able to connect to Google Cloud without a key. Learn more

Input method
[✓ Enter manually]
Upload

Public key format: RS256_X509

Public key value

Public key expiration date (optional)
Expires on:
Date   CST

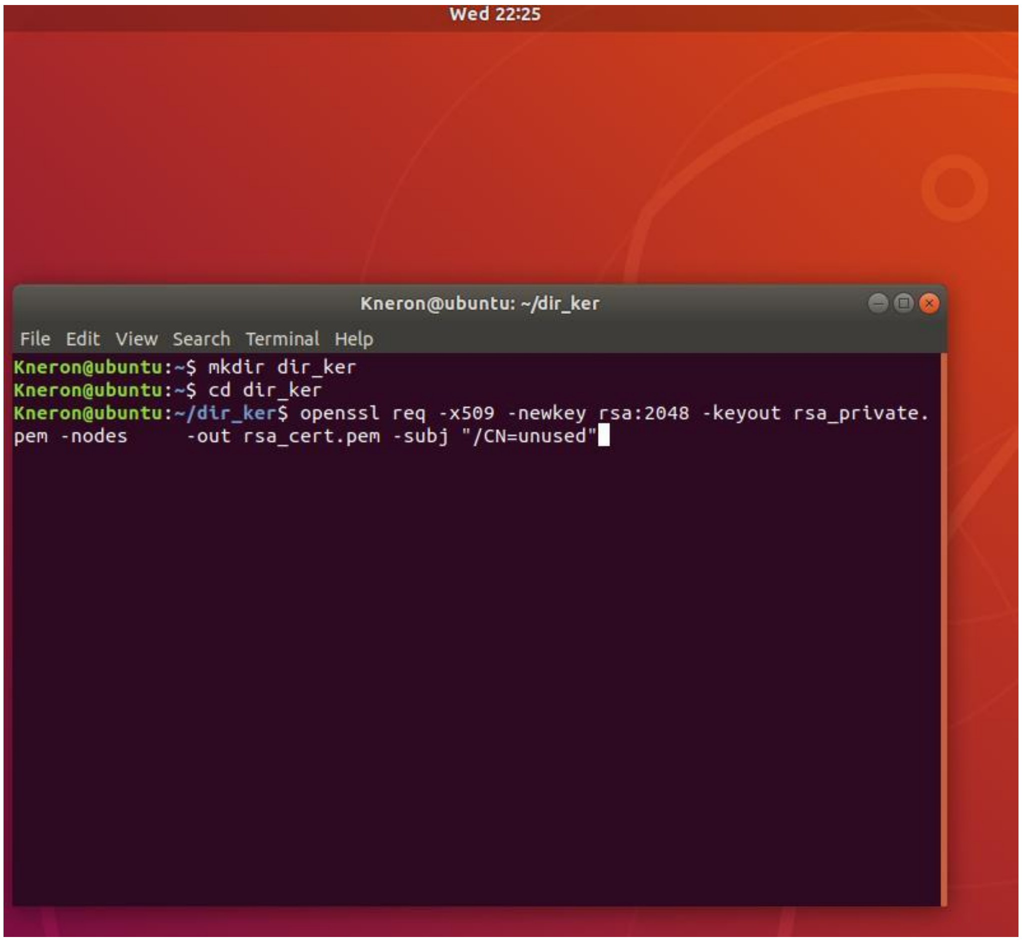
COMMUNICATION, CLOUD LOGGING, AUTHENTICATION

CREATE   CANCEL

ttps://console.cloud.google.com/iot/locations/us-central1/registries/ee517-device/overview?authuser=1&project=ee517-kneron&support...

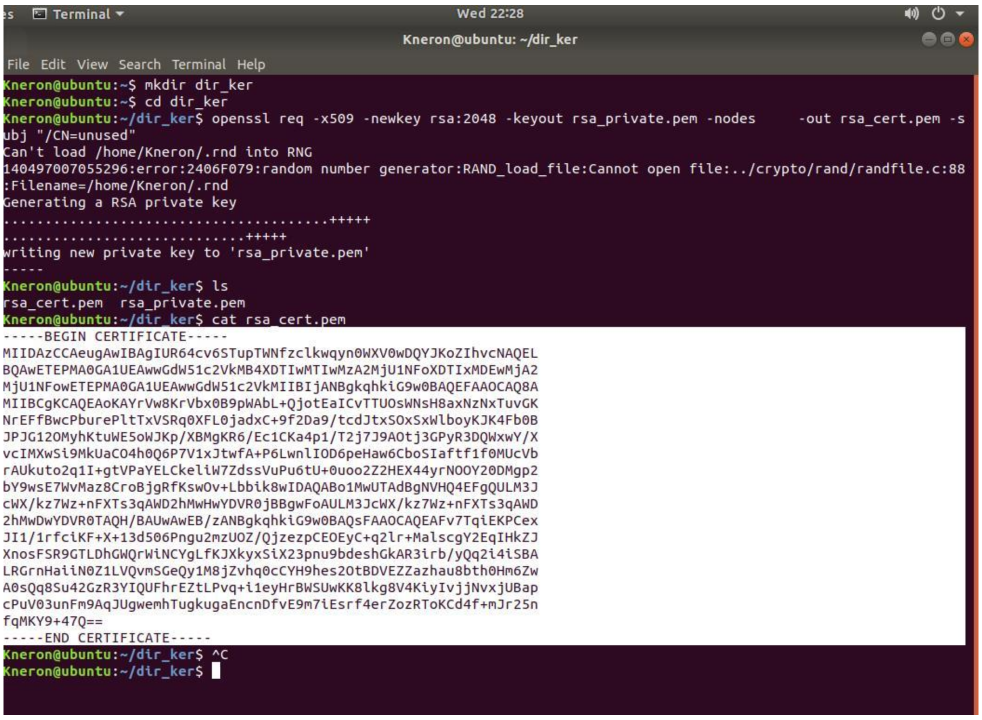
0:18

Open Ubuntu, make a new folder, and open terminal in the folder

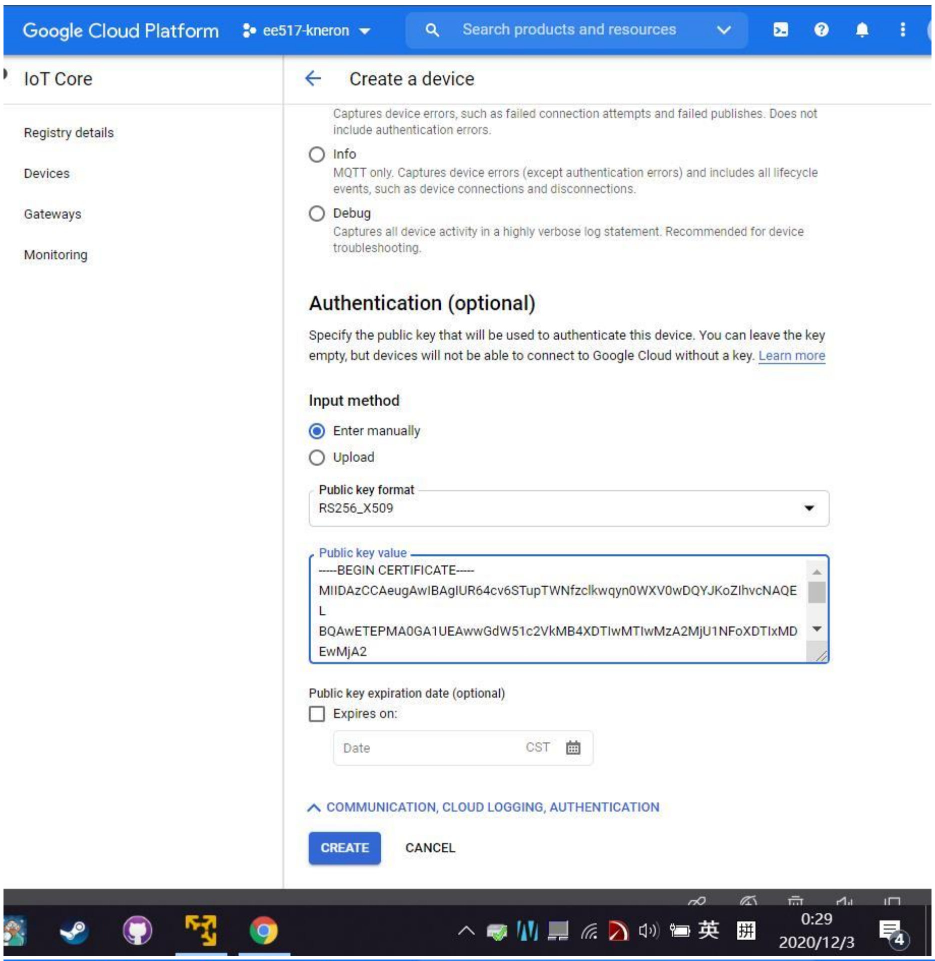


```
Kneron@ubuntu:~$ mkdir dir_ker
Kneron@ubuntu:~$ cd dir_ker
Kneron@ubuntu:~/dir_ker$ openssl req -x509 -newkey rsa:2048 -keyout rsa_private.
pem -nodes      -out rsa_cert.pem -subj "/CN=unused"
```

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



```
Kneron@ubuntu:~$ mkdir dir_ker
Kneron@ubuntu:~$ cd dir_ker
Kneron@ubuntu:~/dir_ker$ openssl req -x509 -newkey rsa:2048 -keyout rsa_private.pem -nodes     -out rsa_cert.pem -s
ubj "/CN=unused"
Can't load /home/Kneron/.rnd into RNG
140497007055296:error:2406F079:random number generator:RAND_load_file:Cannot open file:../crypto/rand/randfile.c:88
:Filename=/home/Kneron/.rnd
Generating a RSA private key
.......................................+++++
.............................+++++
writing new private key to 'rsa_private.pem'
-----
Kneron@ubuntu:~/dir_ker$ ls
rsa_cert.pem  rsa_private.pem
Kneron@ubuntu:~/dir_ker$ cat rsa_cert.pem
-----BEGIN CERTIFICATE-----
MIIDAzCCAeugAwIBAgIUR64cv6STupTWNfzclkwqyn0WXV0wDQYJKoZIhvcNAQEL
BQAwETEPMA0GA1UEAwwGdW51c2VkMB4XDTIwMTIwMzA2MjU1NFoXDTIxMDEwMjA2
MjU1NFowETEPMA0GA1UEAwwGdW51c2VkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A
MIIBCgKCAQEAoKAYrVw8KrVbx0B9pWAbL+QjotEaICvTTUOsWNsH8axNzNxTuvGK
NrEFfBwcPburePltTxVSRq0XFL0jadxC+9f2Da9/tcdJtxSOxSxWlboyKJK4Fb0B
JPJG12OMyhKtuWE5oWJKp/XBMgKR6/Ec1CKa4p1/T2j7J9AOtj3GPyR3DQWxwY/X
vcIMXwSi9MkUaCO4h0Q6P7V1xJtwfA+P6LwnlIOD6peHaw6CboSIaftf1f0MUcVb
rAUkuto2q1I+gtVPaYELCkeliW7ZdssVuPu6tU+0uoo2Z2HEX44yrNOOY20DMgp2
bY9wsE7WvMaz8CroBjgRfKswOv+Lbbik8wIDAQABo1MwUTAdBgNVHQ4EFgQULM3J
cWX/kz7Wz+nFXTs3qAWD2hMwHwYDVR0jBBgwFoAULM3JcWX/kz7Wz+nFXTs3qAWD
2hMwDwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAFv7TqiEKPCex
JI1/1rfciKF+X+13d506Pngu2mzUOZ/QjzezpCEOEyC+q2lr+MalscgY2EqIHkZJ
XnosFSR9GTLDhGWQrWiNCYgLfKJXkyxSiX23pnu9bdeshGkAR3irb/yQq2i4iSBA
LRGrnHaiiN0Z1LVQvmSGeQy1M8jZvhq0cCYH9hes2OtBDVEZZazhau8bth0Hm6Zw
A0sQq8Su42GzR3YIQUFhrEZtLPvq+i1eyHrBWSUwKK8lkg8V4KiyIvjjNvxjUBap
cPuV03unFm9AqJUgwemhTugkugaEncnDfvE9m7iEsrf4erZozRToKCd4f+mJr25n
fqMKY9+47Q==
-----END CERTIFICATE-----
Kneron@ubuntu:~/dir_ker$ ^C
Kneron@ubuntu:~/dir_ker$
```


Google Cloud Platform ee517-kneron Search products and resources

IoT Core

Registry details
Devices
Gateways
Monitoring

Create a device

Captures device errors, such as failed connection attempts and failed publishes. Does not include authentication errors.

Info
MQTT only. Captures device errors (except authentication errors) and includes all lifecycle events, such as device connections and disconnections.

Debug
Captures all device activity in a highly verbose log statement. Recommended for device troubleshooting.

Authentication (optional)

Specify the public key that will be used to authenticate this device. You can leave the key empty, but devices will not be able to connect to Google Cloud without a key. Learn more

Input method
[✓ Enter manually]
Upload

Public key format
RS256_X509

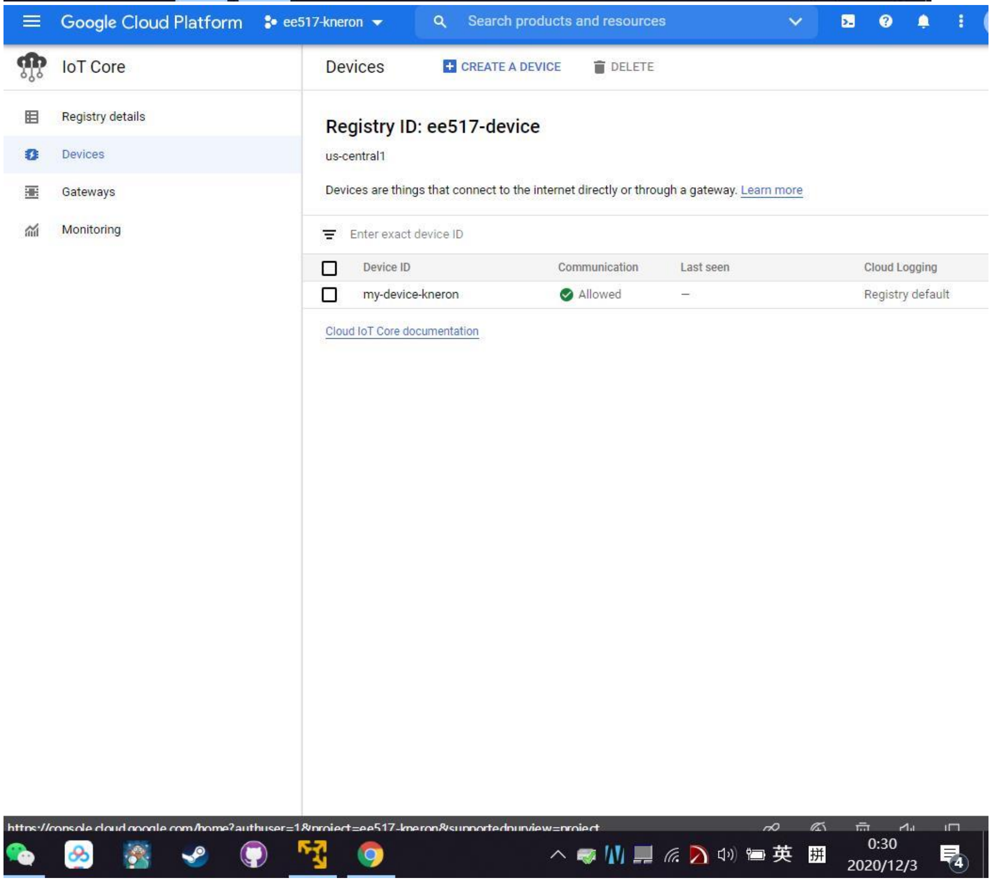
Public key value
```
-----BEGIN CERTIFICATE-----
MIIDAzCCAeugAwIBAgIUR64cv6STupTWNfzclkwqyn0WXV0wDQYJKoZIhvcNAQE
L
BQAwETEPMA0GA1UEAwwGdW51c2VkMB4XDTIwMTIwMzA2MjU1NFoXDTIxMD
EwMjA2
```

Public key expiration date (optional)
Expires on:
Date CST

COMMUNICATION, CLOUD LOGGING, AUTHENTICATION

CREATE CANCEL

0:29
2020/12/3


Google Cloud Platform ee517-kneron Search products and resources

IoT Core

Registry details
Devices
Gateways
Monitoring

Devices CREATE A DEVICE DELETE

Registry ID: ee517-device

us-central1

Devices are things that connect to the internet directly or through a gateway. Learn more

Enter exact device ID

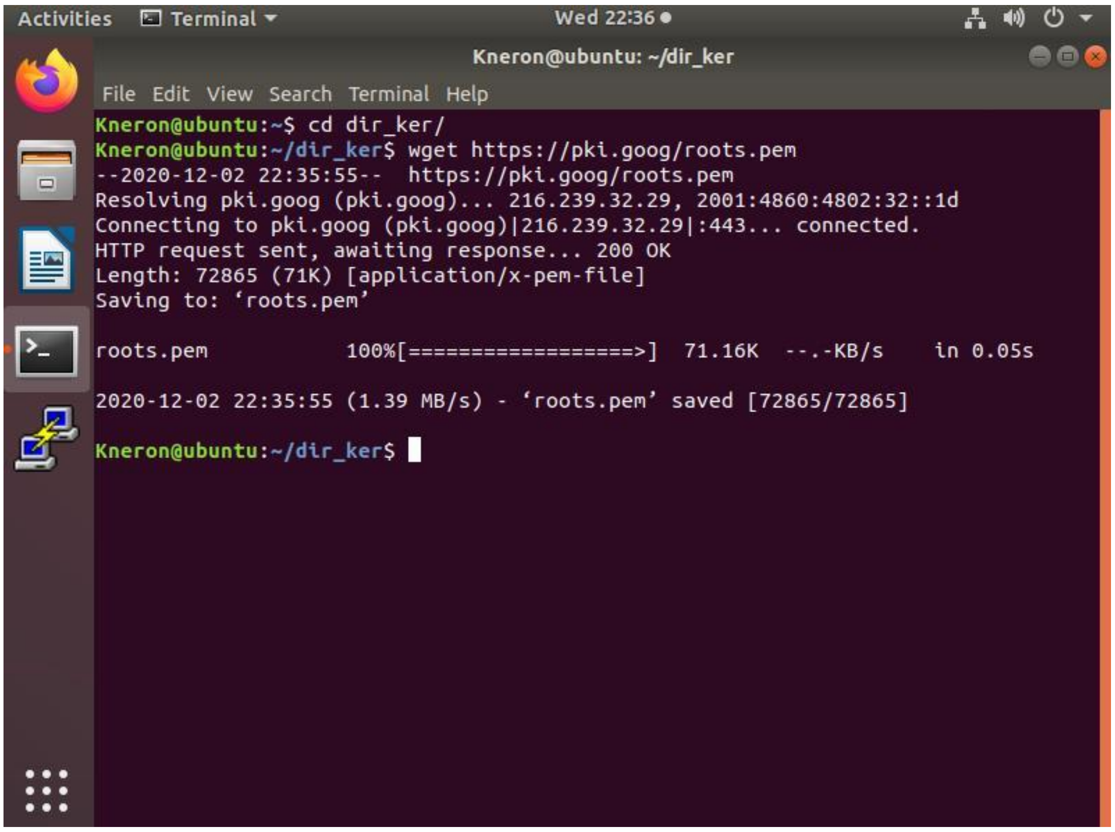
Device ID	Communication	Last seen	Cloud Logging
my-device-kneron	✓ Allowed	–	Registry default

Cloud IoT Core documentation

https://console.cloud.google.com/home?authuser=1&project=ee517-kneron&supportedpurview=project

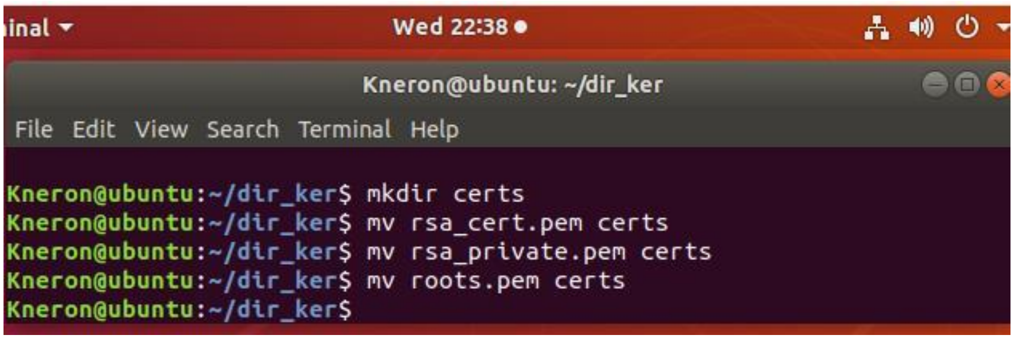
0:30
2020/12/3

Enter： wget https://pki.goog/roots.pem



```
Kneron@ubuntu:~$ cd dir_ker/
Kneron@ubuntu:~/dir_ker$ wget https://pki.goog/roots.pem
--2020-12-02 22:35:55--  https://pki.goog/roots.pem
Resolving pki.goog (pki.goog)... 216.239.32.29, 2001:4860:4802:32::1d
Connecting to pki.goog (pki.goog)|216.239.32.29|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 72865 (71K) [application/x-pem-file]
Saving to: ‘roots.pem’

roots.pem           100%[===================>]  71.16K  --.-KB/s    in 0.05s

2020-12-02 22:35:55 (1.39 MB/s) - ‘roots.pem’ saved [72865/72865]

Kneron@ubuntu:~/dir_ker$ 
```

Create a new folder to store as below

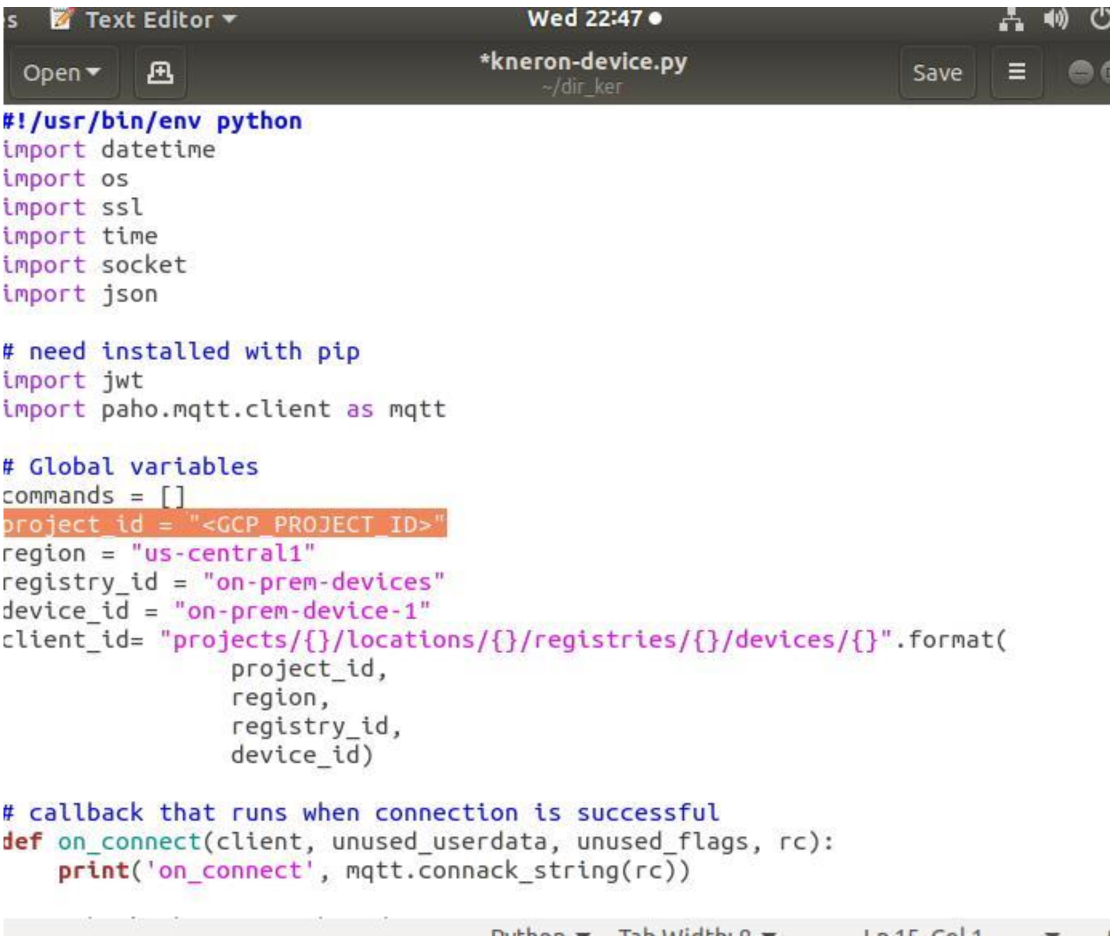


```
Kneron@ubuntu:~/dir_ker$ mkdir certs
Kneron@ubuntu:~/dir_ker$ mv rsa_cert.pem certs
Kneron@ubuntu:~/dir_ker$ mv rsa_private.pem certs
Kneron@ubuntu:~/dir_ker$ mv roots.pem certs
Kneron@ubuntu:~/dir_ker$
```

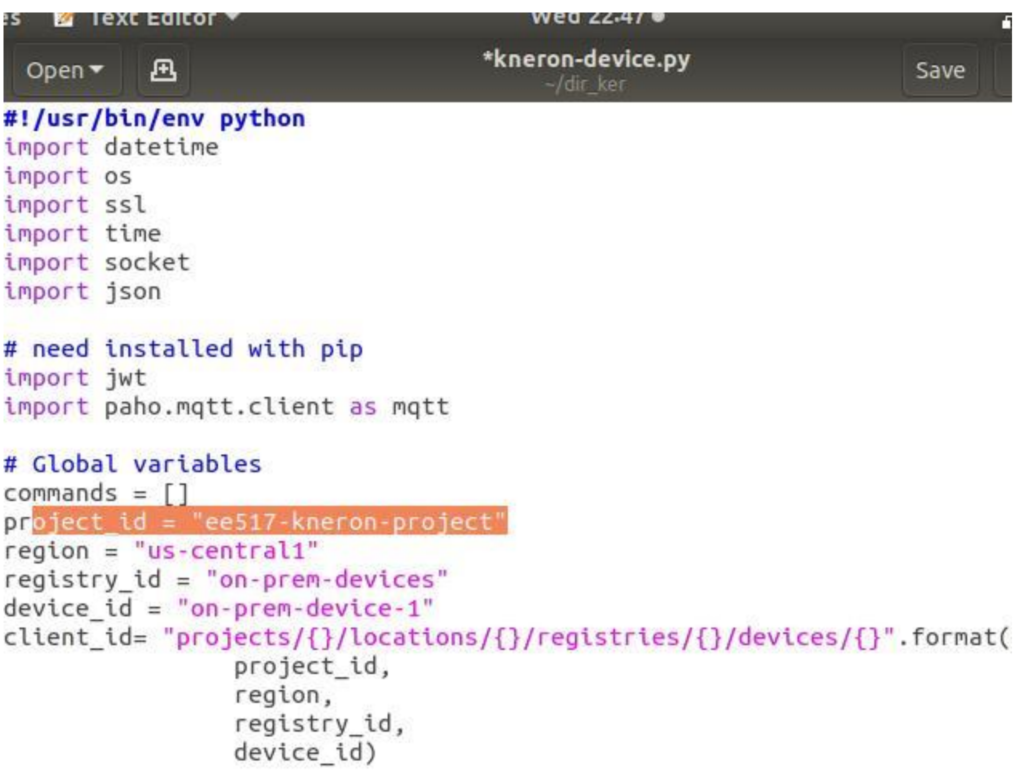
Crate a new python file, and enter:

```
#!/usr/bin/env python
import datetime
import os
import ssl
import time
import socket
import json

# need installed with pip
import jwt
import paho.mqtt.client as mqtt

# Global variables
commands = []
project_id = "<GCP_PROJECT_ID>"
region = "us-central1"
registry_id = "on-prem-devices"
device_id = "on-prem-device-1"
client_id=
"projects/{}/locations/{}/registries/{}/devices/{}".format(
               project_id,
               region,
               registry_id,
               device_id)

# callback that runs when connection is successful
def on_connect(client, unused_userdata, unused_flags, rc):
    print('on_connect', mqtt.connack_string(rc))

# callback that runs when disconnection is successful
def on_disconnect(unused_client, unused_userdata, rc):
    print('on_disconnect', error_str(rc))

# callback that runs when data is published
def on_publish(unused_client, unused_userdata, unused_mid):
    print('on_publish')

# callback that runs when a message is recieved from a subscription
def on_message(unused_client, unused_userdata, message):
    global commands
    payload = str(message.payload.decode('utf-8'))
    print('Received message \'{}\' on topic \'{}\' with Qos
{}'.format(payload, message.topic, str(message.qos)))
    # check if message is a command or state
    if "commands" in message.topic:
        commands.append(payload)

# creates jwt token to authenticate
def create_jwt(project_id, algorithm):
    token = {
            'iat': datetime.datetime.utcnow(),
            'exp': datetime.datetime.utcnow() +
datetime.timedelta(minutes=60),
            'aud': project_id
    }
    private_key_file = "./key.pem"

    # Read the private key file.
    with open(private_key_file, 'r') as f:
        private_key = f.read()

    print('Creating JWT using {} from private key file {}'.format(
            algorithm, private_key_file))

    return jwt.encode(token, private_key, algorithm=algorithm)

# initialises the MQTT client and connects
def get_client(project_id, client_id):
    client = mqtt.Client(client_id=client_id)
    client.username_pw_set(
            username='unused',
            password=create_jwt(project_id, "RS256"))
    client.tls_set(ca_certs="./roots.pem",
tls_version=ssl.PROTOCOL_TLSv1_2)

    client.on_connect = on_connect
    client.on_publish = on_publish
    client.on_disconnect = on_disconnect
    client.on_message = on_message

    # Connect to the Google MQTT bridge.
    client.connect("mqtt.googleapis.com", 8883)

    mqtt_config_topic = '/devices/{}/config'.format(device_id)
    client.subscribe(mqtt_config_topic, qos=1)
    mqtt_command_topic = '/devices/{}/commands/#'.format(device_id)
    client.subscribe(mqtt_command_topic, qos=1)

    return client

def main():
    global project_id
    global client_id
    global commands
    client = get_client(project_id, client_id)
    client.loop_start()

    print("starting loop")
    while True:
        # check if we have recieved any commands
        if len(commands) > 0:
            command = commands.pop(0)
            # parse the command and get the dns name
            loaded_json = json.loads(command)

            # do a lookup on the dns name
            addr = socket.gethostbyname(loaded_json["dnsName"])

            # publish the results back to MQTT
            payload = {"address": addr}
            mqtt_topic = '/devices/{}/events'.format(device_id)
            print('Publishing to {}'.format(mqtt_topic))
            infot = client.publish(mqtt_topic, json.dumps(payload),
qos=0, retain=False)
            infot.wait_for_publish()
        # we sleep each loop to keep within the MQTT limits
        time.sleep(1)

if __name__ == '__main__':
    main()
```

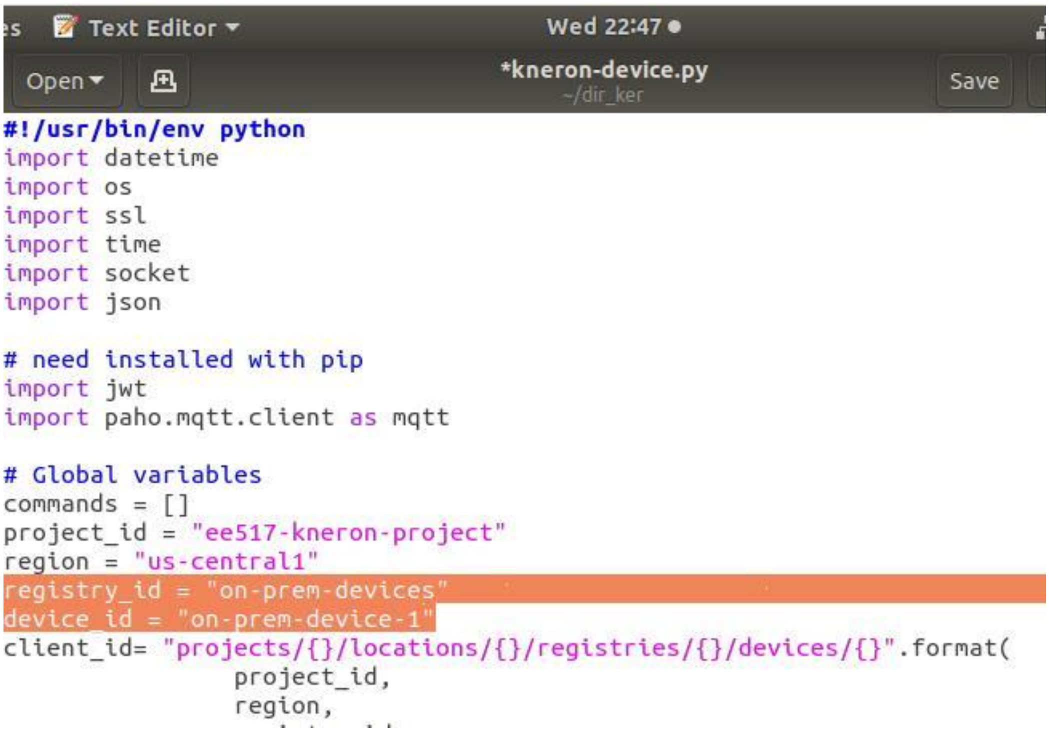
change the marked part as yours



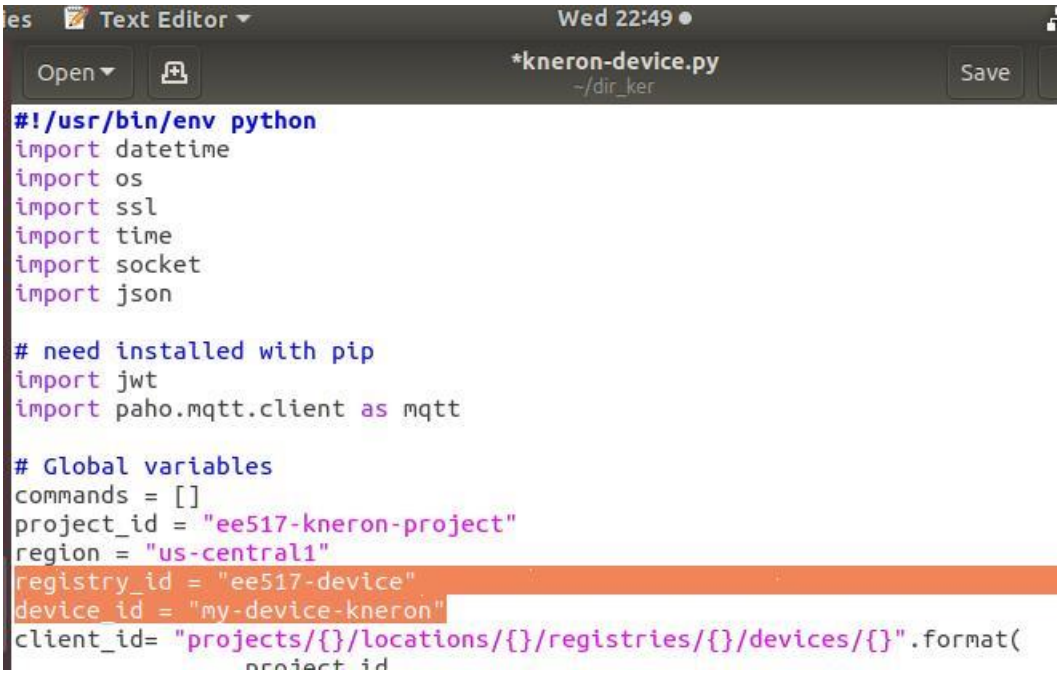
```
#!/usr/bin/env python
import datetime
import os
import ssl
import time
import socket
import json

# need installed with pip
import jwt
import paho.mqtt.client as mqtt

# Global variables
commands = []
project_id = "<GCP_PROJECT_ID>"
region = "us-central1"
registry_id = "on-prem-devices"
device_id = "on-prem-device-1"
client_id= "projects/{}/locations/{}/registries/{}/devices/{}".format(
                project_id,
                region,
                registry_id,
                device_id)

# callback that runs when connection is successful
def on_connect(client, unused_userdata, unused_flags, rc):
    print('on_connect', mqtt.connack_string(rc))
```



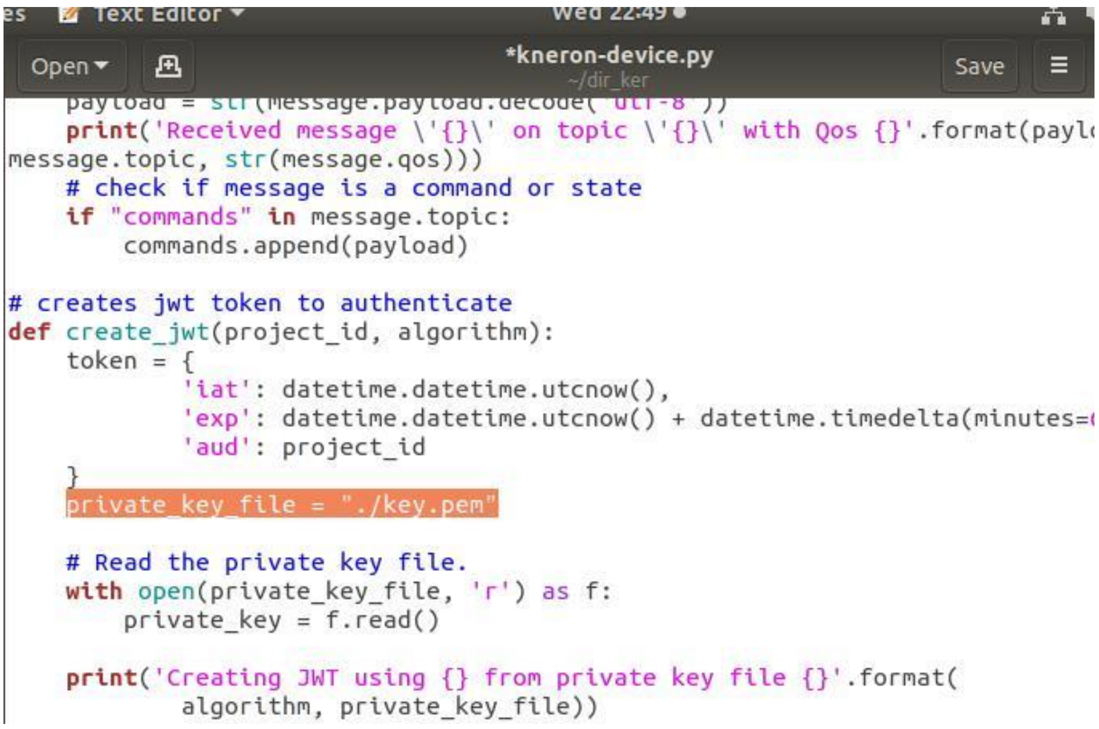
```
#!/usr/bin/env python
import datetime
import os
import ssl
import time
import socket
import json

# need installed with pip
import jwt
import paho.mqtt.client as mqtt

# Global variables
commands = []
project_id = "ee517-kneron-project"
region = "us-central1"
registry_id = "on-prem-devices"
device_id = "on-prem-device-1"
client_id= "projects/{}/locations/{}/registries/{}/devices/{}".format(
                project_id,
                region,
                registry_id,
                device_id)
```



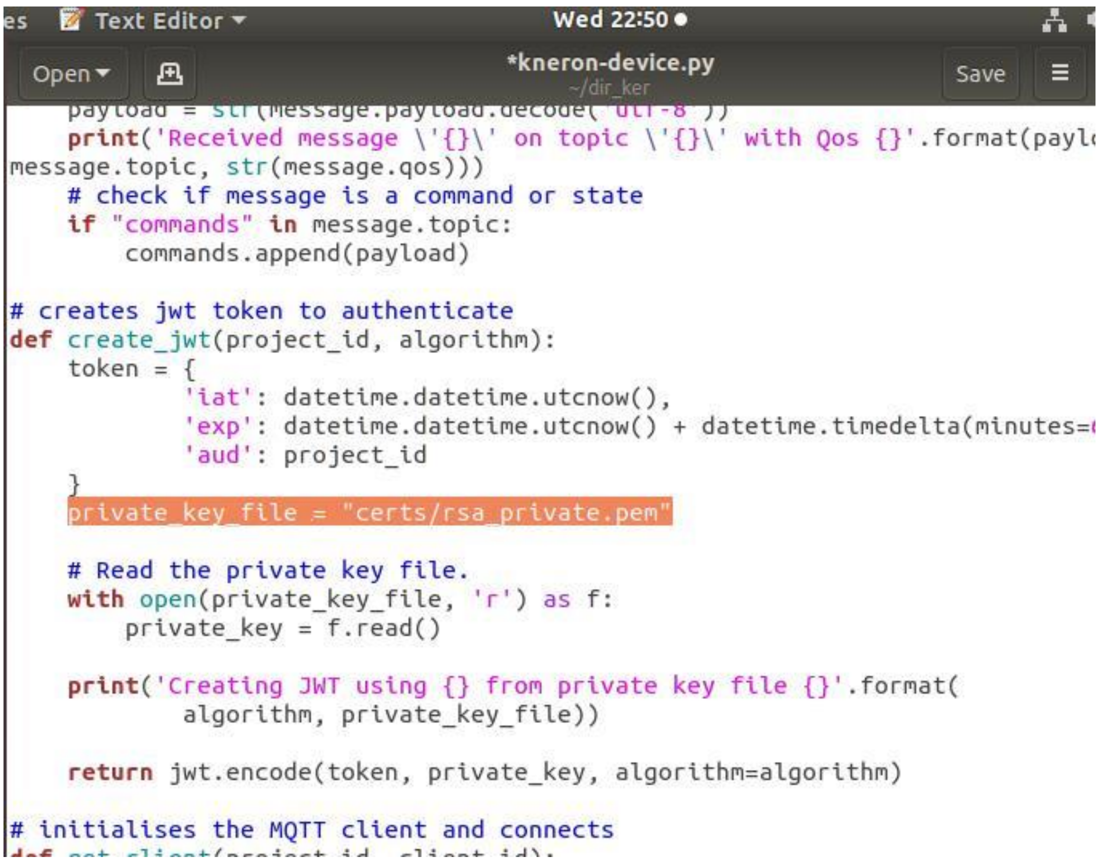
```
#!/usr/bin/env python
import datetime
import os
import ssl
import time
import socket
import json

# need installed with pip
import jwt
import paho.mqtt.client as mqtt

# Global variables
commands = []
project_id = "ee517-kneron-project"
region = "us-central1"
registry_id = "on-prem-devices"
device_id = "on-prem-device-1"
client_id= "projects/{}/locations/{}/registries/{}/devices/{}".format(
                project_id,
                region,
```



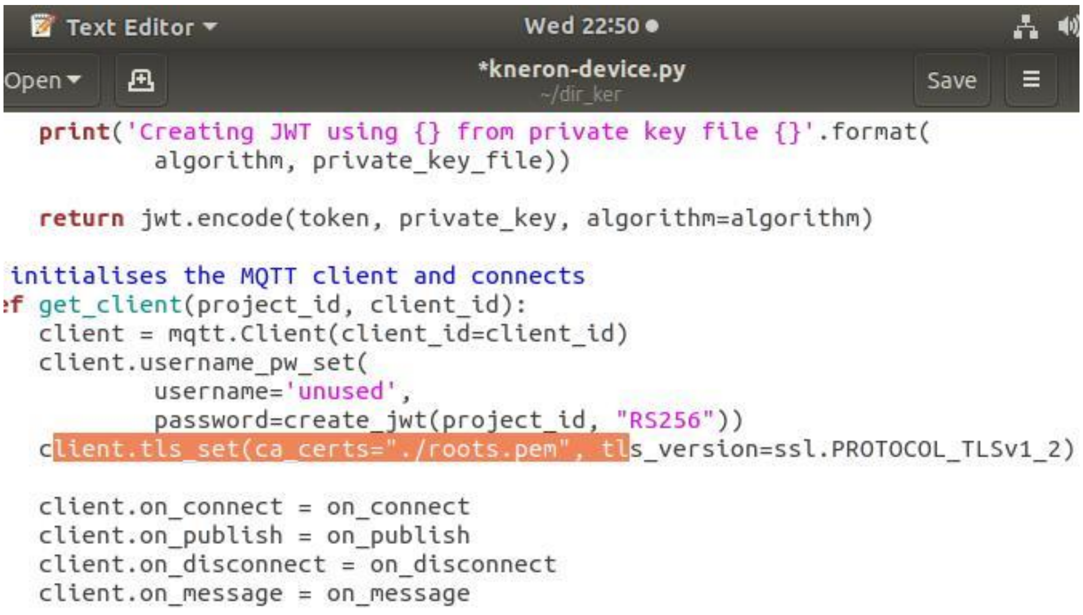
```
#!/usr/bin/env python
import datetime
import os
import ssl
import time
import socket
import json

# need installed with pip
import jwt
import paho.mqtt.client as mqtt

# Global variables
commands = []
project_id = "ee517-kneron-project"
region = "us-central1"
registry_id = "ee517-device"
device_id = "my-device-kneron"
client_id= "projects/{}/locations/{}/registries/{}/devices/{}".format(
```



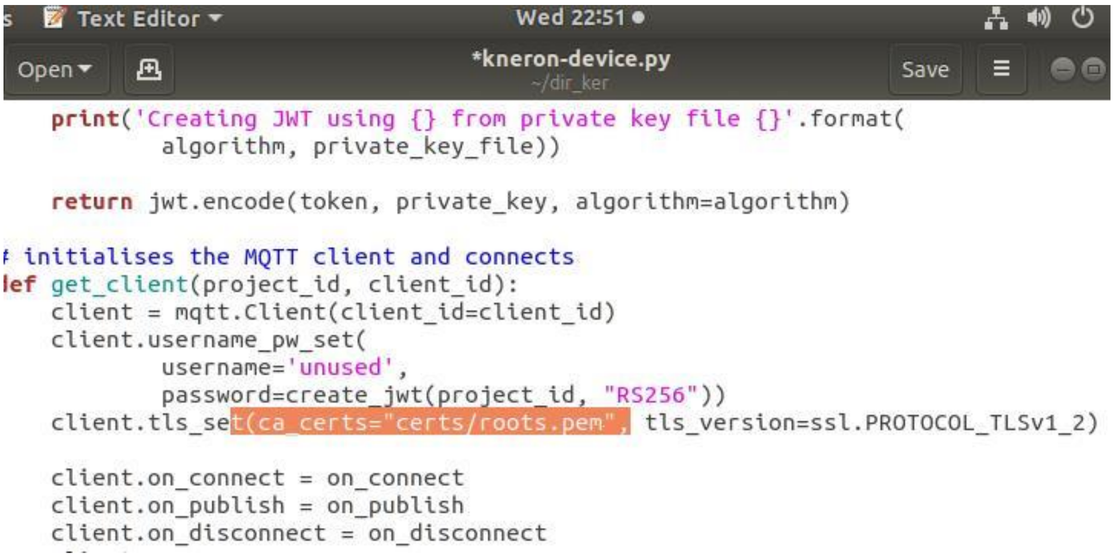
```
    payload = str(message.payload.decode('utf-8'))
    print('Received message \'{}\' on topic \'{}\' with Qos {}'.format(payl
message.topic, str(message.qos)))
    # check if message is a command or state
    if "commands" in message.topic:
        commands.append(payload)

# creates jwt token to authenticate
def create_jwt(project_id, algorithm):
    token = {
            'iat': datetime.datetime.utcnow(),
            'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=
            'aud': project_id
    }
    private_key_file = "./key.pem"

    # Read the private key file.
    with open(private_key_file, 'r') as f:
        private_key = f.read()

    print('Creating JWT using {} from private key file {}'.format(
            algorithm, private_key_file))
```



```
    payload = str(message.payload.decode('utf-8'))
    print('Received message \'{}\' on topic \'{}\' with Qos {}'.format(payl
message.topic, str(message.qos)))
    # check if message is a command or state
    if "commands" in message.topic:
        commands.append(payload)

# creates jwt token to authenticate
def create_jwt(project_id, algorithm):
    token = {
            'iat': datetime.datetime.utcnow(),
            'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=
            'aud': project_id
    }
    private_key_file = "certs/rsa_private.pem"

    # Read the private key file.
    with open(private_key_file, 'r') as f:
        private_key = f.read()

    print('Creating JWT using {} from private key file {}'.format(
            algorithm, private_key_file))

    return jwt.encode(token, private_key, algorithm=algorithm)

# initialises the MQTT client and connects
def get_client(project_id, client_id):
```



```
    print('Creating JWT using {} from private key file {}'.format(
            algorithm, private_key_file))

    return jwt.encode(token, private_key, algorithm=algorithm)

 initialises the MQTT client and connects
ef get_client(project_id, client_id):
    client = mqtt.Client(client_id=client_id)
    client.username_pw_set(
            username='unused',
            password=create_jwt(project_id, "RS256"))
    client.tls_set(ca_certs="./roots.pem", tls_version=ssl.PROTOCOL_TLSv1_2)

    client.on_connect = on_connect
    client.on_publish = on_publish
    client.on_disconnect = on_disconnect
    client.on_message = on_message
```



```
    print('Creating JWT using {} from private key file {}'.format(
            algorithm, private_key_file))

    return jwt.encode(token, private_key, algorithm=algorithm)

# initialises the MQTT client and connects
lef get_client(project_id, client_id):
    client = mqtt.Client(client_id=client_id)
    client.username_pw_set(
            username='unused',
            password=create_jwt(project_id, "RS256"))
    client.tls_set(ca_certs="certs/roots.pem", tls_version=ssl.PROTOCOL_TLSv1_2)

    client.on_connect = on_connect
    client.on_publish = on_publish
    client.on_disconnect = on_disconnect
```

Create a .txt file and enter:

cryptography==3.2.1

google-api-python-client==1.12.8

google-auth-httplib2==0.0.4

google-auth==1.23.0

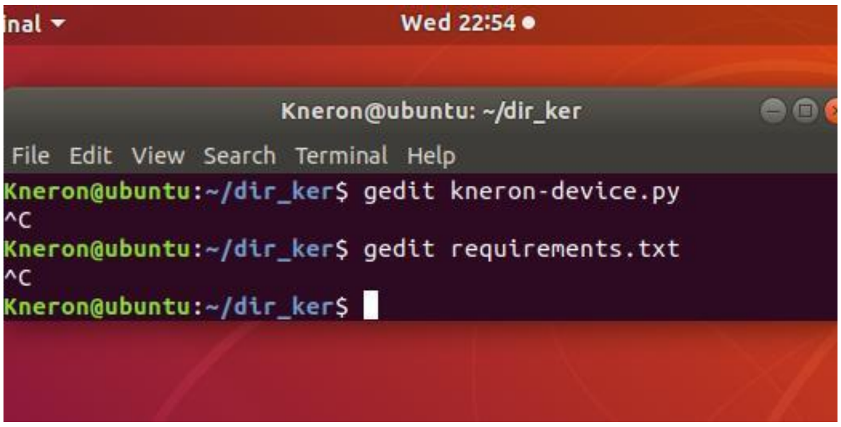
google-cloud-pubsub==1.7.0

google-cloud-iot==2.0.1

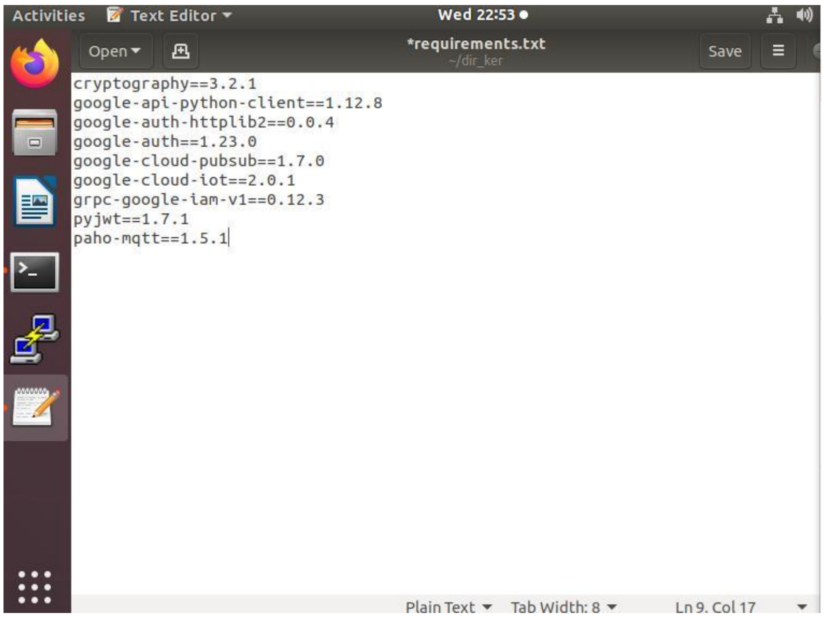
grpc-google-iam-v1==0.12.3

pyjwt==1.7.1

paho-mqtt==1.5.1

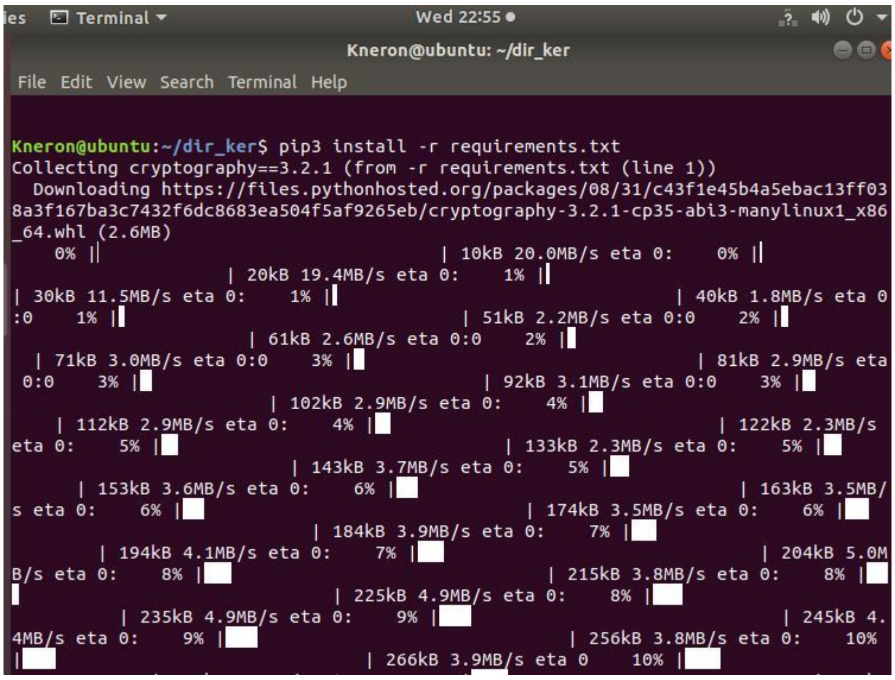


```
Kneron@ubuntu:~/dir_ker$ gedit kneron-device.py
^C
Kneron@ubuntu:~/dir_ker$ gedit requirements.txt
^C
Kneron@ubuntu:~/dir_ker$ 
```



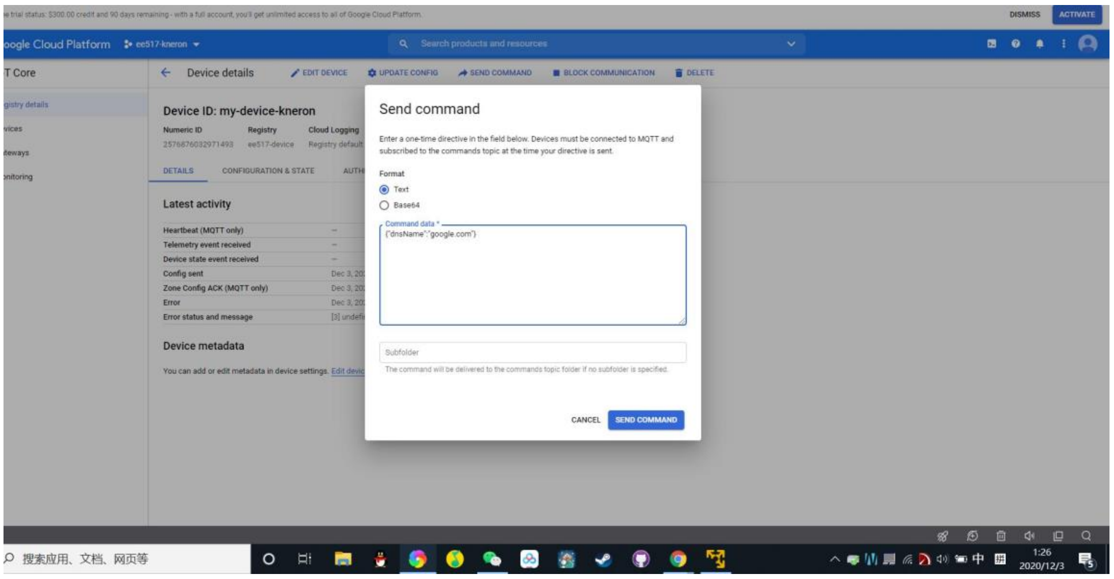
```
cryptography==3.2.1
google-api-python-client==1.12.8
google-auth-httplib2==0.0.4
google-auth==1.23.0
google-cloud-pubsub==1.7.0
google-cloud-iot==2.0.1
grpc-google-iam-v1==0.12.3
pyjwt==1.7.1
paho-mqtt==1.5.1
```

Enter: pip3 install -r reuqirements.txt



```
Kneron@ubuntu:~/dir_ker$ pip3 install -r requirements.txt
Collecting cryptography==3.2.1 (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/08/31/c43f1e45b4a5ebac13ff03
8a3f167ba3c7432f6dc8683ea504f5af9265eb/cryptography-3.2.1-cp35-abi3-manylinux1_x86
_64.whl (2.6MB)
    0% ||                               | 10kB 20.0MB/s eta 0:    0% ||
                    | 20kB 19.4MB/s eta 0:    1% ||
| 30kB 11.5MB/s eta 0:    1% |█                              | 40kB 1.8MB/s eta 0
:0    1% |█                             | 51kB 2.2MB/s eta 0:0    2% |█
                    | 61kB 2.6MB/s eta 0:0    2% |█
  | 71kB 3.0MB/s eta 0:0    3% |█                            | 81kB 2.9MB/s eta
 0:0    3% |█                            | 92kB 3.1MB/s eta 0:0    3% |█
                    | 102kB 2.9MB/s eta 0:    4% |█
    | 112kB 2.9MB/s eta 0:    4% |█                          | 122kB 2.3MB/s
eta 0:    5% |█                          | 133kB 2.3MB/s eta 0:    5% |█
                    | 143kB 3.7MB/s eta 0:    5% |█
      | 153kB 3.6MB/s eta 0:    6% |█                       | 163kB 3.5MB/
s eta 0:    6% |█                        | 174kB 3.5MB/s eta 0:    6% |█
                    | 184kB 3.9MB/s eta 0:    7% |█
      | 194kB 4.1MB/s eta 0:    7% |█                      | 204kB 5.0M
B/s eta 0:    8% |█                      | 215kB 3.8MB/s eta 0:    8% |█
█                    | 225kB 4.9MB/s eta 0:    8% |█
      | 235kB 4.9MB/s eta 0:    9% |█                    | 245kB 4.
4MB/s eta 0:    9% |█                    | 256kB 3.8MB/s eta 0:    10%
|█                    | 266kB 3.9MB/s eta 0    10% |█
```

Go the IoT Core, click Devices, click Send Command: {"dnsName":"google.com"}



Send command

Enter a one-time directive in the field below. Devices must be connected to MQTT and subscribed to the commands topic at the time your directive is sent.

Format

[✓ Text] Base64

Command data *

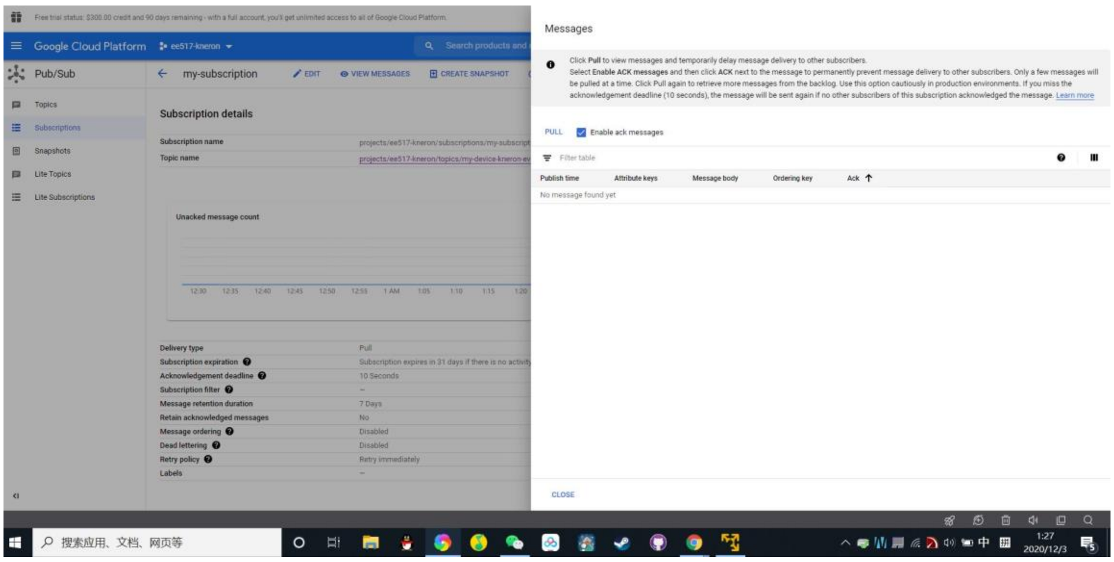
```
{"dnsName":"google.com"}
```

Subfolder

The command will be delivered to the commands topic folder if no subfolder is specified.

Go to Pub/Sub, click Subscription and View Message to check

Also need mark and pull as below

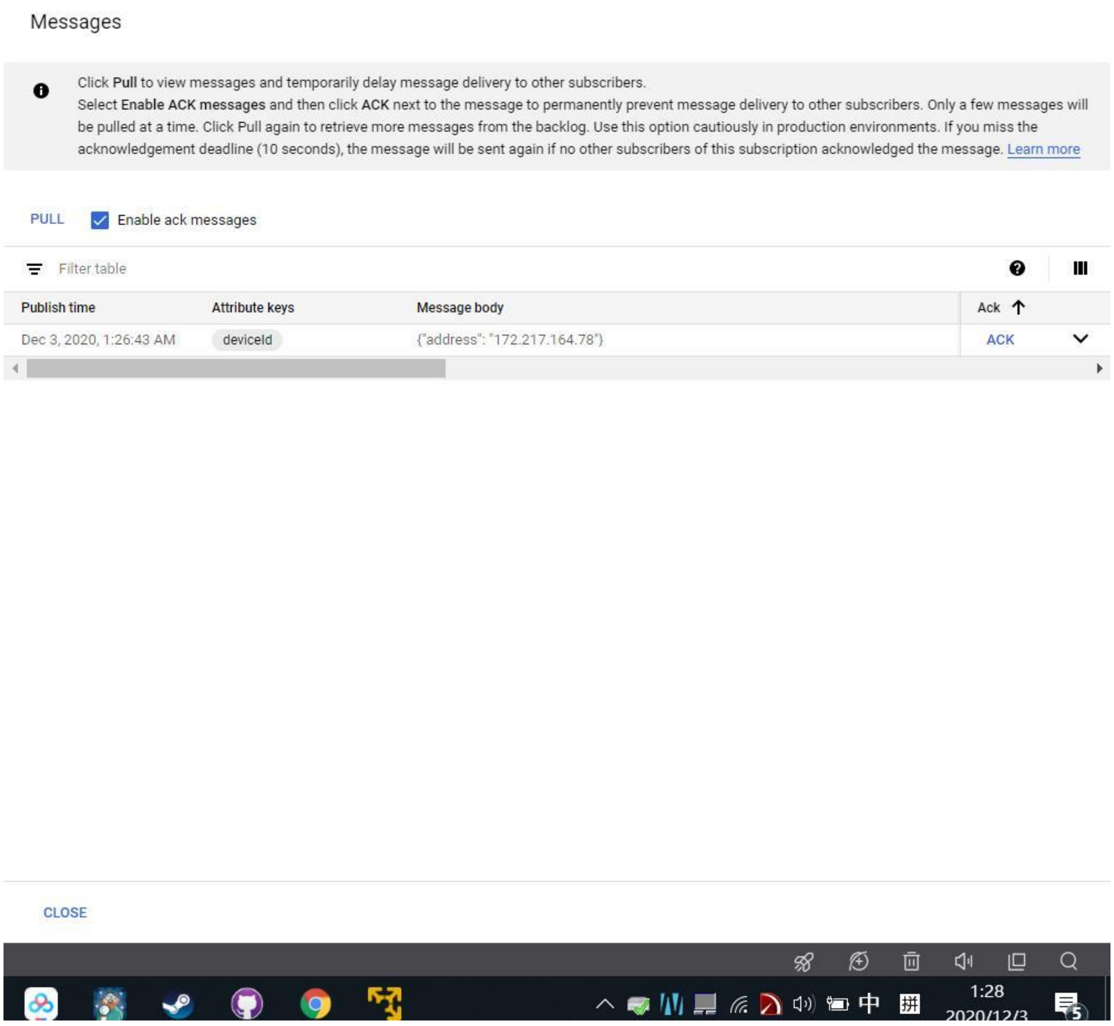


Messages

Click **Pull** to view messages and temporarily delay message delivery to other subscribers.
Select **Enable ACK messages** and then click **ACK** next to the message to permanently prevent message delivery to other subscribers. Only a few messages will be pulled at a time. Click Pull again to retrieve more messages from the backlog. Use this option cautiously in production environments. If you miss the acknowledgement deadline (10 seconds), the message will be sent again if no other subscribers of this subscription acknowledged the message. Learn more

PULL [✓ Enable ack messages]

Publish time	Attribute keys	Message body	Ordering key	Ack
No message found yet				



Messages

Click **Pull** to view messages and temporarily delay message delivery to other subscribers.
Select **Enable ACK messages** and then click **ACK** next to the message to permanently prevent message delivery to other subscribers. Only a few messages will be pulled at a time. Click Pull again to retrieve more messages from the backlog. Use this option cautiously in production environments. If you miss the acknowledgement deadline (10 seconds), the message will be sent again if no other subscribers of this subscription acknowledged the message. Learn more

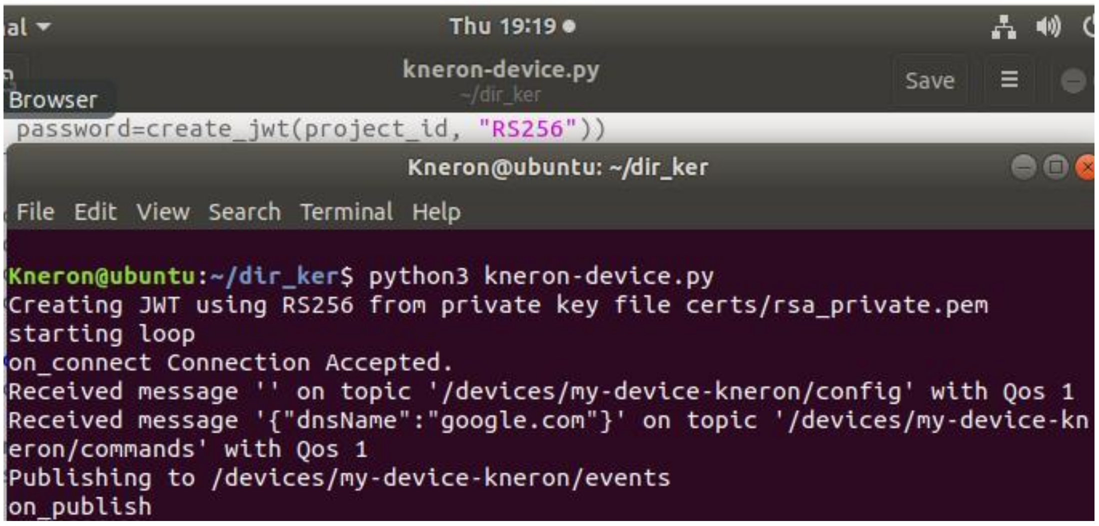
PULL ☑ Enable ack messages

Filter table

Publish time	Attribute keys	Message body	Ack
Dec 3, 2020, 1:26:43 AM	deviceId	{"address": "172.217.164.78"}	ACK

CLOSE

Run the .py file



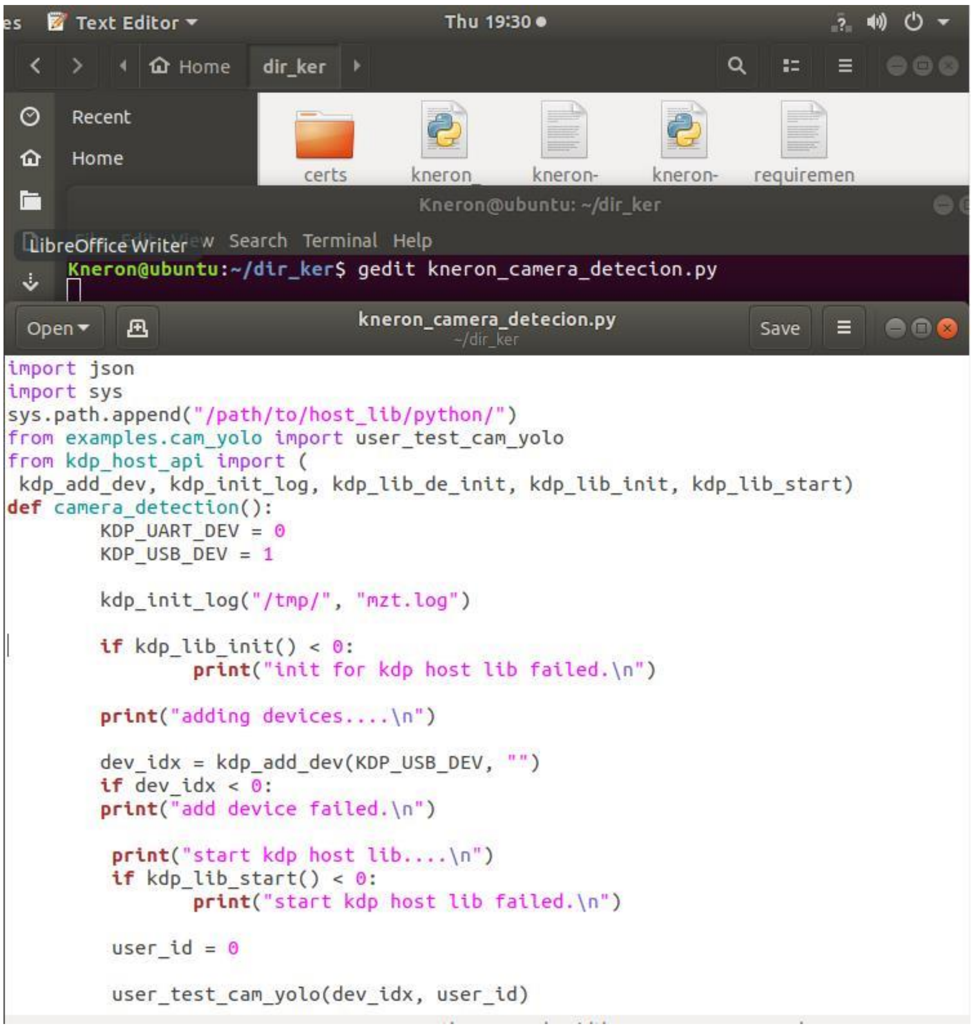
```
password=create_jwt(project_id, "RS256"))
```

Kneron@ubuntu: ~/dir_ker

```
Kneron@ubuntu:~/dir_ker$ python3 kneron-device.py
Creating JWT using RS256 from private key file certs/rsa_private.pem
starting loop
on_connect Connection Accepted.
Received message '' on topic '/devices/my-device-kneron/config' with Qos 1
Received message '{"dnsName":"google.com"}' on topic '/devices/my-device-kn
eron/commands' with Qos 1
Publishing to /devices/my-device-kneron/events
on_publish
```

Create a detection file and enter as below (change the system path to yours):

```
import json
import sys
sys.path.append("/home//kneron/host_lib/python/")
from examples.cam_yolo import user_test_cam_yolo
from kdp_host_api import (
    kdp_add_dev, kdp_init_log, kdp_lib_de_init, kdp_lib_init,
kdp_lib_start)
def camera_detection():
    KDP_UART_DEV = 0
    KDP_USB_DEV = 1

    kdp_init_log("/tmp/", "mzt.log")

    if kdp_lib_init() < 0:
        print("init for kdp host lib failed.\n")

    print("adding devices....\n")

    dev_idx = kdp_add_dev(KDP_USB_DEV, "")
    if dev_idx < 0:
        print("add device failed.\n")

    print("start kdp host lib....\n")
    if kdp_lib_start() < 0:
        print("start kdp host lib failed.\n")

    user_id = 0



    user_test_cam_yolo(dev_idx, user_id)
```



```
Kneron@ubuntu:~/dir_ker$ gedit kneron_camera_detecion.py
```

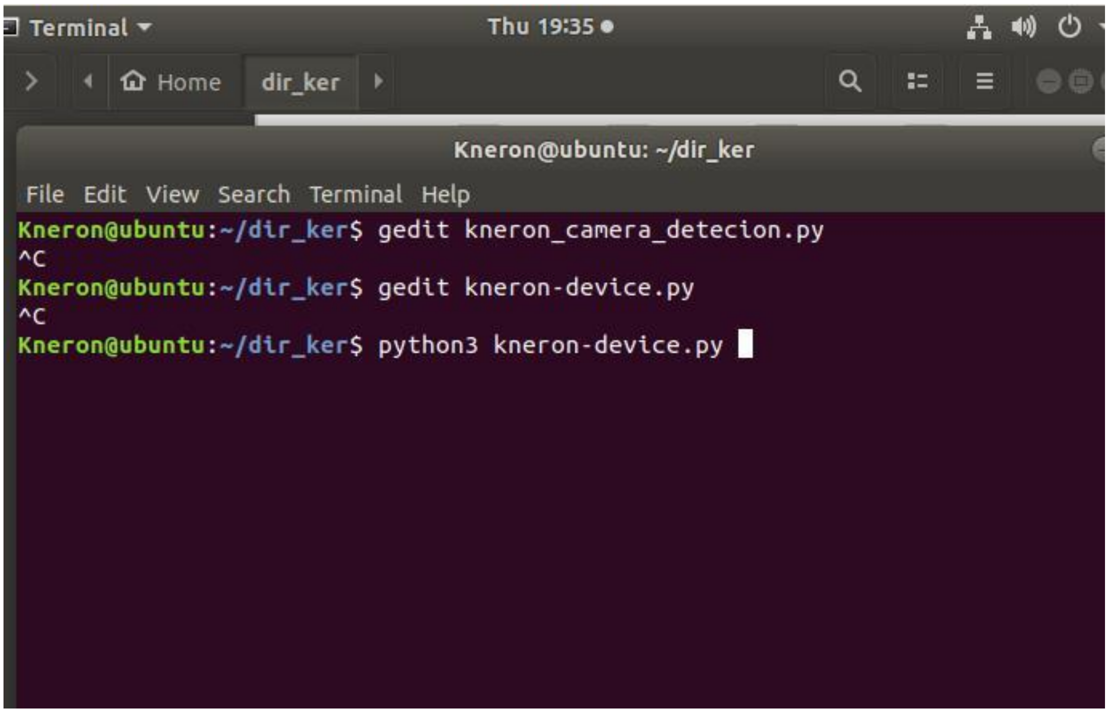
```
import json
import sys
sys.path.append("/path/to/host_lib/python/")
from examples.cam_yolo import user_test_cam_yolo
from kdp_host_api import (
 kdp_add_dev, kdp_init_log, kdp_lib_de_init, kdp_lib_init, kdp_lib_start)
def camera_detection():
        KDP_UART_DEV = 0
        KDP_USB_DEV = 1

        kdp_init_log("/tmp/", "mzt.log")

        if kdp_lib_init() < 0:
                print("init for kdp host lib failed.\n")

        print("adding devices....\n")

        dev_idx = kdp_add_dev(KDP_USB_DEV, "")
        if dev_idx < 0:
        print("add device failed.\n")

         print("start kdp host lib....\n")
         if kdp_lib_start() < 0:
                print("start kdp host lib failed.\n")

         user_id = 0

         user_test_cam_yolo(dev_idx, user_id)
```

Then replace the first .py files content as below (change the project_id, registry_id, device_id to yours):

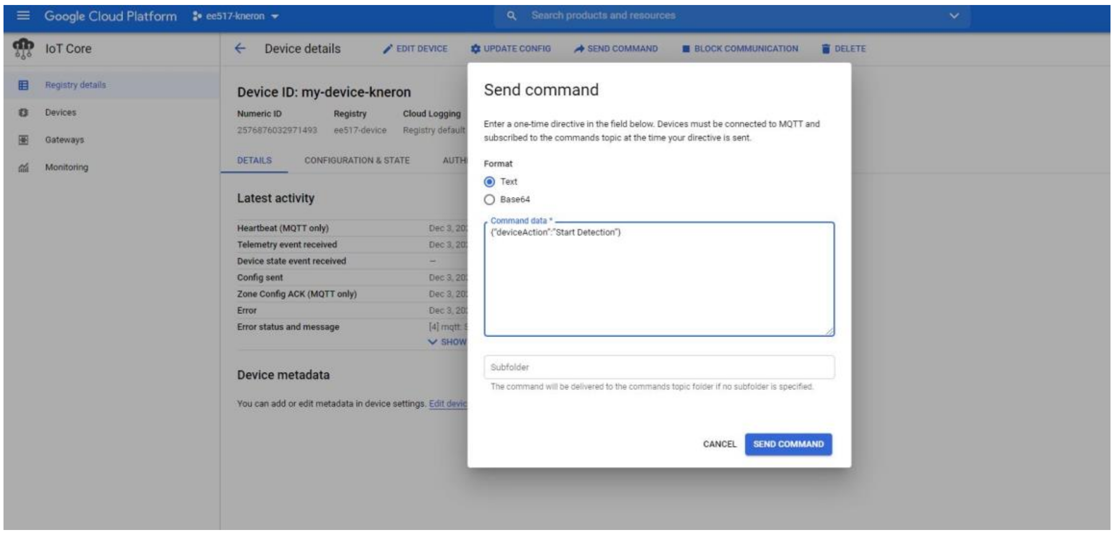
```
#!/usr/bin/env python
import datetime
import os
import ssl
import time
import socket
import json

# need installed with pip
import jwt
import paho.mqtt.client as mqtt
from kneron_camera_detection import camera_detection
# Global variables
commands = []
project_id = "ee517-kneron"
```

```
region = "us-central1"
registry_id = "ee517-device"
device_id = "my-device-kneron"
client_id=
"projects/{}/locations/{}/registries/{}/devices/{}".format(
              project_id,
              region,
              registry_id,
              device_id)

# callback that runs when connection is successful
def on_connect(client, unused_userdata, unused_flags, rc):
    print('on_connect', mqtt.connack_string(rc))

# callback that runs when disconnection is successful
def on_disconnect(unused_client, unused_userdata, rc):
    print('on_disconnect', error_str(rc))

# callback that runs when data is published
def on_publish(unused_client, unused_userdata, unused_mid):
    print('on_publish')

# callback that runs when a message is recieved from a
subscription
def on_message(unused_client, unused_userdata, message):
    global commands
    payload = str(message.payload.decode('utf-8'))
    print('Received message \'{}\' on topic \'{}\' with Qos
{}'.format(payload, message.topic, str(message.qos)))
    # check if message is a command or state
    if "commands" in message.topic:
        commands.append(payload)

# creates jwt token to authenticate
def create_jwt(project_id, algorithm):
    token = {
            'iat': datetime.datetime.utcnow(),
            'exp': datetime.datetime.utcnow() +
datetime.timedelta(minutes=60),
            'aud': project_id
    }
    private_key_file = "resources/rsa_private.pem"

    # Read the private key file.
    with open(private_key_file, 'r') as f:
        private_key = f.read()

    print('Creating JWT using {} from private key file {}'.format(
            algorithm, private_key_file))

    return jwt.encode(token, private_key, algorithm=algorithm)

# initialises the MQTT client and connects
def get_client(project_id, client_id):
    client = mqtt.Client(client_id=client_id)
    client.username_pw_set(
            username='unused',
            password=create_jwt(project_id, "RS256"))
    client.tls_set(ca_certs="resources/roots.pem",
tls_version=ssl.PROTOCOL_TLSv1_2)

    client.on_connect = on_connect
    client.on_publish = on_publish
    client.on_disconnect = on_disconnect
    client.on_message = on_message

    # Connect to the Google MQTT bridge.
    client.connect("mqtt.googleapis.com", 8883)

    mqtt_config_topic = '/devices/{}/config'.format(device_id)
    client.subscribe(mqtt_config_topic, qos=1)

    mqtt_command_topic =
'/devices/{}/commands/#'.format(device_id)
    client.subscribe(mqtt_command_topic, qos=1)

    return client

def main():
    global project_id
    global client_id
    global commands
    client = get_client(project_id, client_id)
    client.loop_start()

    print("starting loop")
    while True:
        # check if we have recieved any commands
        if len(commands) > 0:
            command = commands.pop(0)
            # parse the command and get the dns name
            #print(command)
            loaded_json = json.loads(command)
            print(loaded_json)
            if loaded_json["deviceAction"] == "Start Detection":
                camera_detection()
            # do a lookup on the dns name


            # publish the results back to MQTT
            payload = {"Result": "Camera Detection Finished"}
            mqtt_topic = '/devices/{}/events'.format(device_id)
            print('Publishing to {}'.format(mqtt_topic))
            infot = client.publish(mqtt_topic, json.dumps(payload),
qos=0, retain=False)
            infot.wait_for_publish()
        # we sleep each loop to keep within the MQTT limits
        time.sleep(1)

if __name__ == '__main__':
    main()
```

Run the first .py file, and also go to the IoT Core to send the command ({“deviceAction”:”Start Detection”}), after command send, the terminal will start camera detection
Pay attention to the quotation marks



```
Kneron@ubuntu:~/dir_ker$ gedit kneron_camera_detecion.py
^C
Kneron@ubuntu:~/dir_ker$ gedit kneron-device.py
^C
Kneron@ubuntu:~/dir_ker$ python3 kneron-device.py
```



Google Cloud Platform ee517-kneron

IoT Core

Device details — EDIT DEVICE · UPDATE CONFIG · SEND COMMAND · BLOCK COMMUNICATION · DELETE

Registry details
Devices
Gateways
Monitoring

Device ID: my-device-kneron

Numeric ID	Registry	Cloud Logging
2576876032971493	ee517-device	Registry default

DETAILS · CONFIGURATION & STATE · AUTH

Latest activity

Heartbeat (MQTT only)	Dec 3, 20
Telemetry event received	Dec 3, 20
Device state event received	–
Config sent	Dec 3, 20
Zone Config ACK (MQTT only)	Dec 3, 20
Error	Dec 3, 20
Error status and message	[4] mqtt:

Device metadata

You can add or edit metadata in device settings. Edit devic

Send command

Enter a one-time directive in the field below. Devices must be connected to MQTT and subscribed to the commands topic at the time your directive is sent.

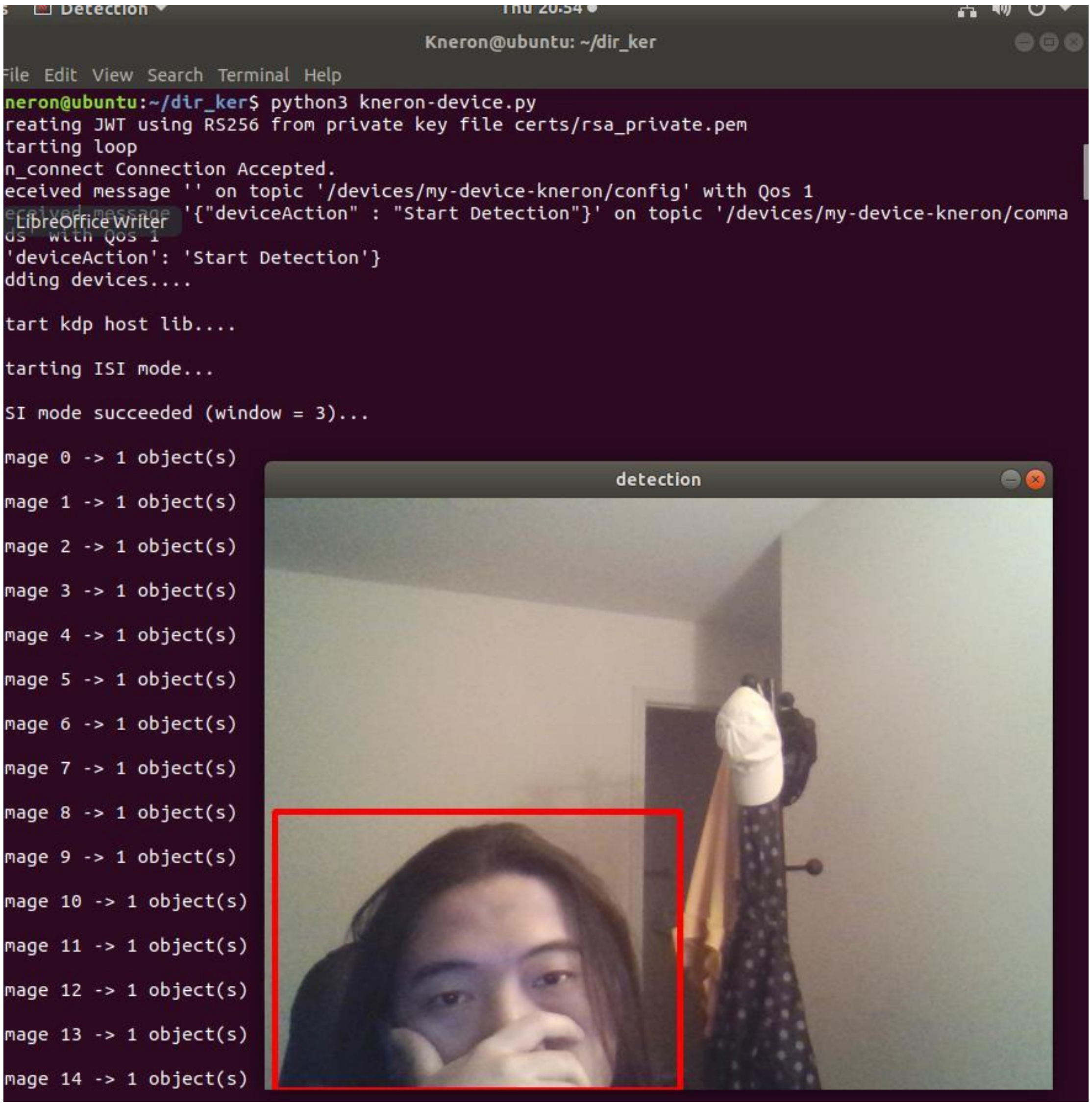
Format

[✓ Text] Base64

Command data *
{"deviceAction":"Start Detection"}

Subfolder
The command will be delivered to the commands topic folder if no subfolder is specified.

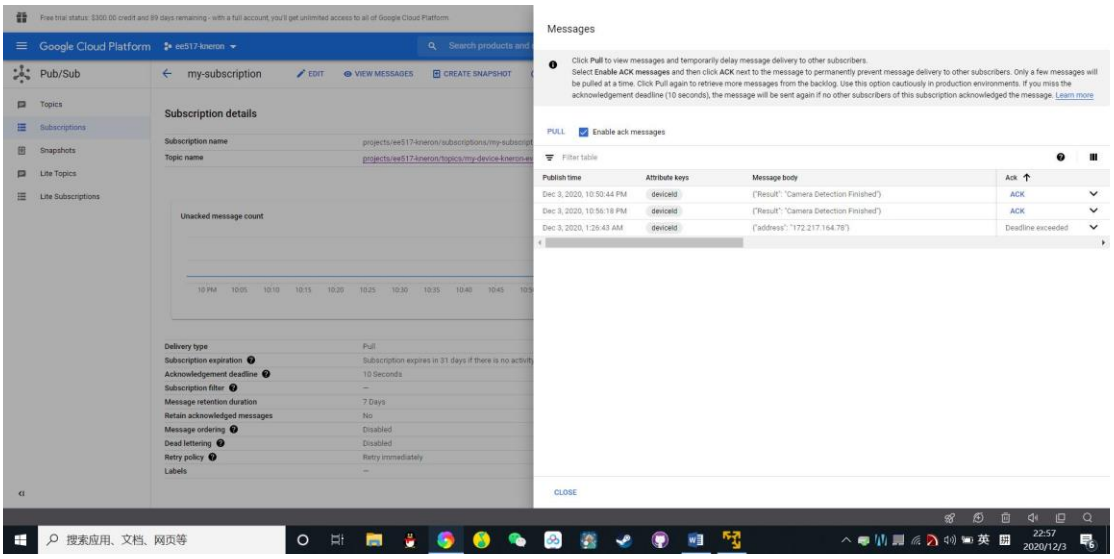
CANCEL · SEND COMMAND



Kneron@ubuntu: ~/dir_ker

```
neron@ubuntu:~/dir_ker$ python3 kneron-device.py
reating JWT using RS256 from private key file certs/rsa_private.pem
tarting loop
n_connect Connection Accepted.
eceived message '' on topic '/devices/my-device-kneron/config' with Qos 1
eceived message '{"deviceAction" : "Start Detection"}' on topic '/devices/my-device-kneron/comma
ds' with Qos 1
'deviceAction': 'Start Detection'}
dding devices....

tart kdp host lib....

tarting ISI mode...

SI mode succeeded (window = 3)...

mage 0 -> 1 object(s)

mage 1 -> 1 object(s)

mage 2 -> 1 object(s)

mage 3 -> 1 object(s)

mage 4 -> 1 object(s)

mage 5 -> 1 object(s)

mage 6 -> 1 object(s)

mage 7 -> 1 object(s)

mage 8 -> 1 object(s)

mage 9 -> 1 object(s)

mage 10 -> 1 object(s)

mage 11 -> 1 object(s)

mage 12 -> 1 object(s)

mage 13 -> 1 object(s)

mage 14 -> 1 object(s)
```

After detection finished, you can go check the Subscription Messages



Free trial status: $300.00 credit and 89 days remaining - with a full account, you'll get unlimited access to all of Google Cloud Platform

Google Cloud Platform · ee517-kneron

Pub/Sub · my-subscription · EDIT · VIEW MESSAGES · CREATE SNAPSHOT

Subscription details

Subscription name	projects/ee517-kneron/subscriptions/my-subscript
Topic name	projects/ee517-kneron/topics/my-device-kneron-e

Unacked message count

Delivery type	Pull
Subscription expiration	Subscription expires in 31 days if there is no activity
Acknowledgement deadline	10 Seconds
Subscription filter	—
Message retention duration	7 Days
Retain acknowledged messages	No
Message ordering	Disabled
Dead lettering	Disabled
Retry policy	Retry immediately
Labels	—

Messages

Click **Pull** to view messages and temporarily delay message delivery to other subscribers.
Select **Enable ACK messages** and then click **ACK** next to the message to permanently prevent message delivery to other subscribers. Only a few messages will be pulled at a time. Click Pull again to retrieve more messages from the backlog. Use this option cautiously in production environments. If you miss the acknowledgement deadline (10 seconds), the message will be sent again if no other subscribers of this subscription acknowledged the message. Learn more

PULL ✓ Enable ack messages

Publish time	Attribute keys	Message body	Ack
Dec 3, 2020, 10:50:44 PM	deviceId	{"Result": "Camera Detection Finished"}	ACK
Dec 3, 2020, 10:56:18 PM	deviceId	{"Result": "Camera Detection Finished"}	ACK
Dec 3, 2020, 1:26:43 AM	deviceId	{"address": "172.217.164.78"}	Deadline exceeded