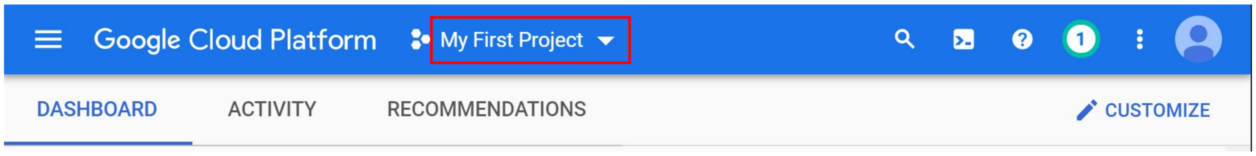
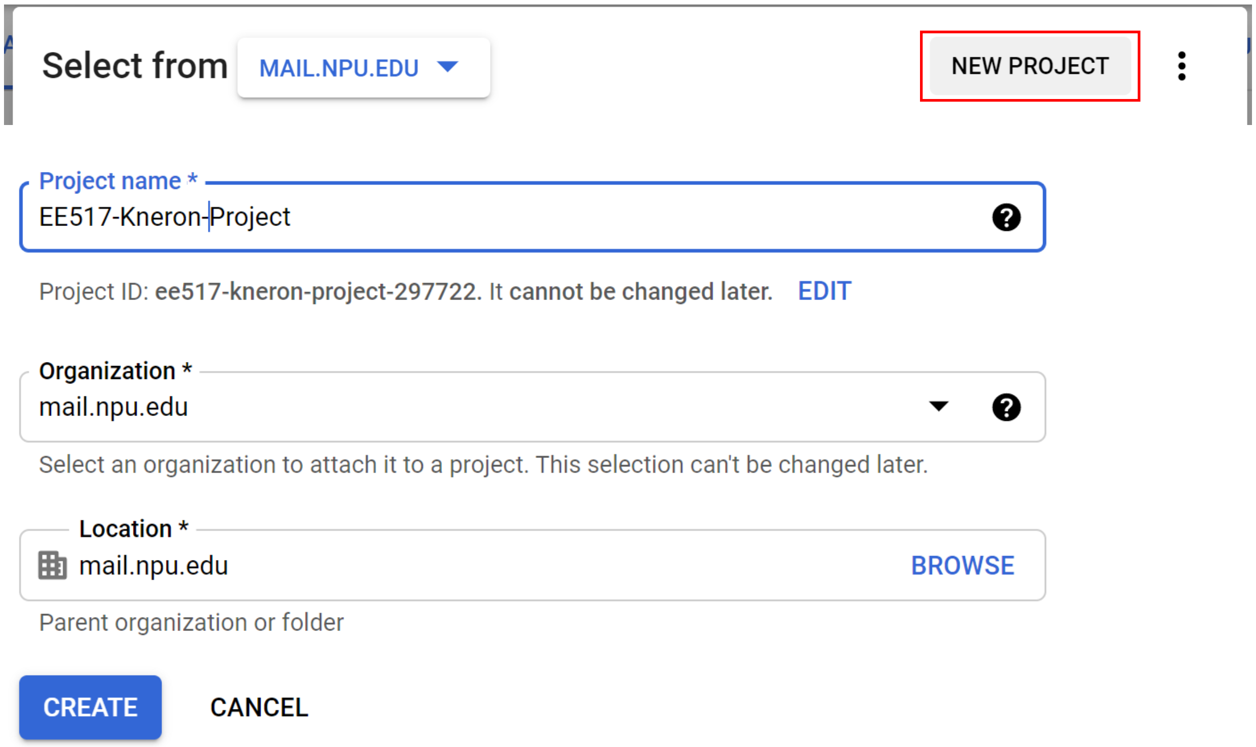
Open Google Cloud Platform Console and create a new project





Project name *
EE517-Kneron-Project

Project ID: ee517-kneron-project-297722. It cannot be changed later. EDIT

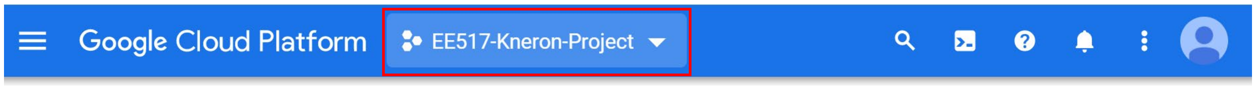
Organization *
mail.npu.edu

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

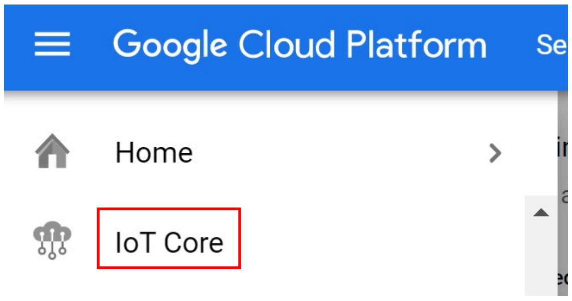
Location *
mail.npu.edu

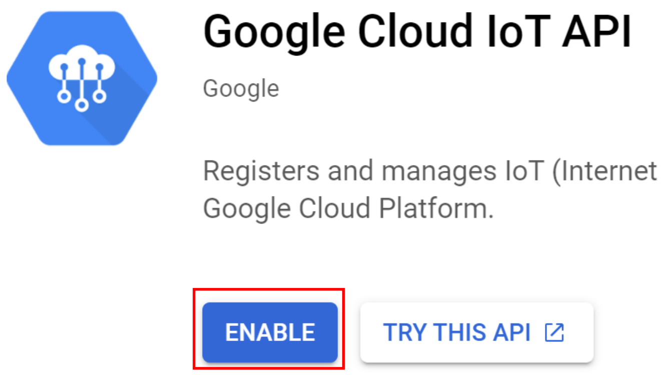
Parent organization or folder

Make sure you are working on your recently created project

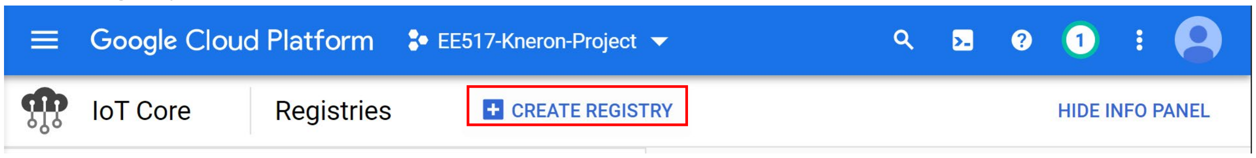


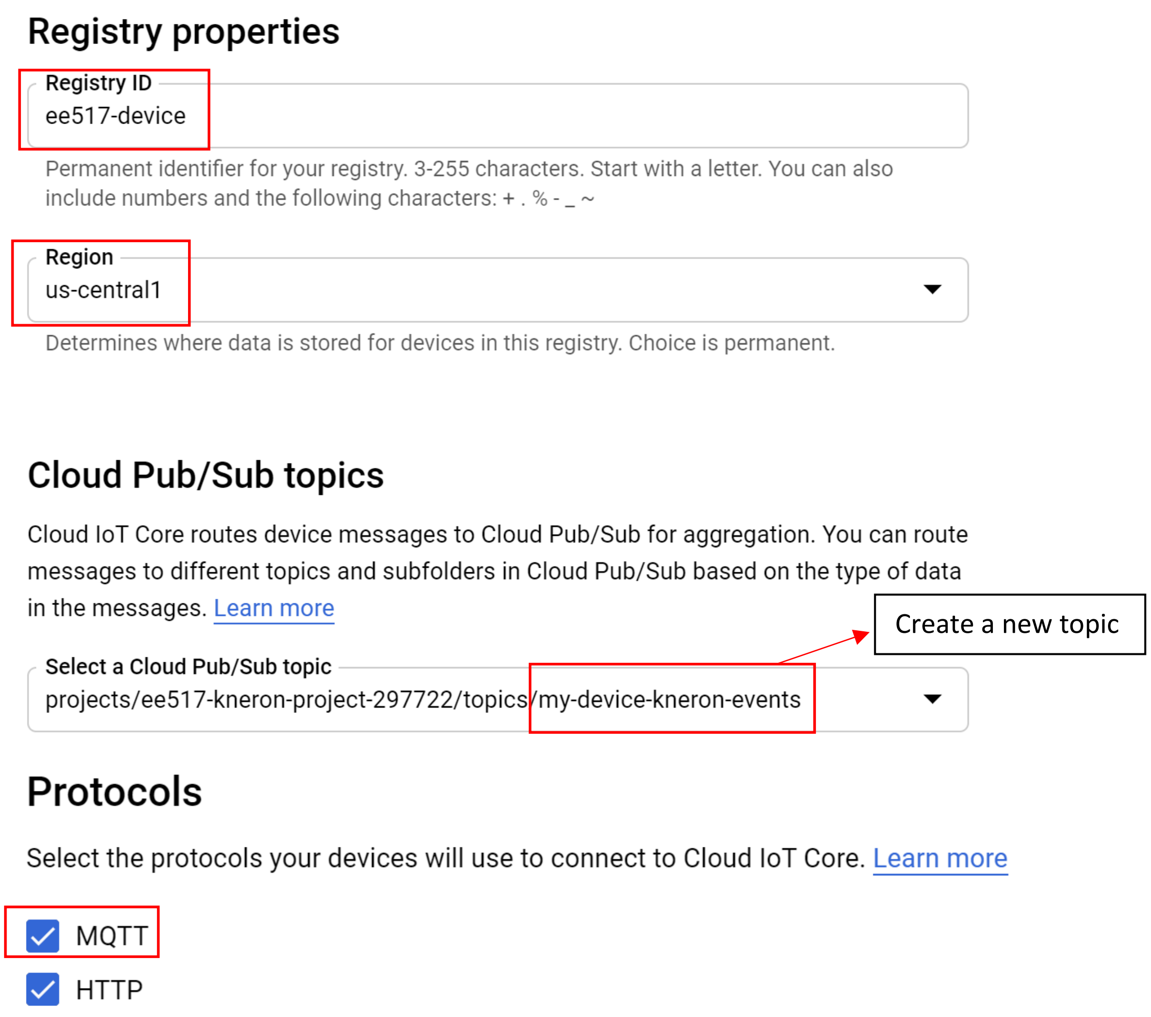
Enable Cloud IoT Core. You may need to enable billing if not already enabled.





Create registry





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: projects/ee517-kneron-project-297722/topics/my-device-kneron-events

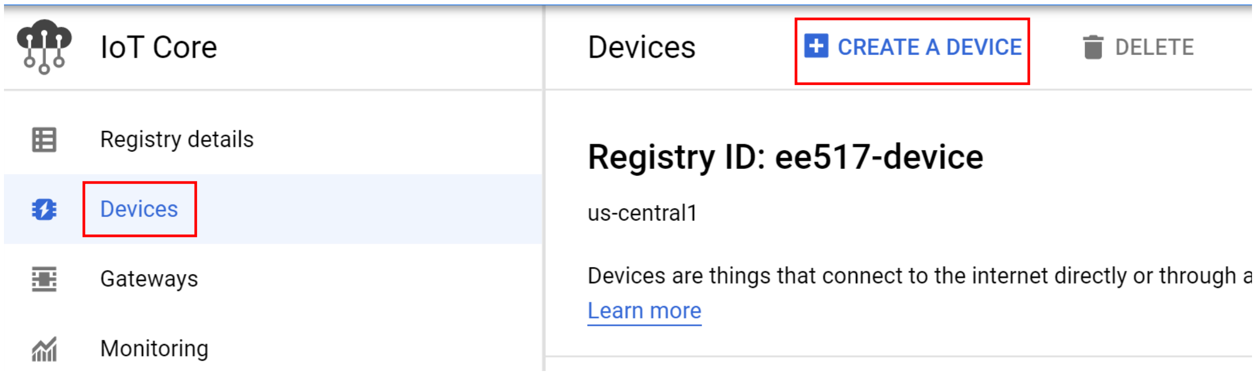
Create a new topic

Protocols

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

- [✓] MQTT
- [✓] HTTP

Create a new device

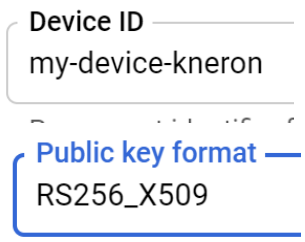


Registry ID: ee517-device

us-central1

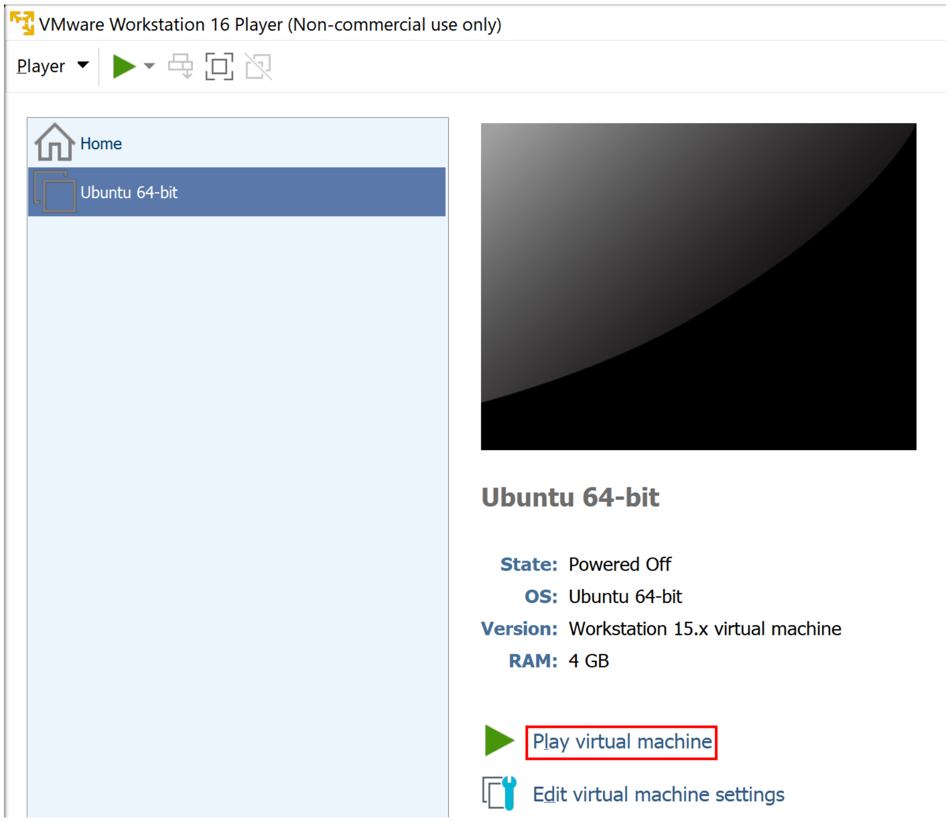
Devices are things that connect to the internet directly or through a

Learn more


Device ID
my-device-kneron

Public key format
RS256_X509

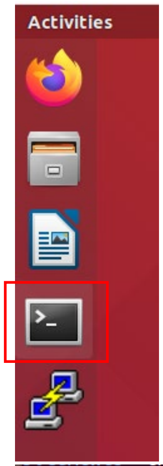
Open VMWare Workstation and play your virtual machine

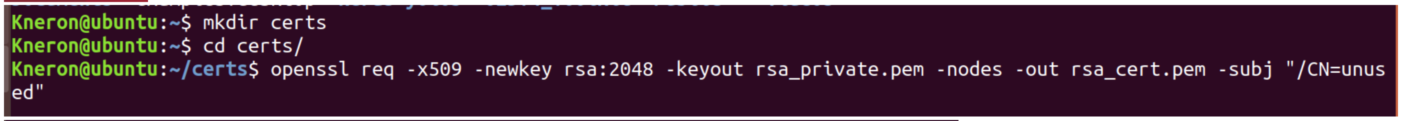

VMware Workstation 16 Player (Non-commercial use only)
Player
Home
Ubuntu 64-bit
Play virtual machine
Edit virtual machine settings

Ubuntu 64-bit

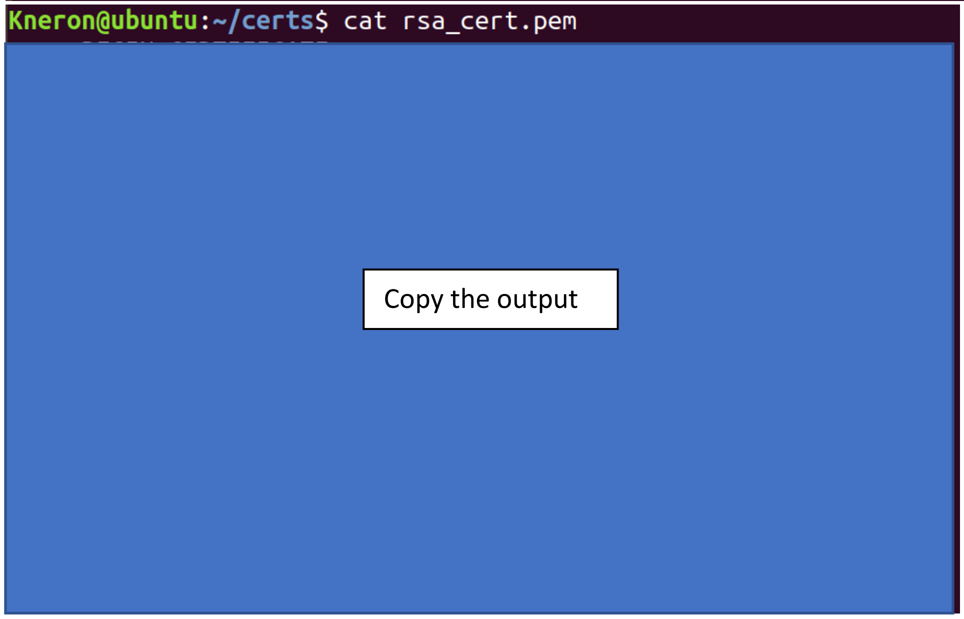
State: Powered Off
OS: Ubuntu 64-bit
Version: Workstation 15.x virtual machine
RAM: 4 GB

Open the terminal





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

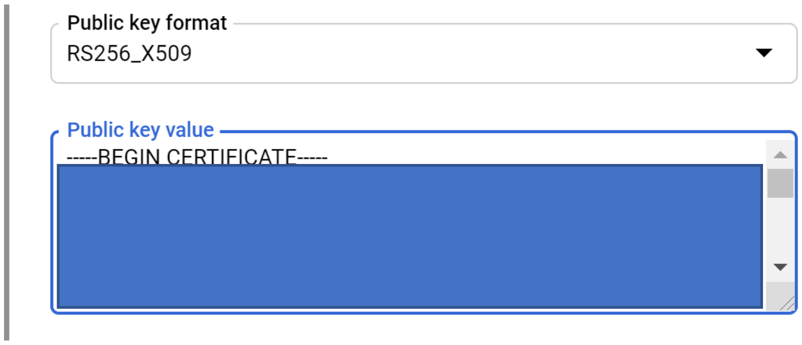


```
Kneron@ubuntu:~/certs$ cat rsa_cert.pem
```

[redacted]

Copy the output

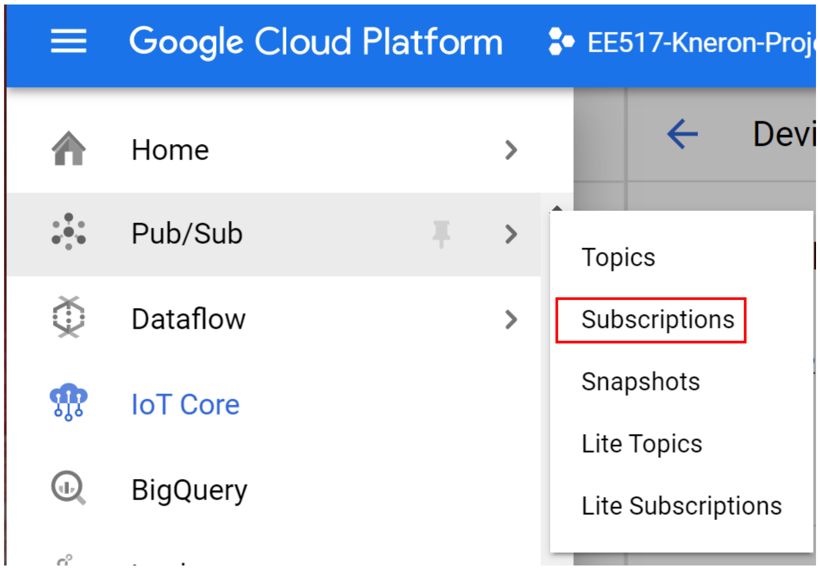
Copy and paste the key then click on Create

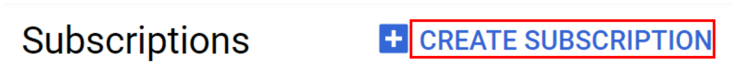


Public key format
RS256_X509

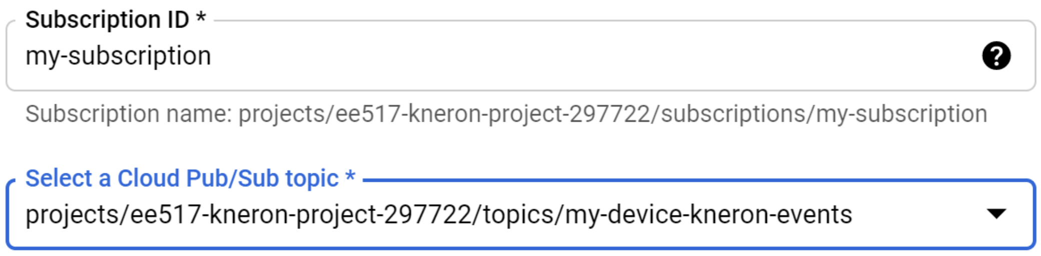
Public key value
-----BEGIN CERTIFICATE-----

[redacted]





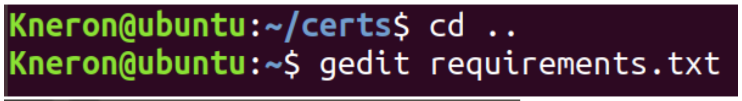
Add subscription ID then click on Create



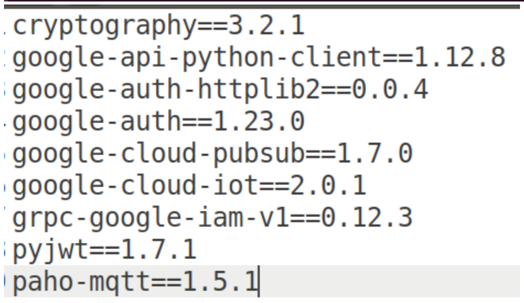
Subscription ID *
my-subscription
Subscription name: projects/ee517-kneron-project-297722/subscriptions/my-subscription

Select a Cloud Pub/Sub topic *
projects/ee517-kneron-project-297722/topics/my-device-kneron-events

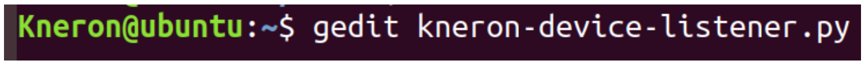
Go back to the terminal and create a script that listens to the cloud



```
Kneron@ubuntu:~/certs$ cd ..
Kneron@ubuntu:~$ gedit requirements.txt
```



```
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
```

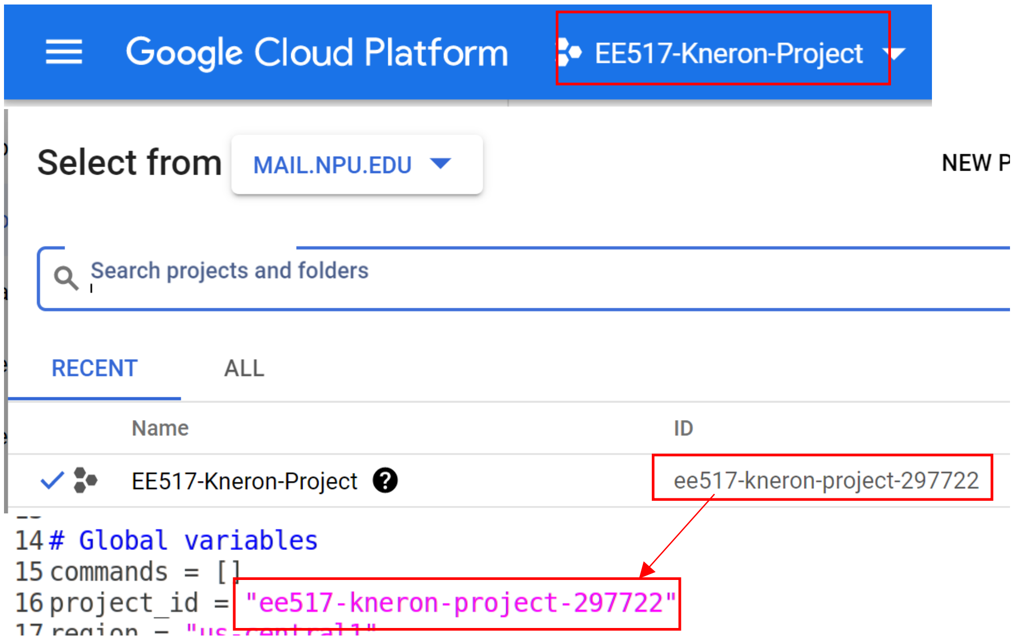


```
Kneron@ubuntu:~$ gedit kneron-device-listener.py
```

```
#!/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-297722"
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 = "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)

# initializes 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="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
   #print(command)
   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()
```

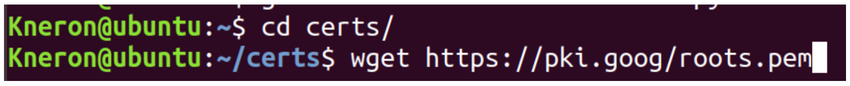
Make sure the project_id is correct



Name	ID
✓ EE517-Kneron-Project	ee517-kneron-project-297722

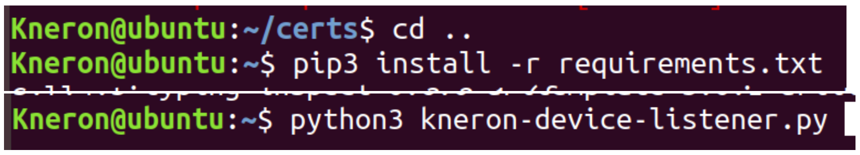
```
# Global variables
commands = []
project_id = "ee517-kneron-project-297722"
region = "us-central1"
```

Download Google root cert



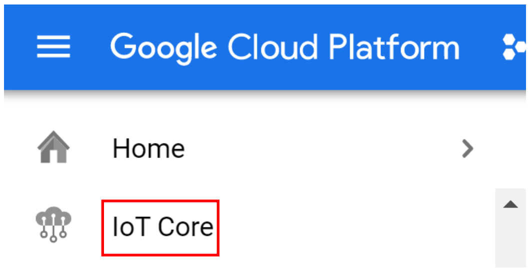
```
Kneron@ubuntu:~$ cd certs/
Kneron@ubuntu:~/certs$ wget https://pki.goog/roots.pem
```

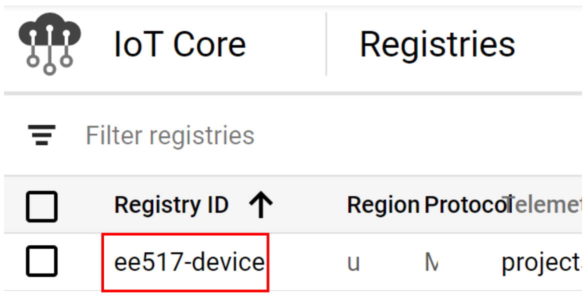
Install requirements and start to listen for command from GCP



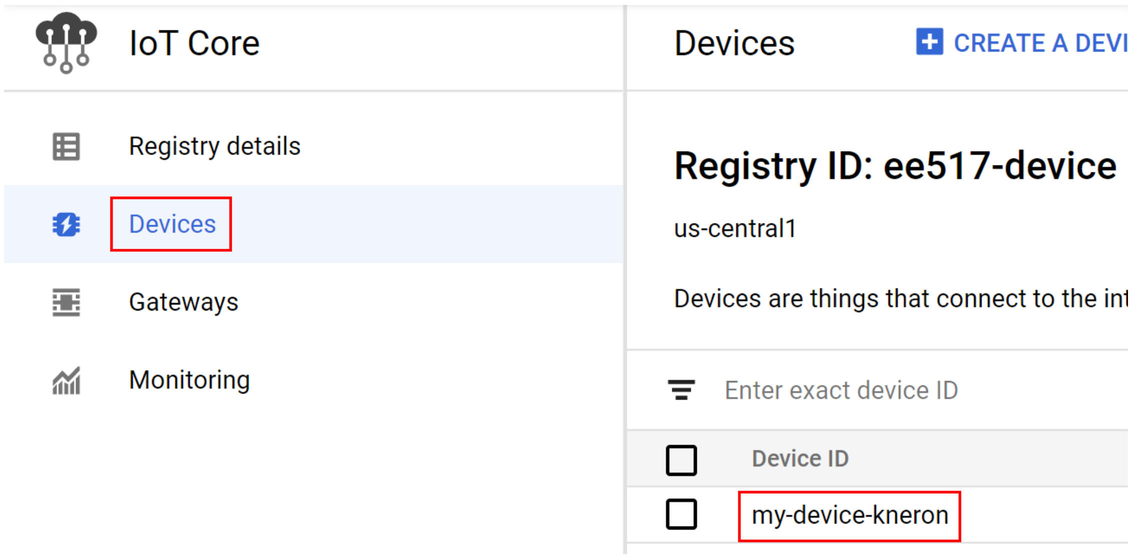
```
Kneron@ubuntu:~/certs$ cd ..
Kneron@ubuntu:~$ pip3 install -r requirements.txt
Kneron@ubuntu:~$ python3 kneron-device-listener.py
```

Go to GCP to send a command





Registry ID ↑	Region	Protocol	Teleme
ee517-device	u	M	project

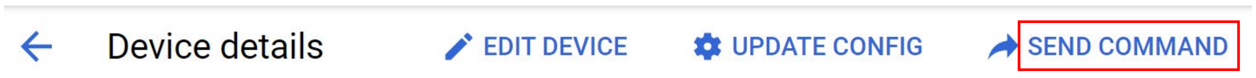


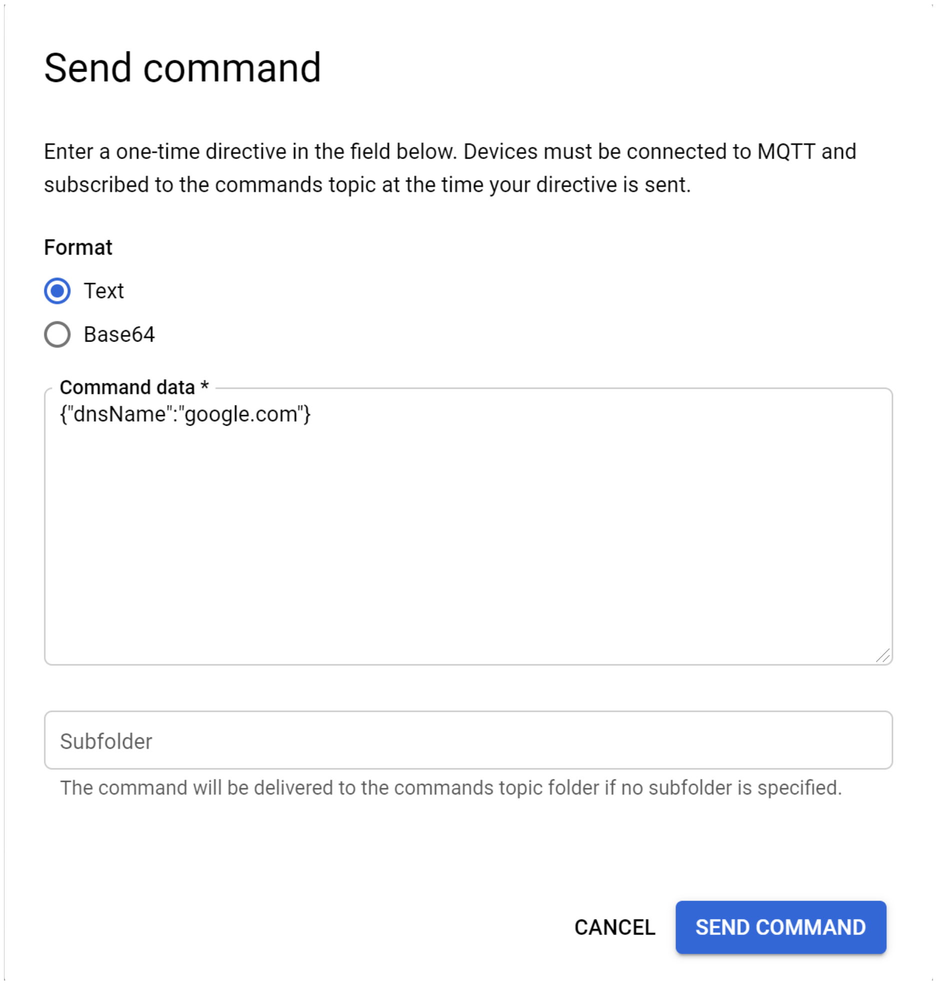
Registry ID: ee517-device

us-central1

Devices are things that connect to the int

Device ID
my-device-kneron





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

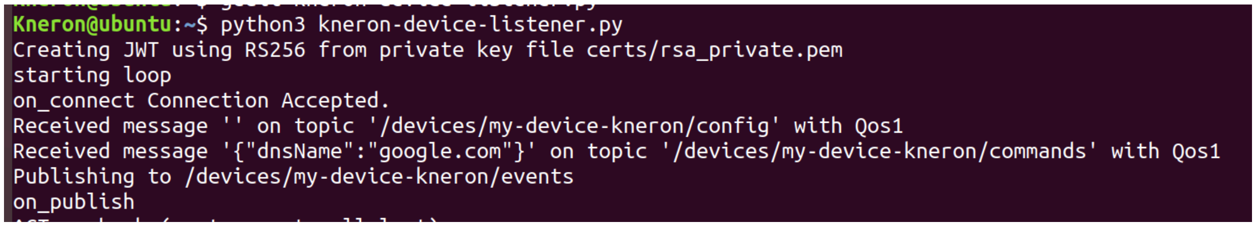
[✓ Text] Base64

Command data *
{"dnsName":"google.com"}

Subfolder

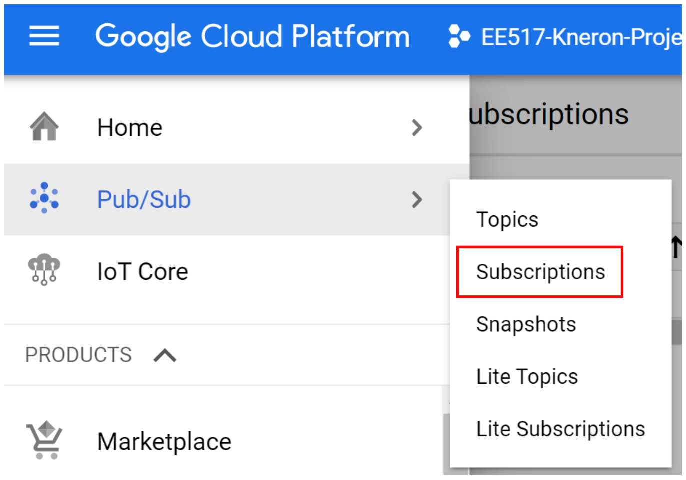
The command will be delivered to the commands topic folder if no subfolder is specified.

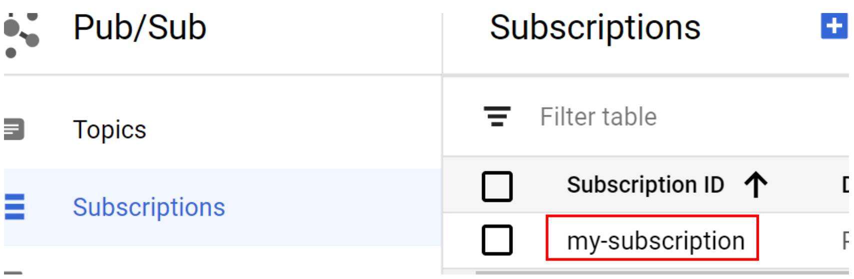
Go to the terminal and it should say on_publish

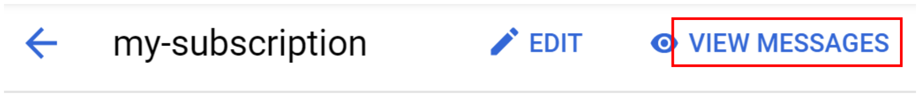


```
Kneron@ubuntu:~$ python3 kneron-device-listener.py
Creating JWT using RS256 from private key file certs/rsa_private.pem
starting loop
on_connect Connection Accepted.
Received message '' on topic '/devices/my-device-kneron/config' with Qos1
Received message '{"dnsName":"google.com"}' on topic '/devices/my-device-kneron/commands' with Qos1
Publishing to /devices/my-device-kneron/events
on_publish
```

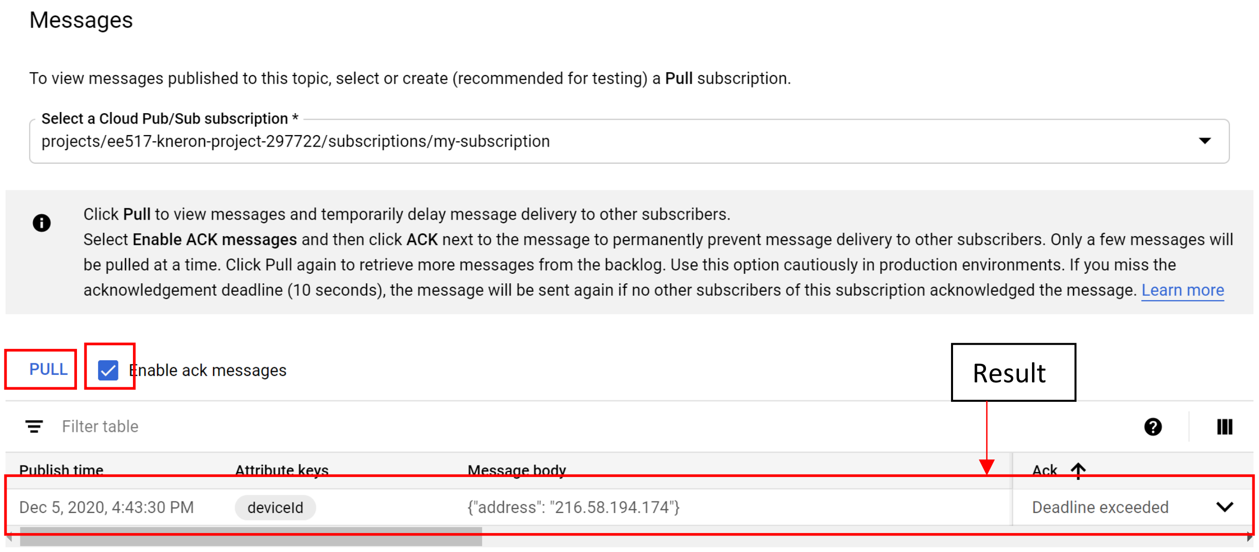
Go to Pub/Sub to view the message







Wait a few minutes then pull the message


Messages

To view messages published to this topic, select or create (recommended for testing) a **Pull** subscription.

Select a Cloud Pub/Sub subscription *
projects/ee517-kneron-project-297722/subscriptions/my-subscription

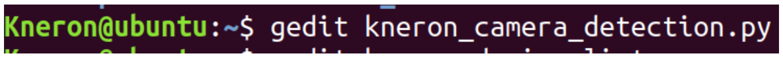
Click **Pull** to view messages and temporarily delay message delivery to other subscribers.
Select **Enable ACK messages** and then click **ACK** next to the message to permanently prevent message delivery to other subscribers. Only a few messages will be pulled at a time. Click Pull again to retrieve 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

Result

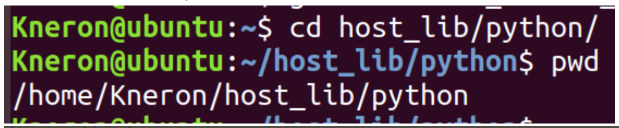
Publish time	Attribute keys	Message body	Ack
Dec 5, 2020, 4:43:30 PM	deviceId	{"address": "216.58.194.174"}	Deadline exceeded

To detect camera from Kneron using GCP

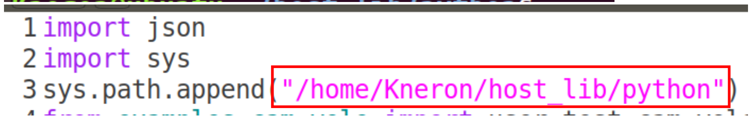

```
Kneron@ubuntu:~$ gedit kneron_camera_detection.py
```

```
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)
```

Make sure the path is correct

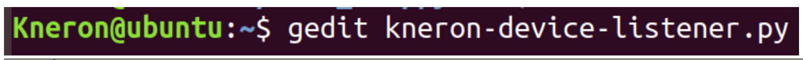


```
Kneron@ubuntu:~$ cd host_lib/python/
Kneron@ubuntu:~/host_lib/python$ pwd
/home/Kneron/host_lib/python
```

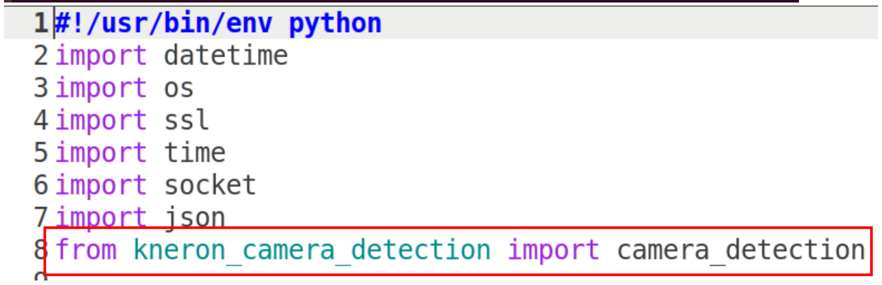


```
import json
import sys
sys.path.append("/home/Kneron/host_lib/python")
```

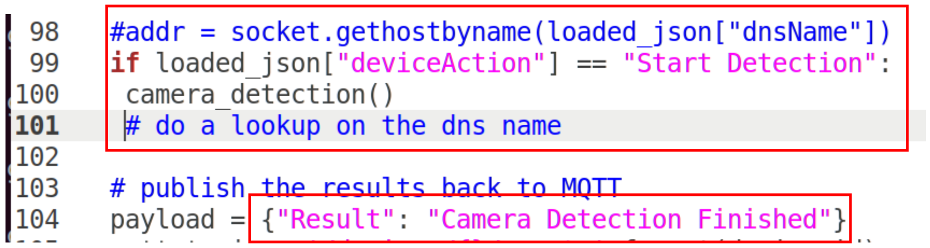
Edit device listener



```
Kneron@ubuntu:~$ gedit kneron-device-listener.py
```

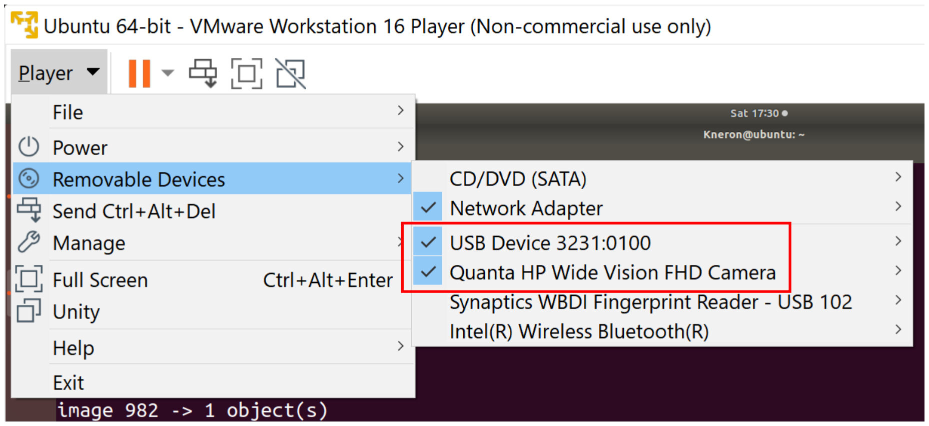


```
#!/usr/bin/env python
import datetime
import os
import ssl
import time
import socket
import json
from kneron_camera_detection import camera_detection
```



```
#addr = socket.gethostbyname(loaded_json["dnsName"])
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"}
```

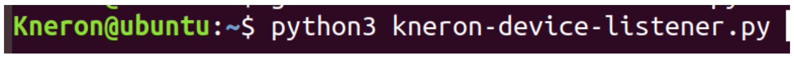
Make sure the Kneron USB and camera is connected to the virtual machine



Ubuntu 64-bit - VMware Workstation 16 Player (Non-commercial use only)
Player
File
Power
Removable Devices
Send Ctrl+Alt+Del
Manage
Full Screen Ctrl+Alt+Enter
Unity
Help
Exit
CD/DVD (SATA)
Network Adapter
USB Device 3231:0100
Quanta HP Wide Vision FHD Camera
Synaptics WBDI Fingerprint Reader - USB 102
Intel(R) Wireless Bluetooth(R)
Sat 17:30
Kneron@ubuntu: ~

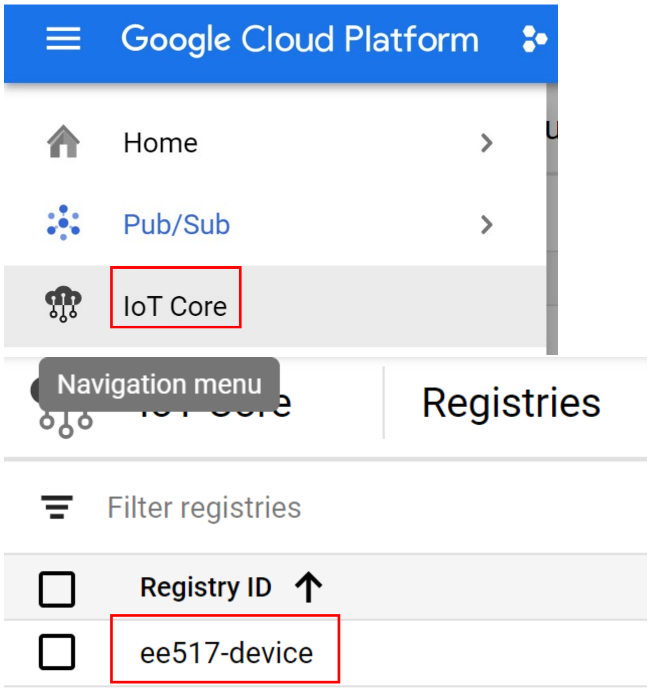
```
image 982 -> 1 object(s)
```

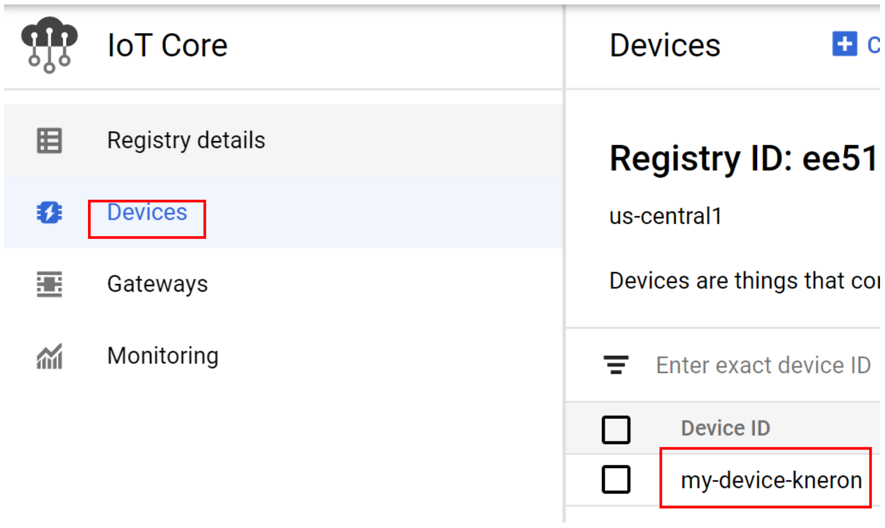
Start the device listener

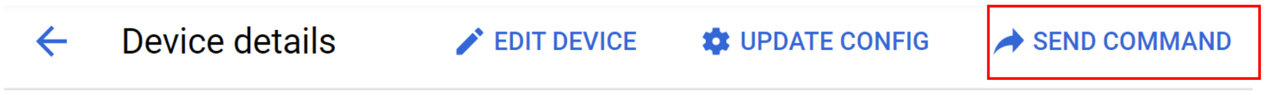


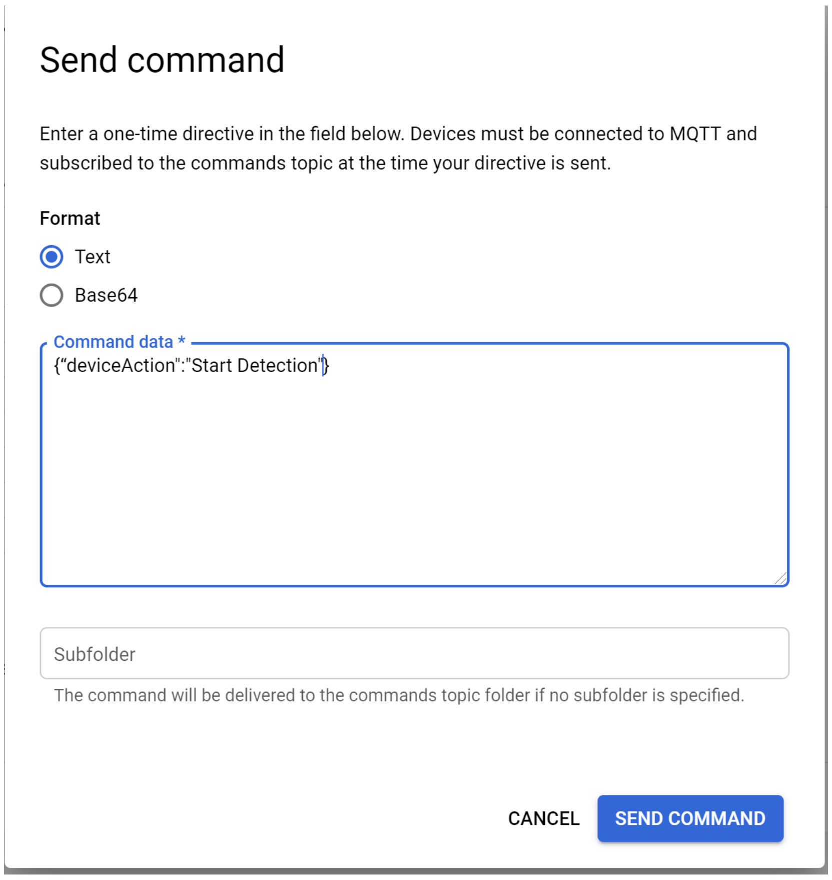
```
Kneron@ubuntu:~$ python3 kneron-device-listener.py
```

Send a command in GCP









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

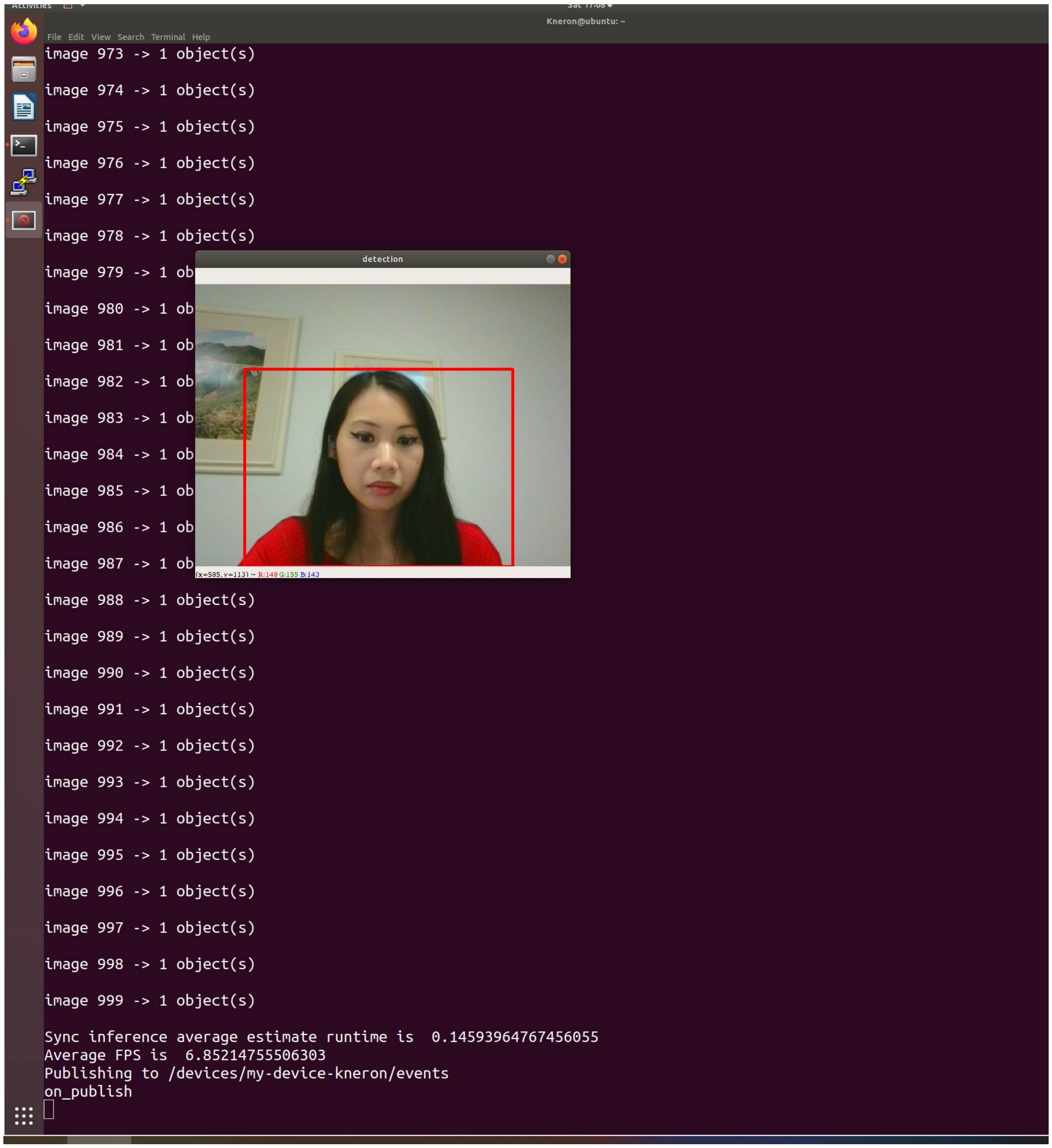
Command data *

```
{"deviceAction":"Start Detection"}
```

Subfolder

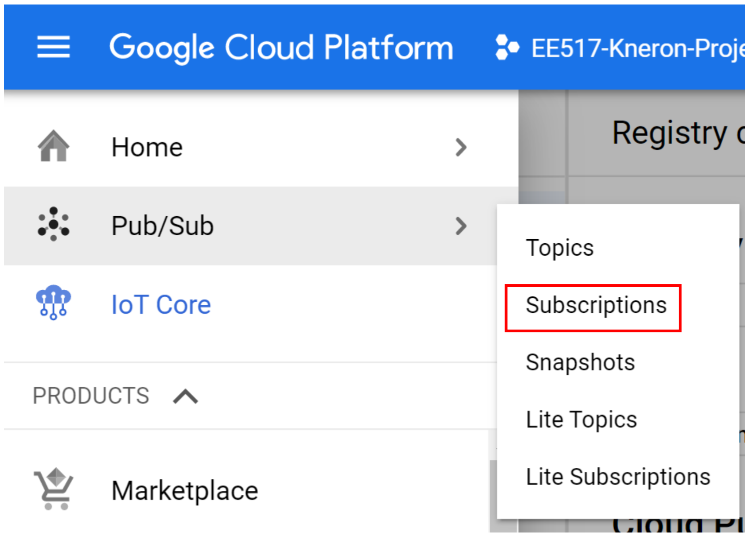
The command will be delivered to the commands topic folder if no subfolder is specified.

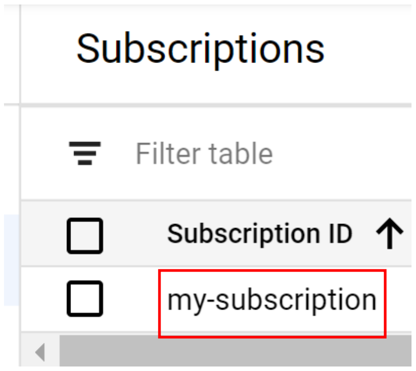
Result in the terminal

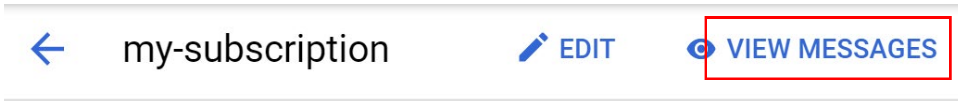

Kneron@ubuntu: ~
File Edit View Search Terminal Help
detection
(x=585, y=113) ~ R:148 G:155 B:143

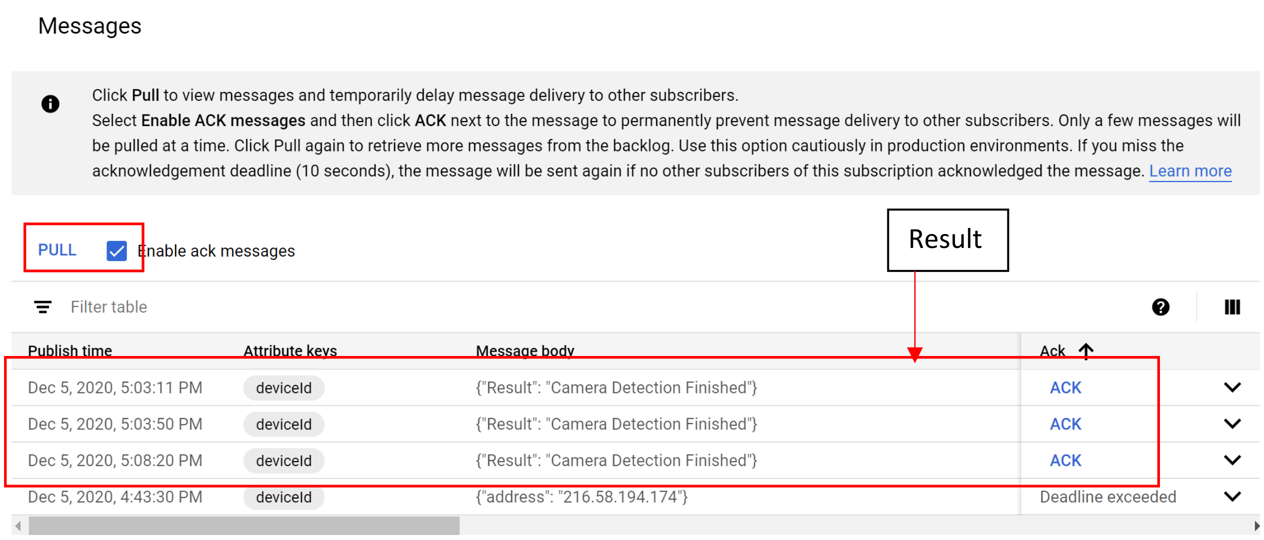
```
image 973 -> 1 object(s)

image 974 -> 1 object(s)

image 975 -> 1 object(s)

image 976 -> 1 object(s)

image 977 -> 1 object(s)

image 978 -> 1 object(s)

image 979 -> 1 ob

image 980 -> 1 ob

image 981 -> 1 ob

image 982 -> 1 ob

image 983 -> 1 ob

image 984 -> 1 ob

image 985 -> 1 ob

image 986 -> 1 ob

image 987 -> 1 ob

image 988 -> 1 object(s)

image 989 -> 1 object(s)

image 990 -> 1 object(s)

image 991 -> 1 object(s)

image 992 -> 1 object(s)

image 993 -> 1 object(s)

image 994 -> 1 object(s)

image 995 -> 1 object(s)

image 996 -> 1 object(s)

image 997 -> 1 object(s)

image 998 -> 1 object(s)

image 999 -> 1 object(s)

Sync inference average estimate runtime is  0.14593964767456055
Average FPS is  6.85214755506303
Publishing to /devices/my-device-kneron/events
on_publish
```

View the message after a few minutes








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

Publish time	Attribute keys	Message body	Ack
Dec 5, 2020, 5:03:11 PM	deviceId	{"Result": "Camera Detection Finished"}	ACK
Dec 5, 2020, 5:03:50 PM	deviceId	{"Result": "Camera Detection Finished"}	ACK
Dec 5, 2020, 5:08:20 PM	deviceId	{"Result": "Camera Detection Finished"}	ACK
Dec 5, 2020, 4:43:30 PM	deviceId	{"address": "216.58.194.174"}	Deadline exceeded