

Open Google Cloud Platform Console and create a new project

The screenshot shows the Google Cloud Platform console interface. At the top, the header bar is blue with the Google Cloud Platform logo, a dropdown menu showing 'My First Project', and several icons including a search icon, a mail icon, a help icon, a notification icon with a green circle containing the number '1', and a user profile icon. Below the header, there are tabs for 'DASHBOARD', 'ACTIVITY', and 'RECOMMENDATIONS', along with a 'CUSTOMIZE' link. The main content area is titled 'Select from' with a dropdown menu showing 'MAIL.NPU.EDU'. To the right of this is a 'NEW PROJECT' button. Below this, there is a 'Project name *' field with the text 'EE517-Kneron-Project' and a help icon. Below the project name field, it says 'Project ID: ee517-kneron-project-297722. It cannot be changed later. EDIT'. Below that is an 'Organization *' dropdown menu showing 'mail.npu.edu' and a help icon. Below the organization dropdown, it says 'Select an organization to attach it to a project. This selection can't be changed later.' Below that is a 'Location *' dropdown menu showing 'mail.npu.edu' and a 'BROWSE' button. Below the location dropdown, it says 'Parent organization or folder'. At the bottom, there are two buttons: 'CREATE' and 'CANCEL'.

Google Cloud Platform My First Project

DASHBOARD ACTIVITY RECOMMENDATIONS CUSTOMIZE

Select from MAIL.NPU.EDU 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 BROWSE

Parent organization or folder

CREATE CANCEL

Make sure you are working on your recently created project

The screenshot shows the Google Cloud Platform console interface. At the top, the header bar is blue with the Google Cloud Platform logo, a dropdown menu showing 'EE517-Kneron-Project', and several icons including a search icon, a mail icon, a help icon, a notification icon, and a user profile icon.

Google Cloud Platform EE517-Kneron-Project

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

The screenshot shows the Google Cloud Platform console interface. At the top, the header bar is blue with the Google Cloud Platform logo and a 'Search' input field. On the left side, there is a sidebar with a home icon and the text 'Home', and a cloud icon with the text 'IoT Core' which is highlighted with a red box.

Google Cloud Platform Search

Home

IoT Core



Google Cloud IoT API

Google

Registers and manages IoT (Internet Google Cloud Platform).

ENABLE

[TRY THIS API](#)

Create registry

Google Cloud Platform EE517-Kneron-Project 1

IoT Core | Registries **+ CREATE REGISTRY** [HIDE INFO PANEL](#)

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

 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 [Learn more](#)


Device ID
my-device-kneron


Public key format
RS256_X509

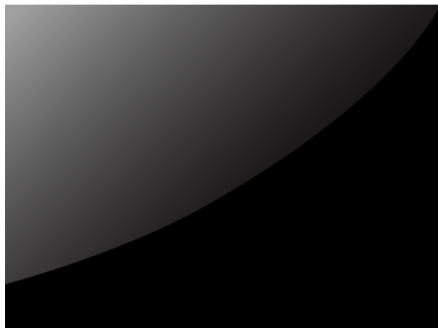
Open VMWare Workstation and play your virtual machine



VMware Workstation 16 Player (Non-commercial use only)

Player    

 Home

 Ubuntu 64-bit



Ubuntu 64-bit
State: Powered Off
OS: Ubuntu 64-bit
Version: Workstation 15.x virtual machine
RAM: 4 GB
 [Play virtual machine](#)
 [Edit virtual machine settings](#)

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=unused"
```

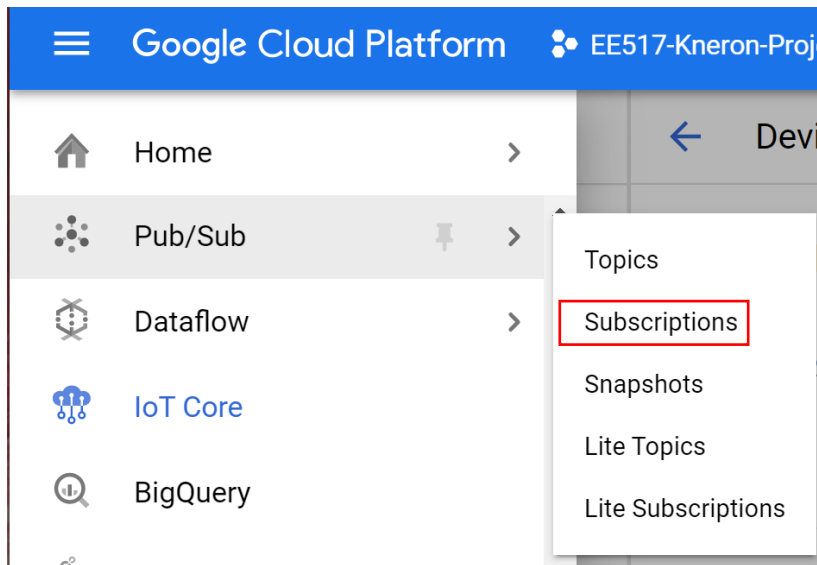
```
Kneron@ubuntu:~/certs$ cat rsa_cert.pem
```

Copy the output

Copy and paste the key then click on Create

Public key format
RS256_X509

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



Subscriptions

[+ CREATE SUBSCRIPTION](#)

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-httpplib2==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

The screenshot shows the Google Cloud Platform interface. At the top, the 'EE517-Kneron-Project' is selected. Below, in the 'Select from' section, the project 'EE517-Kneron-Project' is listed with its ID 'ee517-kneron-project-297722'. A red box highlights the ID, and an arrow points to it in the code snippet below.

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

```

14 # Global variables
15 commands = []
16 project_id = "ee517-kneron-project-297722"
17 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

The screenshot shows the Google Cloud Platform IoT Core console. At the top, the 'Google Cloud Platform' header is visible. Below it, the 'IoT Core' menu is highlighted. The main content area is divided into two sections: 'IoT Core' and 'Registries'. In the 'Registries' section, a table lists registries with columns for 'Registry ID', 'Region', 'Protocol', and 'Element'. The registry 'ee517-device' is highlighted. Below this, the 'Devices' section is shown, with a 'CREATE A DEVICE' button. The 'Devices' section displays the 'Registry ID: ee517-device' and the region 'us-central1'. A search bar for 'Enter exact device ID' is present, and a table lists devices with a 'Device ID' column. The device 'my-device-kneron' is highlighted. At the bottom, a navigation bar includes links for 'Device details', 'EDIT DEVICE', 'UPDATE CONFIG', and 'SEND COMMAND', with the 'SEND COMMAND' link highlighted.

Google Cloud Platform

Home > IoT Core

IoT Core Registries

Filter registries

Registry ID	Region	Protocol	Element
ee517-device	us-central1	MQTT	project

IoT Core

Devices [+ CREATE A DEVICE](#)

Registry details

Devices

Gateways

Monitoring

Registry ID: ee517-device

us-central1

Devices are things that connect to the internet

Enter exact device ID

Device ID
my-device-kneron

Device details EDIT DEVICE UPDATE CONFIG SEND COMMAND

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.

CANCEL

SEND COMMAND

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

The screenshot shows the Google Cloud Platform interface. In the top navigation bar, the project is 'EE517-Kneron-Proje'. The left sidebar has a menu with 'Home', 'Pub/Sub', 'IoT Core', and 'Marketplace'. A dropdown menu from 'Pub/Sub' shows 'Topics', 'Subscriptions' (highlighted with a red box), 'Snapshots', 'Lite Topics', and 'Lite Subscriptions'. Below this, the 'Subscriptions' page is shown with a table containing one entry: 'my-subscription' (highlighted with a red box). At the bottom of the page, there are buttons for 'EDIT' and 'VIEW MESSAGES' (highlighted with a red box).

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

i 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](#)

The screenshot shows the 'my-subscription' message view. At the top, there is a 'PULL' button (highlighted with a red box) and a checkbox for 'Enable ack messages' (checked and highlighted with a red box). Below this is a table of messages. The first row is highlighted with a red box and contains the following data:

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

A red arrow points from a box labeled 'Result' to the 'Deadline exceeded' status in the table.

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

```

```

1 import json
2 import sys
3 sys.path.append("/home/Kneron/host_lib/python")

```

Edit device listener

```

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

```

```

1#!/usr/bin/env python
2import datetime
3import os
4import ssl
5import time
6import socket
7import json
8from kneron_camera_detection import camera_detection

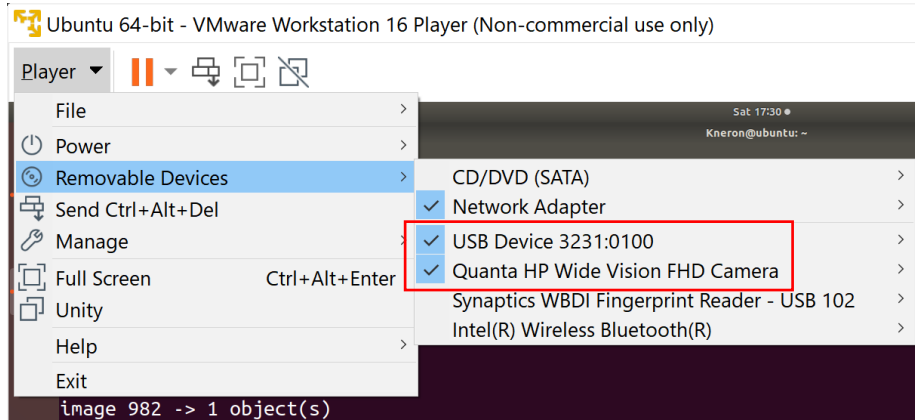
```

```

98 #addr = socket.gethostbyname(loaded_json["dnsName"])
99 if loaded_json["deviceAction"] == "Start Detection":
100     camera_detection()
101     # do a lookup on the dns name
102
103 # publish the results back to MQTT
104 payload = {"Result": "Camera Detection Finished"}

```

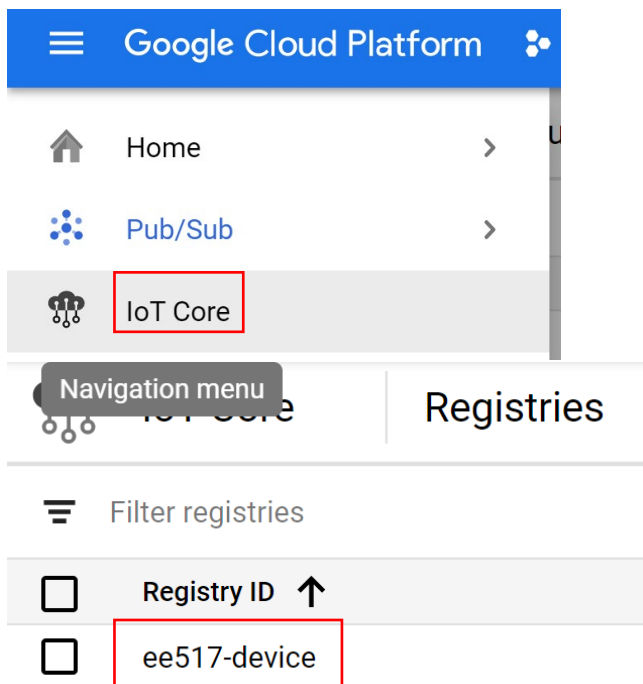
Make sure the Kneron USB and camera is connected to the virtual machine




Start the device listener

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

Send a command in GCP



 IoT Core

Registry details

Devices

Gateways

Monitoring

Devices

Registry ID: ee51

us-central1

Devices are things that cor

Enter exact device ID

Device ID

my-device-kneron

← Device details

EDIT DEVICE

UPDATE CONFIG

SEND COMMAND

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" }

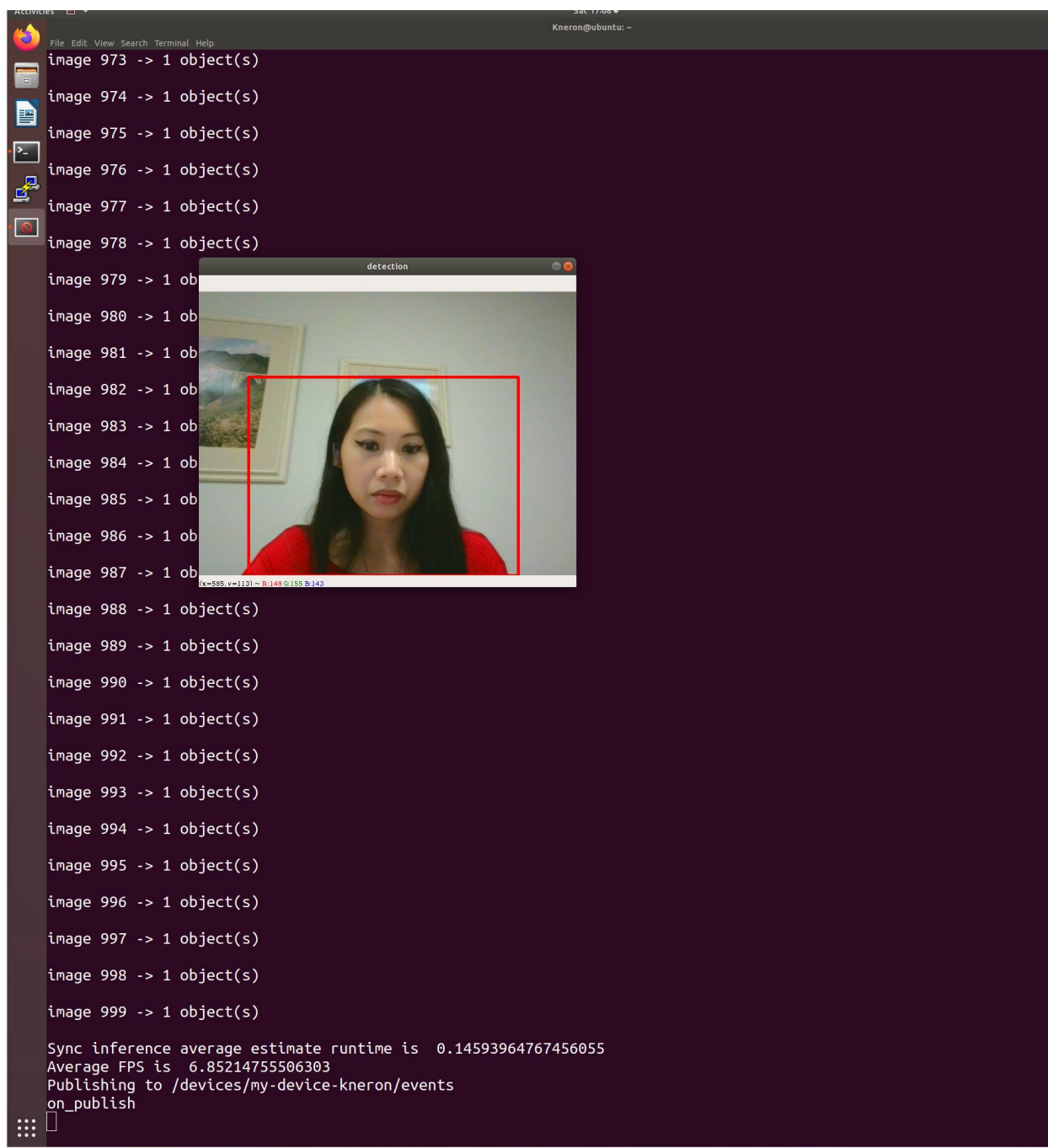
Subfolder

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

CANCEL

SEND COMMAND

Result in the terminal



View the message after a few minutes

☰

Google Cloud Platform

EE517-Kneron-Proje

Home

Pub/Sub

IoT Core

PRODUCTS

Marketplace

Registry c

Topics

Subscriptions

Snapshots

Lite Topics

Lite Subscriptions

Subscriptions

Filter table

☐

Subscription ID ↑

☐

my-subscription

← my-subscription

EDIT

VIEW MESSAGES

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

Result

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