

INSTRUCTIVO AWS X EMQX

1. Creación de Instancia en AWS

En primer lugar, debemos de iniciar sesión en nuestra cuenta de AWS. Después, en el buscador de servicios que provee la plataforma, escribimos EC2, Este servicio nos permite crear instancias desde el cual podemos alojar desde nuestro Front-end, Back-end hasta LLM. Ahora, damos clic en el botón “Lanzar la instancia” y nos aparecerá la siguiente ventana.

The screenshot shows the AWS Management Console 'Launch an instance' page. The left pane has sections for 'Nombre y etiquetas', 'Imágenes de aplicaciones y sistemas operativos', and a search bar. The right pane shows a 'Resumen' (Summary) with details like 'Número de instancias' (1), 'Imagen de software (AMI)' (Amazon Linux 2023), 'Tipo de servidor virtual' (t2.micro), 'Firewall' (New security group), and 'Almacenamiento' (1 8 GiB volume). At the bottom right are 'Cancelar' and 'Lanzar instancia' buttons.

Aquí seleccionaremos Ubuntu y el tipo de instancia (en nuestro caso la capa gratuita). Luego, crearemos un nuevo par de claves, para ello haremos clic en dicha opción, digitaremos el Nombre del par de claves, el tipo y el formato. Para este ejemplo, seleccionaremos “RSA” como tipo y “pem” como formato.

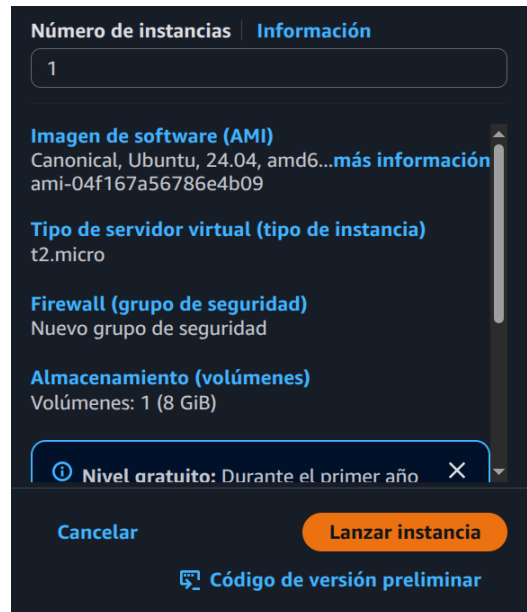
The screenshot shows the 'Create key pair' dialog box. It includes a text input for 'Nombre del par de claves' (labeled 'llave_prueba'), a section for 'Tipo de par de claves' with 'RSA' selected, and a section for 'Formato de archivo de clave privada' with '.pem' selected. A warning message at the bottom states that the private key must be stored securely. At the bottom are 'Cancelar' and 'Crear par de claves' buttons.

Una vez creado el nuevo par de claves, se descargará un archivo con la extensión .pem (u otra previamente seleccionada). A continuación, será necesario seleccionarlo desde el menú desplegable. Asimismo, se deberá configurar la red en donde marcaran las siguientes opciones:

- Permitir tráfico de SSH desde Cualquier lugar en nuestro caso. Si se quiere que solamente se pueda acceder a la instancia desde una IP o varias en específica se debe seleccionar alguna de las otras opciones

- Permitir el tráfico de HTTP y HTTPS

Con todo lo anterior hecho, solamente nos queda hacer clic en el botón “Lanzar instancia” y esperar unos minutos mientras se crea,



2. Despliegue

Para este ejemplo, se desplegará un back-end creado en FastApi, el cual permite detectar personas mediante un modelo entrenado en MobileNet SSD.

Link del repositorio: https://github.com/Fullops/Api_Rest_Detection_Raspberry.git

Teniendo en cuenta lo anterior, deberemos de conectarnos a la instancia que previamente habíamos creado. Para ello, ingresaremos a EC2 > Instancias y seleccionaremos la que se había creado y daremos clic en el botón “Conectar”.

A partir de este punto, todo el proceso se debe realizar desde una consola. Se recomienda utilizar **Git Bash** en sistemas Windows o la terminal predeterminada en sistemas Linux. Ahora, abriremos una nueva consola, nos ubicaremos en la misma carpeta donde se encuentra el archivo que se había descargado y ejecutaremos los siguientes comando:

```
Chmod 400 "<nombre de la llave>.pem"
ssh -i "<nombre de la llave>.pem" ubuntu@<DNS publica que provee Amazon>
```

Con esto ya deberíamos estar conectados a la instancia. Para validar en la terminal debiera decir ubuntu@<ip de la instancia>

```
[fullops@fullops Descargas]$ chmod 400 "llave_prueba.pem"
[fullops@fullops Descargas]$ ssh -i "llave_prueba.pem" ubuntu@ec2-3-145-153-44.us-east-2.compute.amazonaws.com
The authenticity of host 'ec2-3-145-153-44.us-east-2.compute.amazonaws.com (3.145.153.44)' can't be established.
ED25519 key fingerprint is SHA256:DFcbGjCrg42bo/W3uZFcNpX7VgJNdir7TbkjH8msRgg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-145-153-44.us-east-2.compute.amazonaws.com' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-1024-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Thu May 29 01:04:15 UTC 2025

System load:  0.0                Processes:      104
Usage of /:   24.9% of 6.71GB    Users logged in: 0
Memory usage: 20%                IPv4 address for enX0: 172.31.6.74
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-6-74:~$
```

A continuación, actualizarán los paquetes de Ubuntu y se instalará python3 y Nginx utilizando los siguientes comandos:

```
sudo apt-get update
sudo apt install python3 python3-pip python3-venv
sudo apt install nginx
sudo apt install libgl1
```

Una vez hecho lo anterior, se clonará el repositorio de github, se creará el entorno virtual e instalarán las respectivas librerías. Se recomienda ejecutar previamente el proyecto que hemos creado para verificar si no faltan algunas librerías o existe alguna incompatibilidad entre versiones de python.

Luego, creamos un archivo con “sudo nano /etc/nginx/sites-available/fastapi” y pegamos lo siguiente:

```
server {
    listen 80;
    server_name <Dirección IPv4 pública>;

    location / {
        proxy_pass http://127.0.0.1:<puerto en donde corre el servidor>;
        proxy_set_header Host $host;
```

```

    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
}

```

Ahora, simplemente tendremos que guardar con Ctrl+O+Enter, y nos saldremos de nano con Ctrl+x.

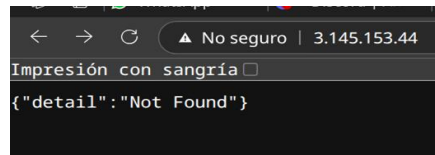
A su vez, tendremos que activar el sitio y comprobar si el sitio se ejecuta.

```

sudo ln -s /etc/nginx/sites-available/fastapi /etc/nginx/sites-enabled
sudo nginx -t
sudo systemctl restart nginx

```

A continuación, deberemos ejecutar el proyecto que habíamos clonado y con ello deberíamos ver lo siguiente, si ingresamos a “http:<Dirección IPv4 pública>”:



Para finalizar, nos quedaría crear un servicio que permita que el proyecto se ejecute como un proceso del sistema. Con ello, tendremos que ejecutar “sudo nano /etc/systemd/system/fastapi.service” y pegar lo siguiente:

```

[Unit]
Description=FastAPI Uvicorn App
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/<nombre de la carpeta o repositorio clonado>
ExecStart=/home/ubuntu/<nombre de la carpeta o repositorio clonado>/env/bin/uvicorn app.main:app
--port 8000
Restart=always

[Install]
WantedBy=multi-user.target

```

Ahora debemos guardar y recargar los servicios con “sudo systemctl daemon-reexec” y “sudo systemctl daemon-reload”. Como último paso, faltaría iniciar y habilitar el servicio con los comandos “sudo systemctl start fastapi” y “sudo systemctl enable fastapi”.

Nota: Si queremos verificar el estado del servicio que anteriormente creamos, solamente tenemos de ingresar el comando “sudo systemctl status fastapi”.

3. Instalación y configuración de Mosquitto (local)

En primer lugar, se debe instalar **Mosquitto**, un broker MQTT que permite gestionar publicaciones y suscripciones a tópicos. Para ello, se deben ejecutar los siguientes comandos:

```
sudo apt install mosquitto
sudo apt install mosquitto-clients
```

Una vez instalado, es necesario habilitar el servicio para que se inicie automáticamente al encender el sistema. Esto se logra con los siguientes comandos:

```
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

Para comprobar que el servicio está activo y funcionando correctamente, se recomienda verificar su estado con:

```
sudo systemctl status mosquitto
```

```
ubuntu@ip-172-31-5-37:~$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/usr/lib/systemd/system/mosquitto.service; enabled; preset: enabled)
   Active: active (running) since Fri 2025-05-30 01:45:36 UTC; 4min 7s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Main PID: 19213 (mosquitto)
    Tasks: 1 (limit: 1129)
   Memory: 1.0M (peak: 1.4M)
      CPU: 125ms
   CGroup: /system.slice/mosquitto.service
           └─19213 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

May 30 01:45:36 ip-172-31-5-37 systemd[1]: Starting mosquitto.service - Mosquitto MQTT Broker...
May 30 01:45:36 ip-172-31-5-37 systemd[1]: Started mosquitto.service - Mosquitto MQTT Broker.
```

A continuación, se debe editar la configuración por defecto de Mosquitto para habilitar conexiones anónimas y establecer parámetros básicos de registro. Para ello, se debe editar el archivo principal de configuración con:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

A continuación, se deben pegar las siguientes líneas al final del archivo:

```
listener 1883
allow_anonymous true
log_type all
log_dest stdout
include_dir /etc/mosquitto/conf.d
```

Una vez realizados los cambios, se debe reiniciar el servicio para que la nueva configuración entre en efecto:

```
sudo systemctl restart mosquitto
sudo systemctl status mosquitto
```

```

ubuntu@ip-172-31-5-37:~$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/usr/lib/systemd/system/mosquitto.service; enabled; preset: enabled)
   Active: active (running) since Fri 2025-05-30 01:52:36 UTC; 4s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
   Process: 19581 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
   Process: 19582 ExecStartPre=/bin/chown mosquitto:mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
   Process: 19585 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
   Process: 19587 ExecStartPre=/bin/chown mosquitto:mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
  Main PID: 19589 (mosquitto)
    Tasks: 1 (limit: 1129)
   Memory: 1.0M (peak: 1.4M)
      CPU: 16ms
   CGroup: /system.slice/mosquitto.service
           └─19589 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

May 30 01:52:36 ip-172-31-5-37 systemd[1]: Starting mosquitto.service - Mosquitto MQTT Broker...
May 30 01:52:36 ip-172-31-5-37 mosquitto[19589]: 1748569956: mosquitto version 2.0.18 starting
May 30 01:52:36 ip-172-31-5-37 mosquitto[19589]: 1748569956: Config loaded from /etc/mosquitto/mosquitto.conf.
May 30 01:52:36 ip-172-31-5-37 mosquitto[19589]: 1748569956: Opening ipv4 listen socket on port 1883.
May 30 01:52:36 ip-172-31-5-37 mosquitto[19589]: 1748569956: Opening ipv6 listen socket on port 1883.
May 30 01:52:36 ip-172-31-5-37 mosquitto[19589]: 1748569956: mosquitto version 2.0.18 running
May 30 01:52:36 ip-172-31-5-37 systemd[1]: Started mosquitto.service - Mosquitto MQTT Broker.

```

Para verificar que el broker funciona correctamente, se puede realizar una prueba de publicación y suscripción desde la misma máquina utilizando dos terminales.

En la primera terminal, se ejecuta el siguiente comando para suscribirse al tópico *prueba/topic*:

```
mosquitto_sub -h localhost -t prueba/topic
```

En la segunda terminal, se publica un mensaje en ese mismo tópico con el siguiente comando:

```
mosquitto_pub -h localhost -t prueba/topic -m "Hola MQTT"
```

Si todo está correctamente configurado, el mensaje "Hola MQTT" debería visualizarse en la primera terminal (la que está suscrita al tópico).

```

ubuntu@ip-172-31-5-37:~$ mosquitto_sub -h localhost -t prueba/topic
Hola MQTT

```

4. Creación de MQTT Cloud (EMQX)

Para crear un broker MQTT en la nube utilizando EMQX, lo primero que se debe hacer es crear una cuenta e iniciar sesión en la plataforma oficial. Una vez dentro, se crea un nuevo *Deployment* seleccionando el plan que mejor se ajuste a las necesidades del proyecto. En este caso, se opta por el plan Serverless. A continuación, se solicita un nombre para el *Deployment*, y tras ingresarlo, se hace clic en el botón Deploy para iniciar la creación del entorno.

Después de que el *Deployment* ha sido creado exitosamente, se accede a su panel de control. Desde allí, se debe ingresar a la sección Authentication, donde se creará un nuevo usuario. Para ello, simplemente se especifica un nombre de usuario y una contraseña, los cuales serán utilizados más adelante para autenticar conexiones al broker.

Una vez creado el usuario, nos iremos a "Authorization" y crearemos un nuevo Client ID. Para ello lo dejaremos tal cual como se muestra en la siguiente imagen.

Add Authorization

* Client ID

mosquito

* Permissions

Topic ?	Action	Permission	Add
deteccion/personas	Publis... ^ Publish Subscribe Publish & Subscribe	Allow v	Confirm

Posteriormente, ingresaremos a la opción “Data Integration”, en donde deberemos crear un conector. En la mayoría de los campos se puede conservar la configuración predeterminada, excepto en el campo URL, donde se debe ingresar la dirección a la que se desea que EMQX envíe los datos, correspondiente a un endpoint del backend que esté preparado para recibir eventos. Luego de ingresar la URL, se realiza una prueba haciendo clic en el botón Test, y si el resultado es exitoso, se continúa con la creación de una nueva regla seleccionando las opciones New y luego New Rule.

New Connector (HTTP Server)

* Connector Name

c-d755c7bc-0151d3

Note

* URL

http://3.145.153.44/api/mqtt/message

Headers

Key	Value	Add
keep-alive	timeout=5	
content-type	application/json	

Cancel Test New

En el editor de SQL que se presenta a continuación, se debe escribir la instrucción “SELECT payload FROM "deteccion/personas””. Se hace clic en Next para avanzar al siguiente paso. En la vista llamada New Action (Sink), se conserva la configuración por defecto, con excepción del campo Body, donde se debe ingresar el siguiente JSON:


```
{
  "payload": ${payload}
}
```

Para finalizar, solo tendremos que hacer clic en “Confirmar” y ya tendríamos todo lo necesario para continuar con el siguiente paso.

5. Conexión entre Mosquitto (local) y eMQX


Para establecer la conexión entre Mosquitto, instalado localmente en un equipo, y el servicio EMQX en la nube, es necesario descargar primero el certificado de seguridad generado por EMQX. Este certificado se encuentra en la sección Overview del panel del *Deployment*, específicamente en el apartado MQTT Connection Information.

MQTT Connection Information

Address: d755c7bc.ala.us-east-1.emqxsl.com 

MQTT over TLS/SSL Port: 8883

WebSocket over TLS/SSL Port: 8084

CA Certificate:  CA Certificate Expiration: 2031.11.09

Al hacer clic sobre el ícono de descarga, se obtendrá un archivo con la extensión “.crt”, el cual será utilizado para cifrar la comunicación entre ambos servicios.

Una vez descargado, el certificado debe ser copiado en la ruta “/etc/mosquitto/ca_certificates”. Si esta carpeta no existe en el sistema, será necesario crearla manualmente utilizando el comando “sudo mkdir -p /etc/mosquitto/ca_certificates”.

Con el certificado en su lugar, el siguiente paso consiste en crear un archivo de configuración llamado bridge.conf dentro del directorio “/etc/mosquitto/conf.d”. Si este directorio tampoco existe, se puede crear con el comando “sudo mkdir -p /etc/mosquitto/conf.d”.

Para editar o crear el archivo de configuración, se debe ejecutar el siguiente comando: “sudo nano /etc/mosquitto/conf.d/bridge.conf”. Dentro de este archivo, se debe pegar la siguiente configuración, reemplazando los valores <dirección EMQX del deployment>, <usuario creado en EMQX> y <contraseña del usuario creado en EMQX> por los datos correspondientes:

```
connection bridge_to_emqx
address d755c7bc.ala.us-east-1.emqxsl.com:8883
```

```
#Topics a los cuales vamos a subscribirnos (in) o publicar (out)
topic deteccion/personas out
topic deteccion/respuesta in
```

```
username user_test
password 1234
```

```
bridge_protocol_version mqttv311
bridge_cafile /etc/mosquitto/ca_certificates/emqxsl-ca.crt
bridge_insecure false
```

```
cleansession true
```


start_type automatic
try_private false
notifications false

Una vez finalizada la edición, se deben guardar los cambios y reiniciar Mosquitto con los siguientes comandos:

```
sudo systemctl restart mosquitto
sudo systemctl status mosquitto
```

Si todo salió bien, obtendremos lo siguiente:

```
ubuntu@ip-172-31-5-37:/etc/mosquitto$ sudo systemctl restart mosquitto
ubuntu@ip-172-31-5-37:/etc/mosquitto$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/usr/lib/systemd/system/mosquitto.service; enabled; preset: enabled)
   Active: active (running) since Fri 2025-05-30 03:41:11 UTC; 3s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Process: 20033 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 20036 ExecStartPre=/bin/chown mosquitto:mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 20038 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
  Process: 20040 ExecStartPre=/bin/chown mosquitto:mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
 Main PID: 20042 (mosquitto)
    Tasks: 1 (limit: 1129)
   Memory: 1.2M (peak: 1.5M)
      CPU: 19ms
   CGroup: /system.slice/mosquitto.service
           └─20042 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

May 30 03:41:11 ip-172-31-5-37 mosquitto[20042]: 1748576471: Warning: Error resolving bridge address: Temporary failure in name resolution.
May 30 03:41:11 ip-172-31-5-37 mosquitto[20042]: 1748576471: Warning: Error resolving bridge address: Name or service not known.
May 30 03:41:11 ip-172-31-5-37 mosquitto[20042]: 1748576471: mosquitto version 2.0.18 starting
May 30 03:41:11 ip-172-31-5-37 mosquitto[20042]: 1748576471: Config loaded from /etc/mosquitto/mosquitto.conf.
May 30 03:41:11 ip-172-31-5-37 mosquitto[20042]: 1748576471: Opening ipv4 listen socket on port 1883.
May 30 03:41:11 ip-172-31-5-37 mosquitto[20042]: 1748576471: Opening ipv6 listen socket on port 1883.
May 30 03:41:11 ip-172-31-5-37 mosquitto[20042]: 1748576471: Bridge local.ip-172-31-5-37.bridge_to_emqx doing local SUBSCRIBE on topic #
May 30 03:41:11 ip-172-31-5-37 mosquitto[20042]: 1748576471: Connecting bridge (step 1) bridge_to_emqx (<dirección:1883)
May 30 03:41:11 ip-172-31-5-37 mosquitto[20042]: 1748576471: mosquitto version 2.0.18 running
```

O también podremos obtener esto:

```
ubuntu@ip-172-31-5-37:~$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/usr/lib/systemd/system/mosquitto.service; enabled; preset: enabled)
   Active: active (running) since Fri 2025-05-30 03:41:11 UTC; 6min ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Process: 20033 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 20036 ExecStartPre=/bin/chown mosquitto:mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
  Process: 20038 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
  Process: 20040 ExecStartPre=/bin/chown mosquitto:mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
 Main PID: 20042 (mosquitto)
    Tasks: 1 (limit: 1129)
   Memory: 1.2M (peak: 1.5M)
      CPU: 190ms
   CGroup: /system.slice/mosquitto.service
           └─20042 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

May 30 03:45:26 ip-172-31-5-37 mosquitto[20042]: 1748576726: Bridge local.ip-172-31-5-37.bridge_to_emqx doing local SUBSCRIBE on topic #
May 30 03:45:26 ip-172-31-5-37 mosquitto[20042]: 1748576726: Connecting bridge (step 1) bridge_to_emqx (deployment>:8883)
May 30 03:45:57 ip-172-31-5-37 mosquitto[20042]: 1748576757: Bridge local.ip-172-31-5-37.bridge_to_emqx doing local SUBSCRIBE on topic #
May 30 03:45:57 ip-172-31-5-37 mosquitto[20042]: 1748576757: Connecting bridge (step 1) bridge_to_emqx (<dirección:1883)
May 30 03:46:28 ip-172-31-5-37 mosquitto[20042]: 1748576788: Bridge local.ip-172-31-5-37.bridge_to_emqx doing local SUBSCRIBE on topic #
May 30 03:46:28 ip-172-31-5-37 mosquitto[20042]: 1748576788: Connecting bridge (step 1) bridge_to_emqx (EMQX:1883)
May 30 03:46:59 ip-172-31-5-37 mosquitto[20042]: 1748576819: Bridge local.ip-172-31-5-37.bridge_to_emqx doing local SUBSCRIBE on topic #
May 30 03:46:59 ip-172-31-5-37 mosquitto[20042]: 1748576819: Connecting bridge (step 1) bridge_to_emqx (del:1883)
May 30 03:47:11 ip-172-31-5-37 mosquitto[20042]: 1748576831: Bridge local.ip-172-31-5-37.bridge_to_emqx doing local SUBSCRIBE on topic #
May 30 03:47:11 ip-172-31-5-37 mosquitto[20042]: 1748576831: Connecting bridge (step 1) bridge_to_emqx (deployment>:8883)
```

6. Prueba de implementación de AWS, EMQX y Mosquitto (local)

Para probar la conexión entre EMQX y AWS, debemos ingresar a la plataforma EMQX en la sección Online Test. Allí, debemos autenticarnos utilizando el nombre de usuario y contraseña que se crearon previamente durante la configuración del broker. En el apartado Subscriptions, debemos ingresar el topic donde recibiremos los mensajes o respuestas, que en este caso es: *deteccion/respuesta*

Subscriptions

* Topic QoS

Topic	QoS	Created At	Actions
deteccion/respuesta	0	2025-05-30 11:40:26	

Seguidamente, debemos ingresar el topic desde el cual enviaremos los mensajes, el cual seria “deteccion/personas“. Para este ejemplo, el mensaje que debemos enviar debe ser el siguiente:

```
{  
  "image_base64": "string"  
}
```

Donde "string" debe ser reemplazado por la base64 que represente la imagen que deseamos procesar.

Si la configuración y conexión son correctas, deberemos visualizar los siguientes mensajes:

Para probar la conexión entre Mosquitto (local), AWS y EMQX, debemos repetir los pasos anteriores

Messages

Topic: deteccion/personas QoS: 0
{ "image_base64": "string" }

2025-05-30 11:18:58

Topic: deteccion/respuesta QoS: 0
{ "status": "error", "message": "Error en detecci\u00f3n con MobileNet SSD: Incorrect padding" }

2025-05-30 11:18:58

en EMQX, excepto la parte de envío de mensajes desde su interfaz. En su lugar, debemos utilizar dos terminales en el entorno local con Mosquitto.

En la primera terminal debemos ejecutar el siguiente comando para suscribirnos al tópico: En la primera terminal deberemos ingresar el comando:

```
mosquitto_sub -h localhost -p 1883 -t deteccion/respuesta
```

En la segunda terminal debemos publicar un mensaje con el siguiente comando:

```
mosquitto_pub -h localhost -t deteccion/personas -m '{"image_base64": "prueba"}'
```

Si todo está configurado correctamente, deberemos visualizar los mensajes publicados en la terminal suscriptora.

```
[fullops@fullops ~]$ mosquitto_sub -h localhost -p 1883 -t deteccion/respuesta  
{"status": "error", "message": "Error en detecci\u00f3n con MobileNet SSD: No se pudo decodificar la imagen"}
```

Para finalizar, solamente haría falta enviar en el mensaje un base64 válido y deberemos recibir alguna de estas dos respuestas, dependiendo de si en la imagen hay una persona o no.

Resultado con persona(s) detectada(s)

Messages



Topic: deteccion/respuesta QoS: 0

```
{"status": "error", "message": "Error en detecci\u00f3n con MobileNet SSD: No se pudo decodificar la imagen"}
```

2025-05-31 16:01:55

Resultado sin persona(s) detectada(s)

Messages



Topic: deteccion/respuesta QoS: 0

```
{"status": "ok", "message": "Detecci\u00f3n ejecutada", "result": [{"confidence": 1.0, "bounding_box": {"x1": 30, "y1": 0, "x2": 536, "y2": 481}}]}
```

2025-05-31 16:03:02