Jala University

# LAB-4

Mateo Menvielle

Computer Networks 2

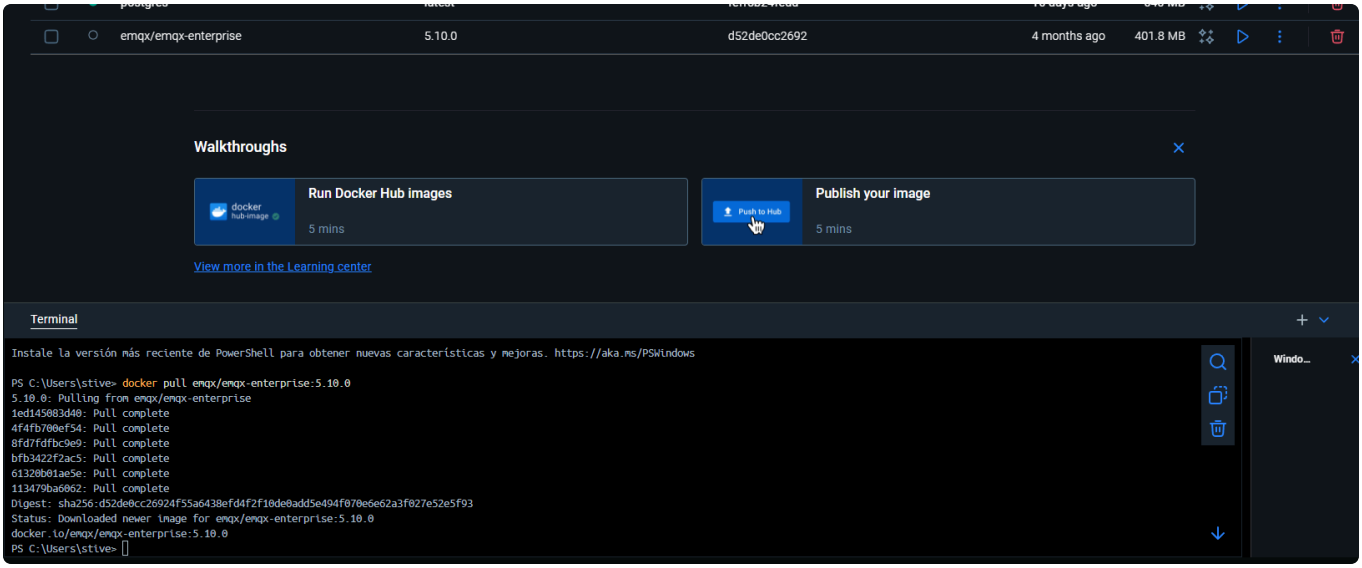
Cristian Stiven Galenao Barbosa

Funza, Cundinamarca

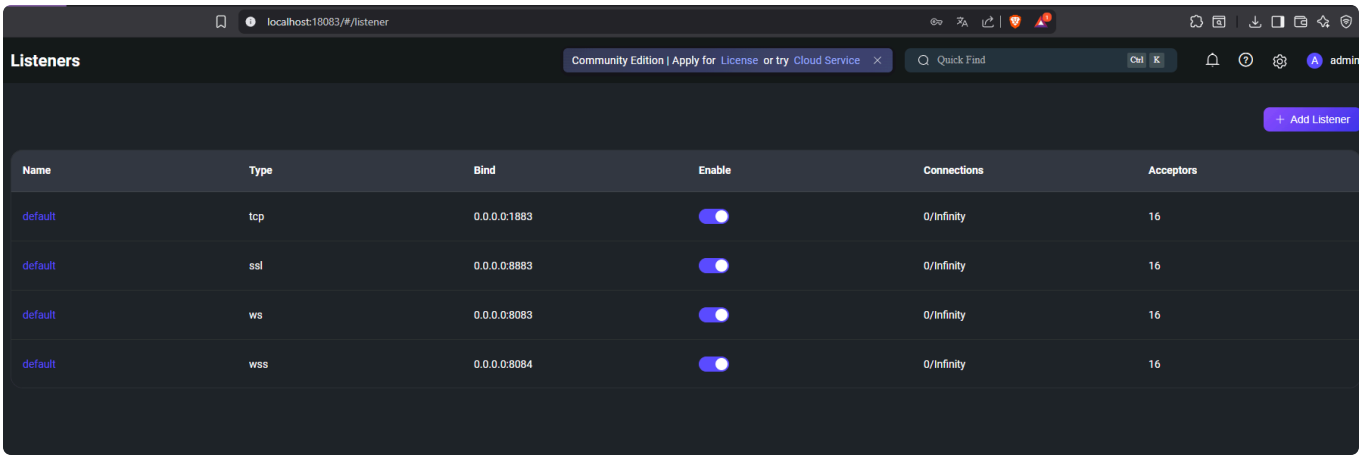
26/09/2025

# LAB 4 - Pub/Sub

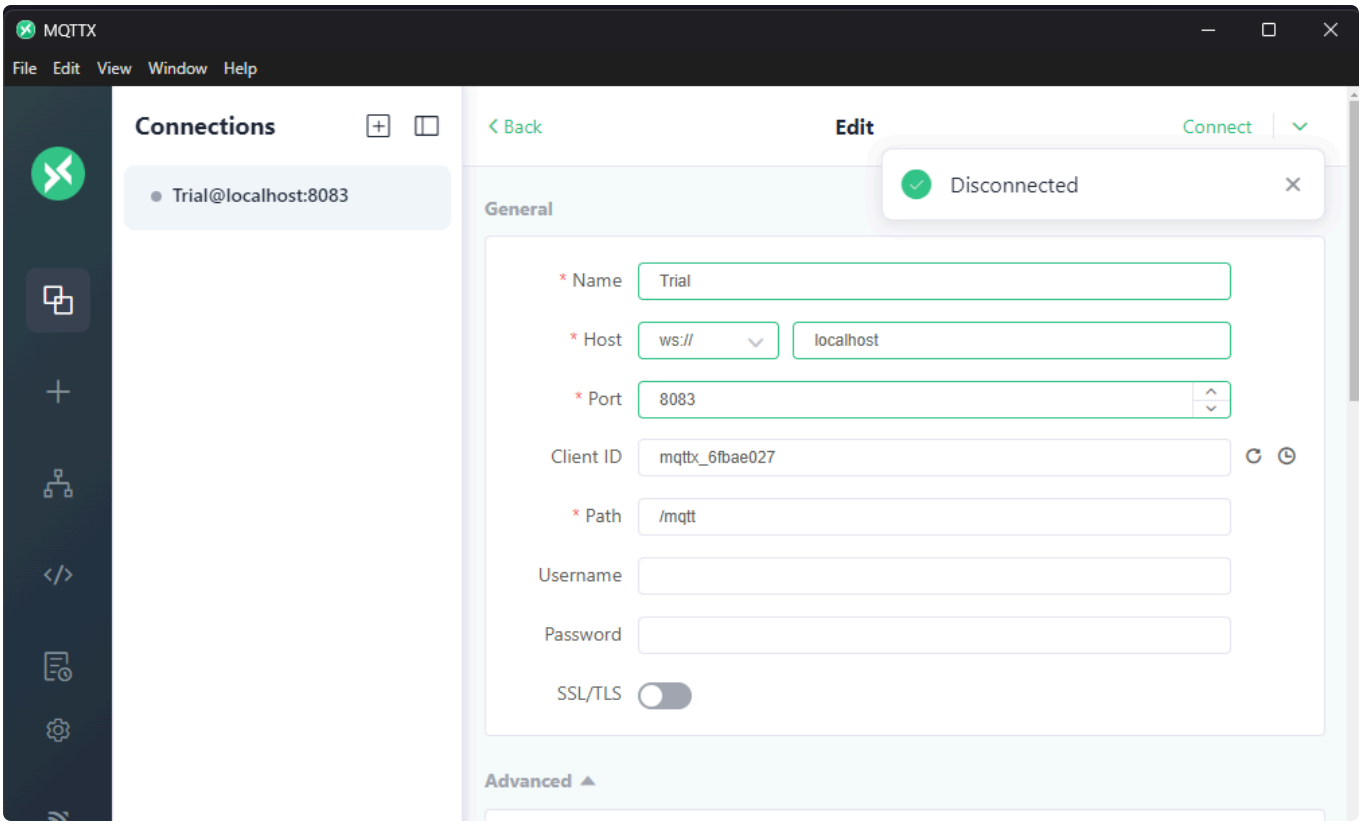
## Instalacion de EMQX en docker




Podemos controlar nuestros clientes a través nuestro localhost:18083, además usaré MQTTX para probar la conexión que esta funcionando correctamente y poder verificarlo en el ya dicho. Hay que tener en cuenta que hay un puerto abierto para los diferentes tipos de protocolos que podemos emplear, como se puede ver en la siguiente imagén.

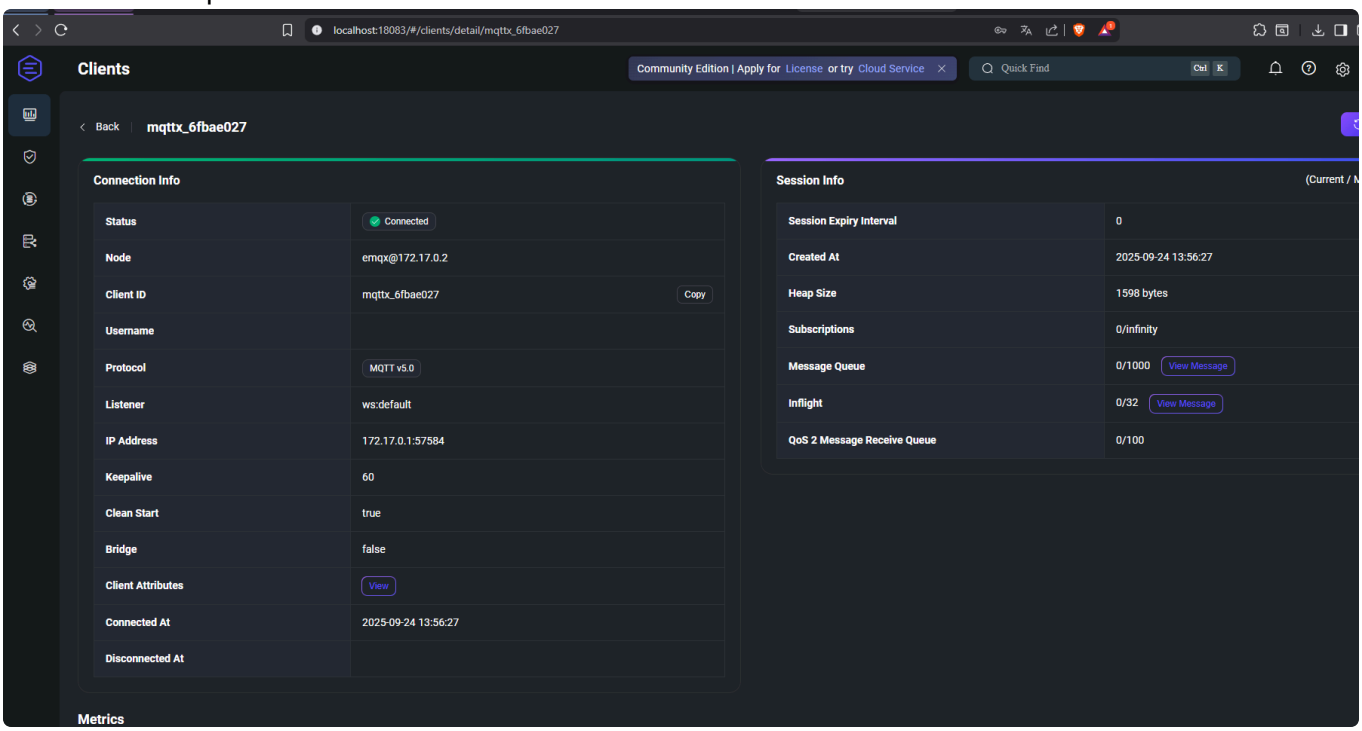



## Conexión




# Verificación

Podemos ver que funciona correctamente




# Repositorio

https://github.com/CristianSGaleanoB/computer-networks-2-week-4

Mi solución implementa el patrón publish/subscribe usando dos aplicaciones Java: un **publisher** que simula un termostato enviando lecturas periódicas cada 2 segundos.

Logica del random:

```
private static class RandomTemperature{
    public static int getRandomTemperature(){
        Random random = new Random();
        int min = -10;
        int max = 55;
        return random.nextInt(max - min + 1) + min;
    }
}
```

Publicación del cliente:

```
    int randomTemperature = RandomTemperature.getRandomTemperature();
    JSONObject payload = new JSONObject();
    payload.put("dispositive", "thermostat_01");
    payload.put("temperature", randomTemperature);
    payload.put("time", java.time.LocalTime.now().toString());

    Mqtt5Publish publicMessage = Mqtt5Publish.builder()
            .topic(TOPIC)
            .payload(payload.toString().getBytes(StandardCharsets.UTF_8))
            .qos(MqttQos.AT_LEAST_ONCE)
            .build();

    clientPub.publish(publicMessage);
    System.out.println("Publish message " + payload + " to topic: " + TOPIC);
```

Un **subscriber** que recibe esas lecturas, calcula el estado (`heating`, `cooling`, `idle`) comparando con la temperatura ideal, y vuelve a publicar la decisión en otro tópico.
Suscripción al publisher:
(Tener en cuenta que SUB_TOPIC = home/thermostat)

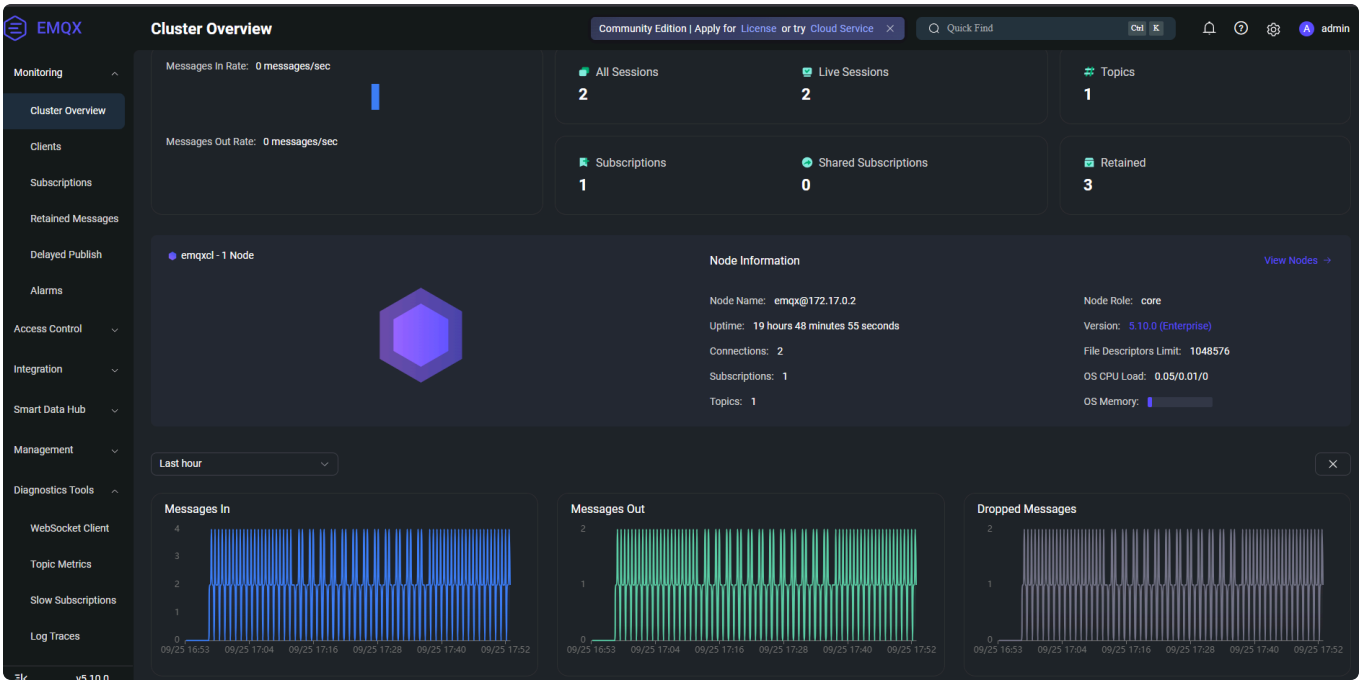
```
try {
    client.subscribeWith()
            .topicFilter(SUB_TOPIC)
            .send();
    System.out.println("Subscribed to topic: " + SUB_TOPIC);
} catch (Exception e) {
    System.err.println("Failed to subscribe: " + e.getMessage());
    return;
}
```

Lógica de toma de decisión:

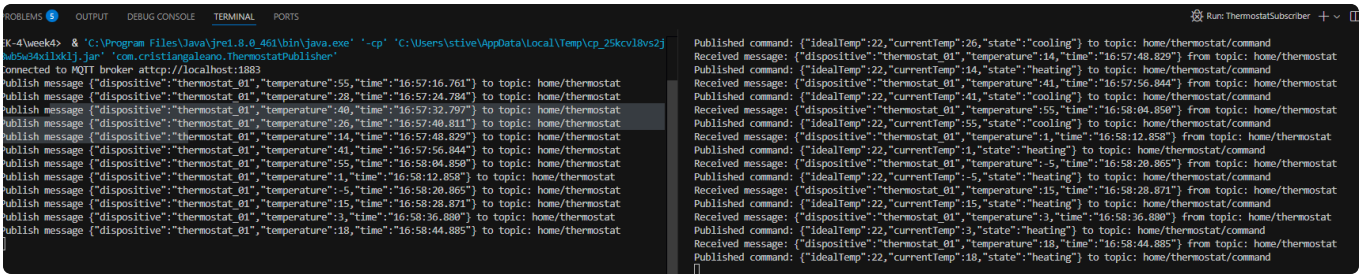
```
JSONObject payloadResponse = new JSONObject();
payloadResponse.put("currentTemp", temperature);
payloadResponse.put("idealTemp", IDEAL_TEMPERATURE);

if (temperature > IDEAL_TEMPERATURE) {
    payloadResponse.put("state", "cooling");
} else if (temperature < IDEAL_TEMPERATURE) {
    payloadResponse.put("state", "heating");
} else {
    payloadResponse.put("state", "idle");
}

try {
    client.publishWith()
            .topic(PUB_TOPIC)
            .payload(payloadResponse.toString().getBytes(StandardCharsets.UTF_8))
            .send();
```

Para la comunicación utilizo **EMQX** como broker MQTT, ejecutado en un contenedor Docker, exponiendo el puerto 1883 para las conexiones MQTT y 18083 para el dashboard de administración.
Los mensajes se envían en formato JSON a tópicos jerárquicos (`home/thermostat/lecturas` y `home/thermostat/control`), siguiendo buenas prácticas de diseño.
La librería **HiveMQ MQTT Client** facilita la creación de clientes MQTT en Java, permitiendo implementar tanto la publicación como la suscripción de manera sencilla.

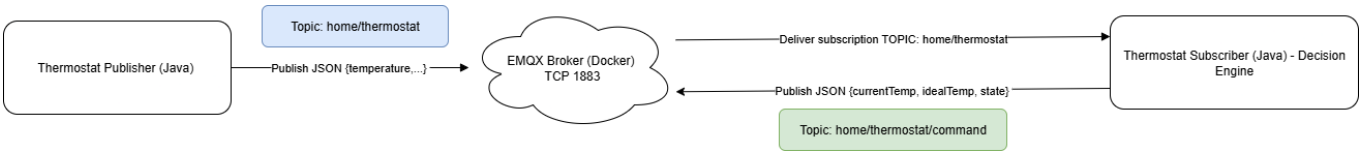
## Evidencia en el dashboard



# Evidencia de funcionamiento entre publisher y subscriber




# Diagrama de bloques




# Conclusión.

Bueno realmente este laboratorio es otro mundo a lo que hemos venido trabajando, quiero decir que al trabajar con sockets trabajamos construyendo nosotros mismos los túneles. Mientras que con MQTT es más asociado a un servicio postal porque ya tiene reglas, buzones que en este caso son los topics y niveles de entrega. Además algo que resaltar es con respecto a PUB/SUB con respecto a MQTT, el que envía no sabe quien recibe y el que recibe no sabe quien envía; Además esta basado en TCP mientrás que con los sockets podremos emplear UDP también.