



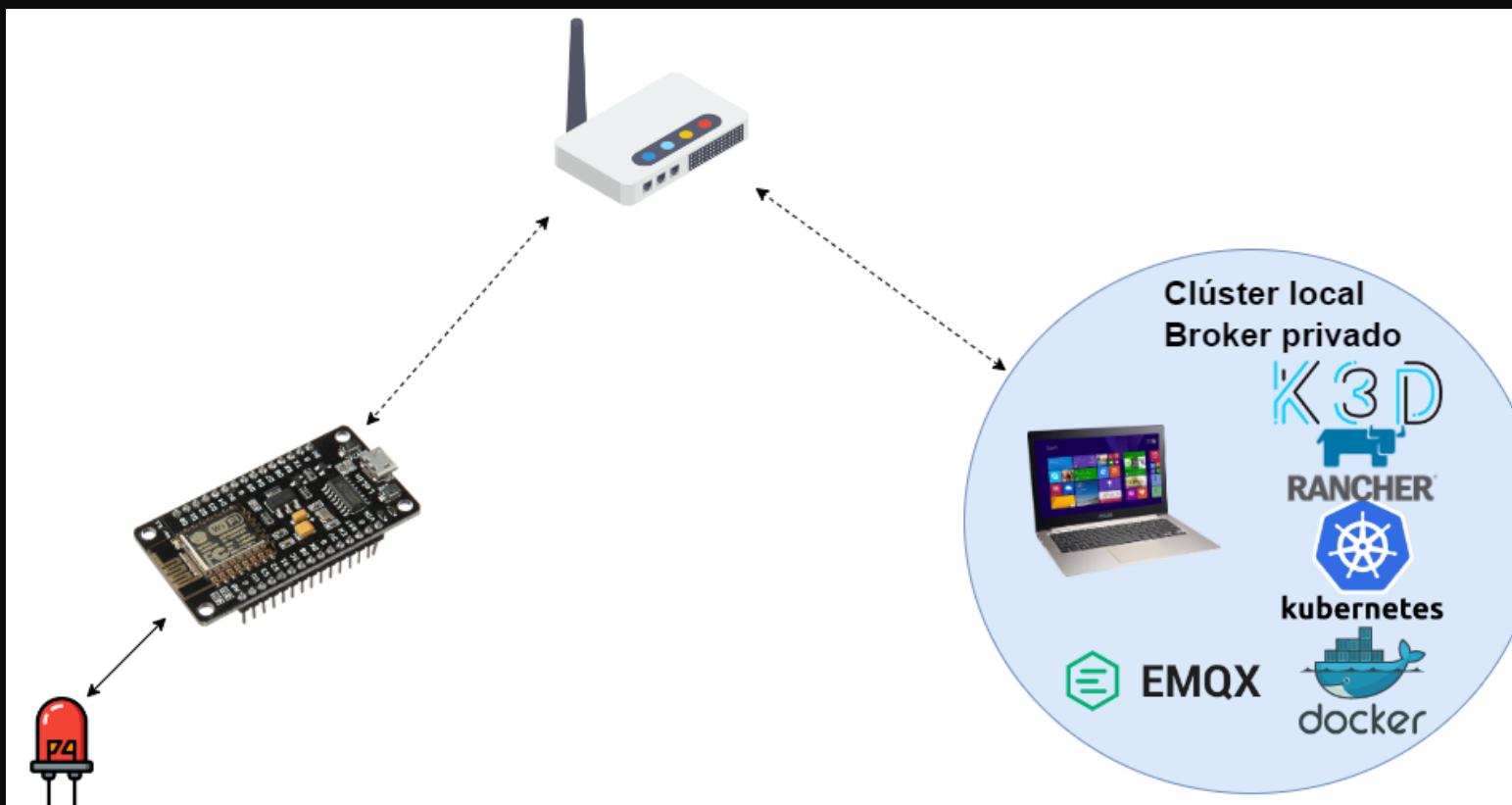
VICTOR MANUEL VELASCO HERNÁNDEZ
JEANETTE MURILLO CORTES

COMPUTACIÓN TOLERANTE A FALLAS
CALENDARIO 2022A
DR. MICHEL EMMANUEL LÓPEZ FRANCO

PROYECTO - IoT

INTRODUCCIÓN

COMPUTACIÓN TOLERANTE



Conexión de una placa NodeMCU8266 a un clúster de Kubernetes para dar una primera etapa a un sistema IoT.

Introducción

04

TECNOLOGÍAS UTILIZADAS

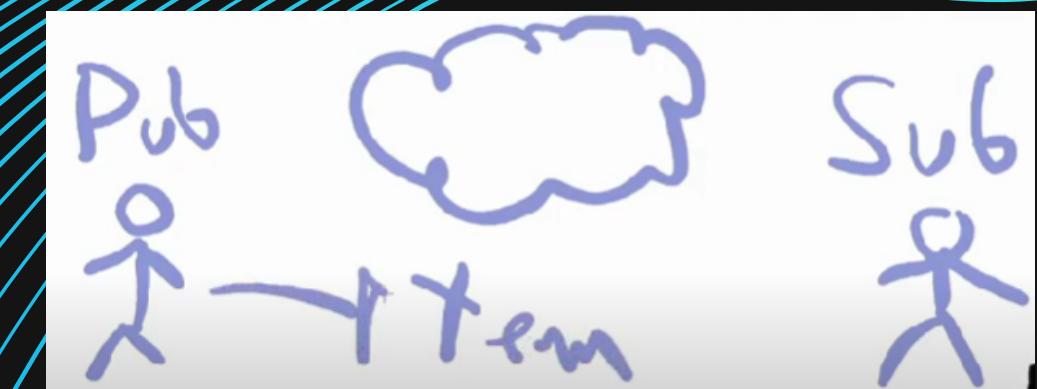
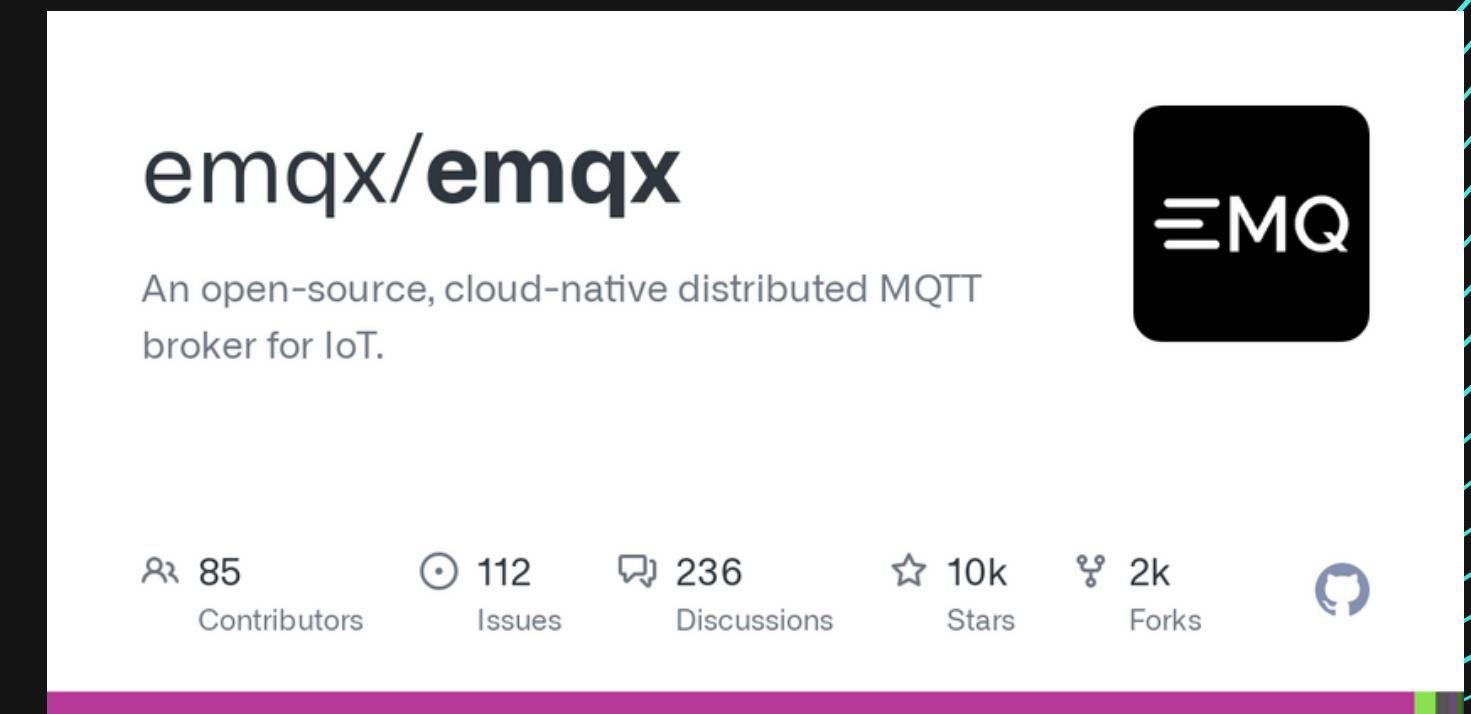
Se expondrán las tecnologías que fueron necesarias para realizar el proyecto.

CONEXIÓN NODEMCU8266

Conexión de una placa NodeMCU8266 a un clúster de Kubernetes para dar una primera etapa a un sistema IoT.

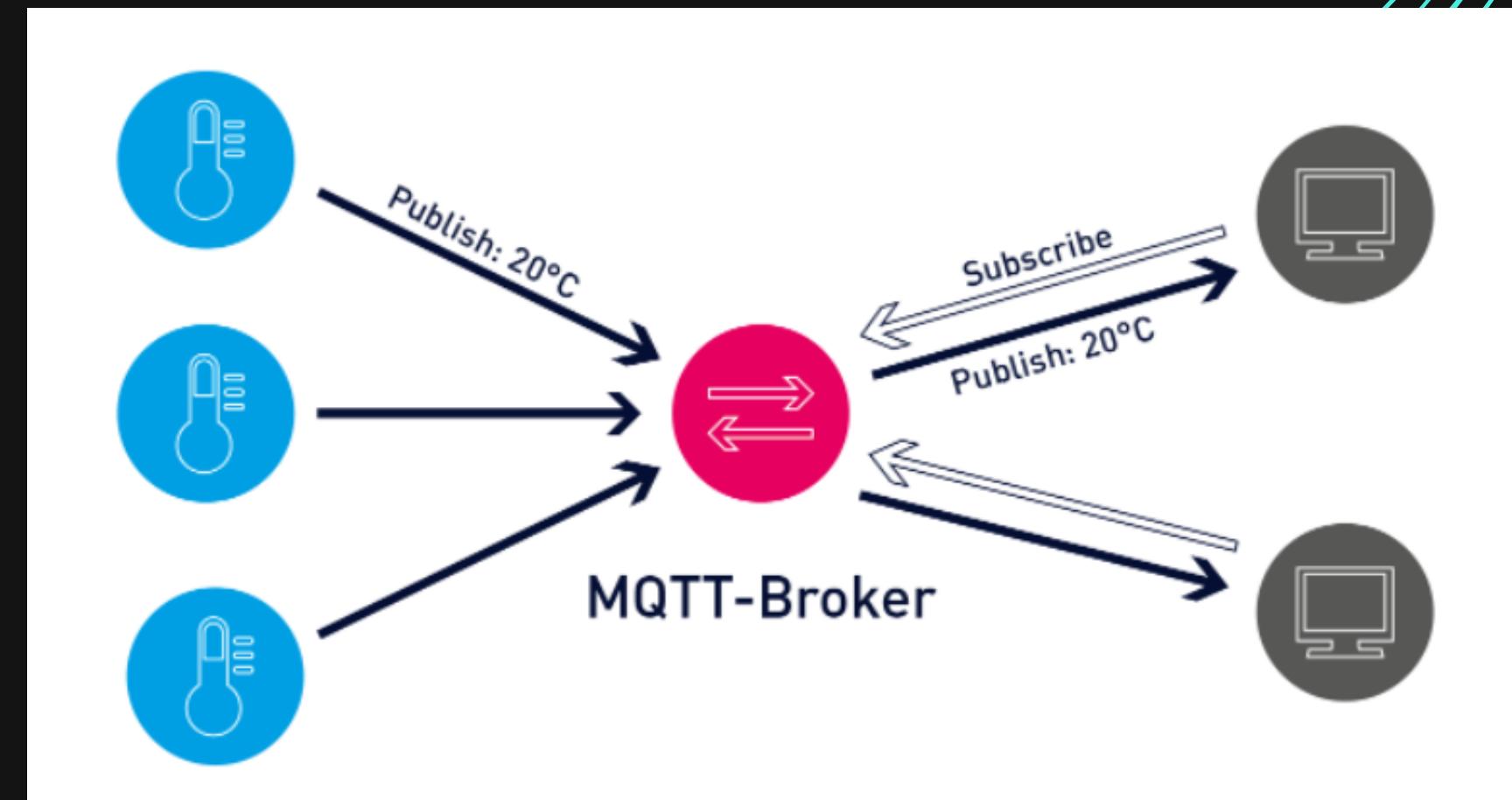
EMQX

Es un proyecto open-source, apoyado por Cloud Native, el cuál funge como broker MQTT para IoT.



MQTT

MQTT (Message Queuing Telemetry Transport) es un protocolo de comunicación maquina a maquina para redes de bajos recursos.

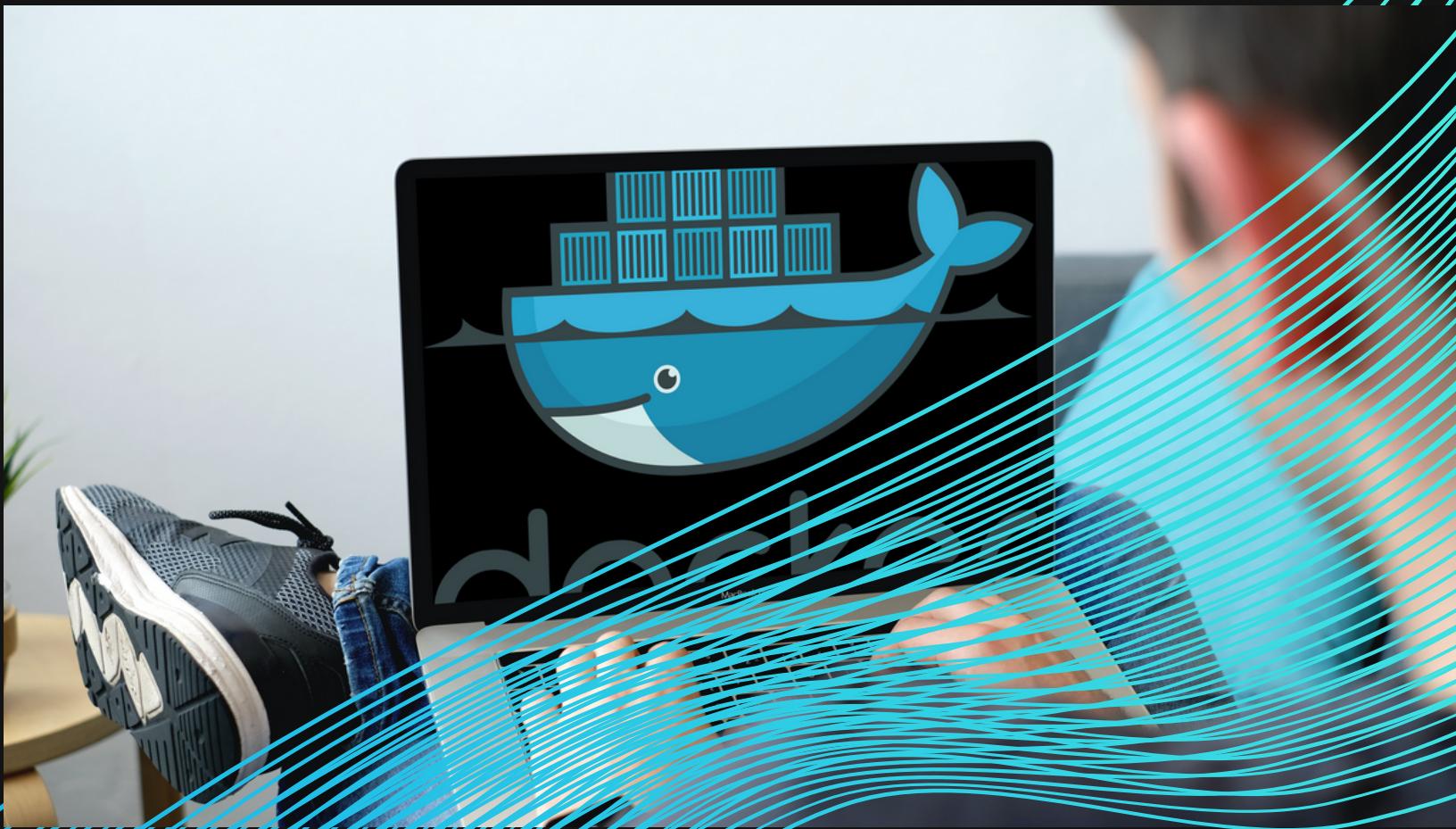


Docker

Se hará uso de la imagen "emqx/emqx", con el cuál será posible desplegarlo en Kubernetes.

DOCKER PULL EMQX/EMQX:4.4.3

```
DOCKER RUN -D --NAME EMQX -P  
1883:1883 -P 8081:8081 -P 8083:8083 -P  
8084:8084 -P 8883:8883 -P 18083:18083  
EMQX/EMQX:4.4.3
```



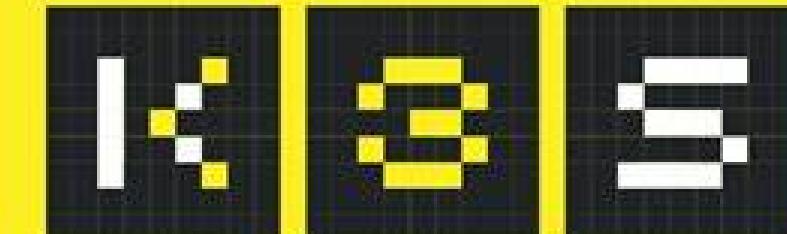
Kubernetes



Se creó un Deployment
y Service para este
clúster.

K3S

K3S es una versión ligera de Kubernetes, el cuál trabaja con Rancher (K3D).



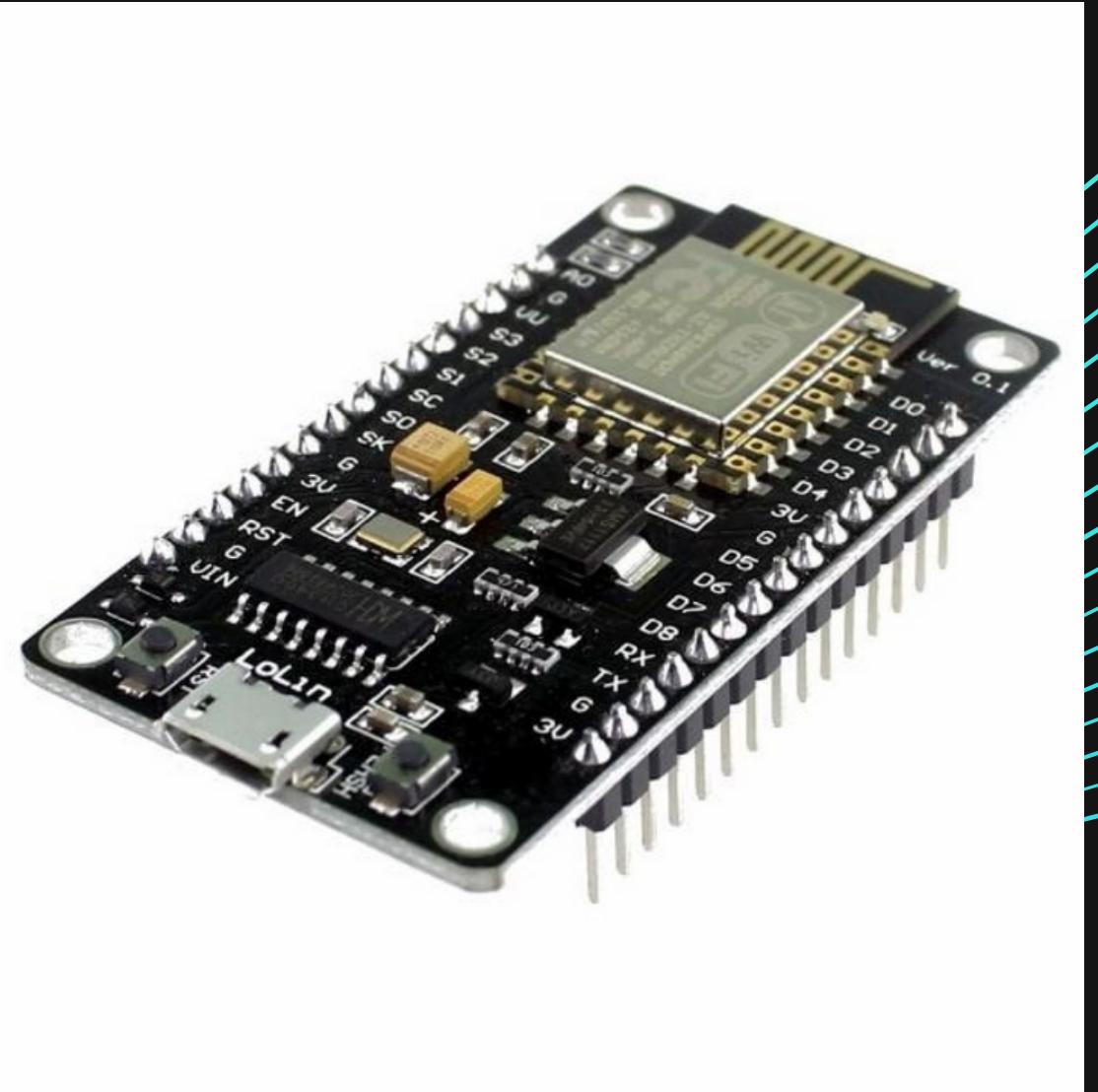
Rancher

Rancher como gestor
del clúster.



NodeMCU8266

Placa de desarrollo que cuenta con el módulo ESP8266, el cuál contiene un dispositivo Wifi de 2.4Ghz.



Instalación

Es fundamental tener instalado Docker, y para facilitar los procesos de instalación se instaló un Packet Manager llamado "Chocolatey" para instalar los programas faltantes.



```
CHOCO INSTALL K3D -Y  
>> CHOCO INSTALL JQ -Y  
>> CHOCO INSTALL YQ -Y  
>> CHOCO INSTALL KUBERNETES-HELM -Y
```

Configuración de K3D

Se creó un cluster "IoT" en K3D,
con su respectivo comando,
declarando los puertos que
serán expuestos:

```
K3D CLUSTER CREATE --API-PORT 6448 -P  
"18083:18083" -P "1883:1883" -P "8883:8883" -P  
"8081:8081" -P "8083:8083" -P "8084:8084" IOT
```

Por otro lado, es importante configurar la
variable de entorno dentro del sistema, en este
caso es Windows 10 con WSL:

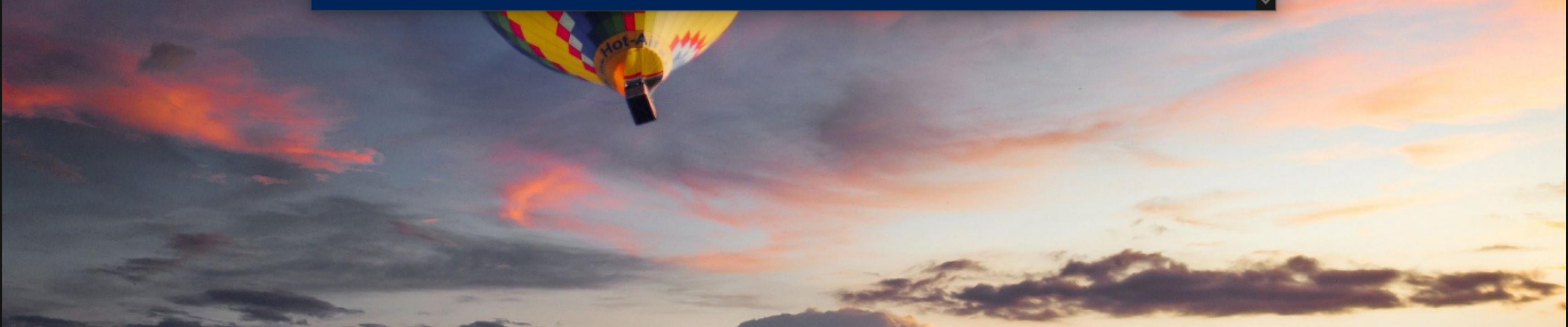
```
K3D KUBECONFIG GET ${ENV:CLUSTER_NAME}  
> $ENV:KUBECONFIG_FILE
```

```
$ENV:KUBECONFIG=($ENV:KUBECONFIG_FILE)
```



Configuración de K3D

```
PS C:\WINDOWS\system32> docker container list
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
4c4e545bd08d        ghcr.io/k3d-io/k3d-tools:5.4.1   "/app/k3d-tools noop"   3 hours ago        Up 2 hours
                                         NAMES
                                         k3d-IoT-tools
329705bddb85        ghcr.io/k3d-io/k3d-proxy:5.4.1    "/bin/sh -c nginx-pr..."  3 hours ago        Up 2 hours   0.0.0.0:1883->188
3/tcp, 0.0.0.0:8081->8081/tcp, 0.0.0.0:8083-8084->8083-8084/tcp, 0.0.0.0:8883->8883/tcp, 80/tcp, 0.0.0.0:18083->18083
                                         /tcp, 0.0.0.0:6448->6443/tcp   k3d-IoT-serverlb
ee5596fc08fe        rancher/k3s:v1.22.7-k3s1       "/bin/k3s server --t..."  3 hours ago        Up 2 hours
                                         k3d-IoT-server-0
PS C:\WINDOWS\system32>
```



Configuración de Deployment y Service

Se hará un Deployment y Servicio dentro de K3S que use la imagen "emqx/emqx", asignando los puertos correspondientes a cada uno de los servicios internos, todo esto lo lograremos con un archivo YAML generado.

K3S KUBECTL APPLY -F EMQX.YAML

```
! EMQX.yaml
 1 apiVersion: apps/v1
 2 kind: Deployment
 3   metadata:
 4     name: emqx-deployment
 5   labels:
 6     app: emqx
 7   spec:
 8     replicas: 1
 9     selector:
10       matchLabels:
11         app: emqx
12     template:
13       metadata:
14         labels:
15           app: emqx
16     spec:
17       containers:
18         - name: emqx
19           image: emqx/emqx:v4.1-rc.1
20           ports:
21             - name: mqtt
22               containerPort: 1883
23             - name: mqttssl
24               containerPort: 8883
25             - name: mgmt
26               containerPort: 8081
```

Configuración de Deployment y Service



```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> kubectl get deployments
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
emqx-deployment 1/1     1           1           163m
PS C:\WINDOWS\system32> kubectl get svc
NAME        TYPE      CLUSTER-IP    EXTERNAL-IP   PORT(S)
AGE
kubernetes  ClusterIP  10.43.0.1    <none>       443/TCP
166m
emqx-service  ClusterIP  10.43.234.76  <none>       1883/TCP,8883/TCP,8081/TCP,8083/TCP,8084/TCP,18083/TCP
163m
```

Creando servicios

Se crearan servicios para cada servicio interno del despliegue, para lo cuál se usaran de tipo NodePort para poderlos exponer al exterior, a continuación, ejemplos:

```
KUBECTL EXPOSE DEPLOYMENT EMQX-  
DEPLOYMENT --NAME EMQX-DASHBOARD --PORT  
18083 --TARGET-PORT=18083 --TYPE NODEPORT
```

```
KUBECTL EXPOSE DEPLOYMENT EMQX-  
DEPLOYMENT --NAME EMQX-SOCKET --PORT 1883  
--TARGET-PORT=1883 --TYPE NODEPORT
```

```
! EMOX.yaml  
1 apiVersion: apps/v1  
2 kind: Deployment  
3   metadata:  
4     name: emqx-deployment  
5     labels:  
6       app: emqx  
7   spec:  
8     replicas: 1  
9     selector:  
10    matchLabels:  
11      app: emqx  
12    template:  
13      metadata:  
14        labels:  
15          app: emqx  
16    spec:  
17      containers:  
18        - name: emqx  
19          image: emqx/emqx:v4.1-rc.1  
20        ports:  
21          - name: mqtt  
22            containerPort: 1883  
23          - name: mqttssl  
24            containerPort: 8883  
25          - name: mgmt  
26            containerPort: 8081
```

Creando servicios

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
emqx-deployment   1/1      1          1          163m
PS C:\WINDOWS\system32> kubectl get svc
NAME           TYPE      CLUSTER-IP    EXTERNAL-IP   PORT(S)
AGE
kubernetes     ClusterIP  10.43.0.1    <none>       443/TCP
166m
emqx-service   ClusterIP  10.43.234.76  <none>       1883/TCP,8883/TCP,8081/TCP,8083/TCP,8084/TCP,18083/TCP
163m
emqx-dashboard NodePort   10.43.243.100  <none>       18083:30535/TCP
163m
emqx-socket    NodePort   10.43.40.41    <none>       1883:30250/TCP
162m
emqx-2         NodePort   10.43.175.255  <none>       8883:31212/TCP
161m
emqx-3         NodePort   10.43.241.101  <none>       8081:30708/TCP
161m
emqx-4         NodePort   10.43.163.136  <none>       8083:32425/TCP
161m
emqx-5         NodePort   10.43.13.213   <none>       8084:32594/TCP
161m
PS C:\WINDOWS\system32>
```

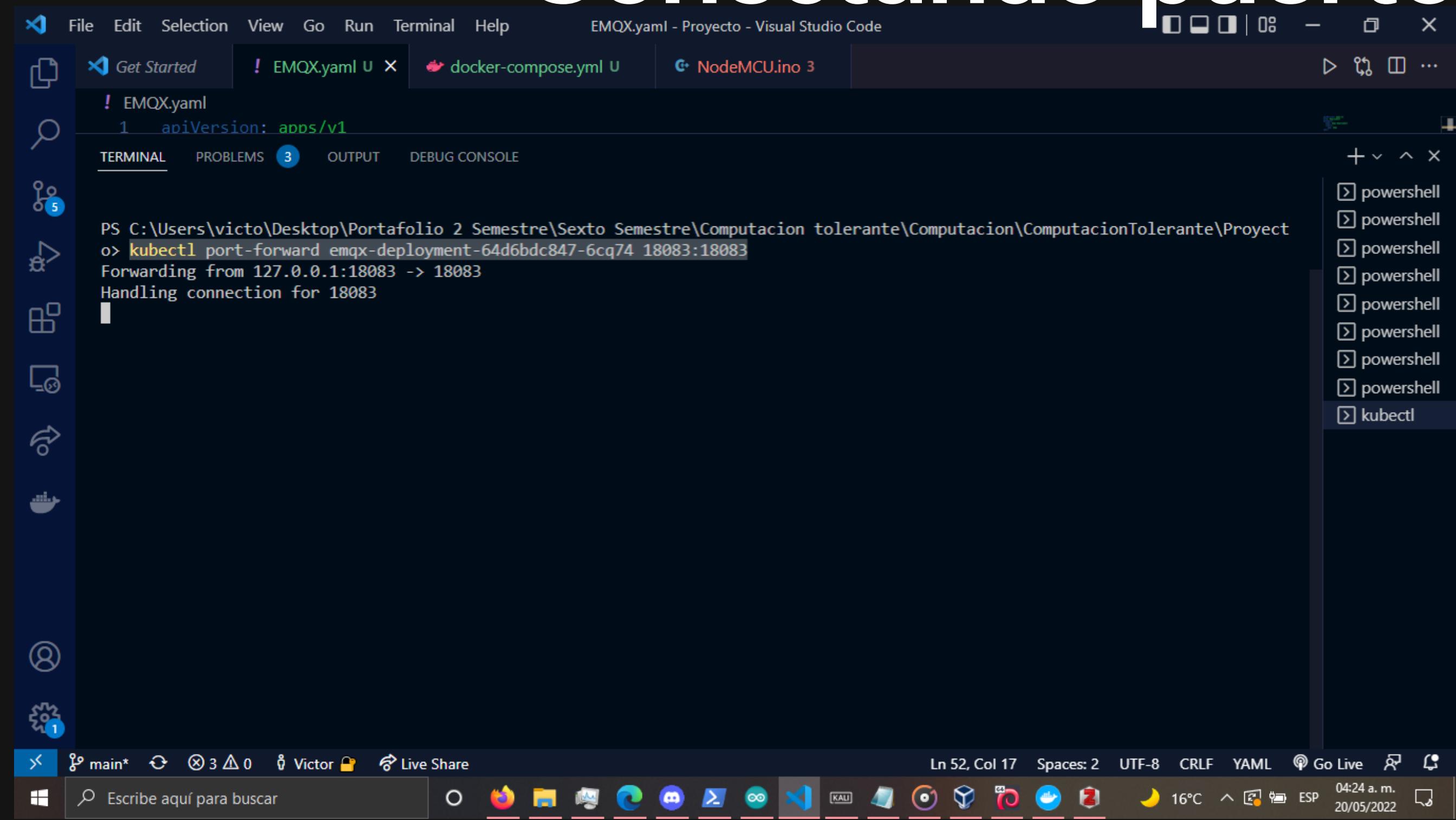
Conectando puertos

Y por ultimo los servicios los enlazamos con los puertos del servidor, a continuación, ejemplos de lo que se implementará:

KUBECTL PORT-FORWARD EMQX-DASHBOARD
18083:18083

```
! EMOX.yaml
 1 apiVersion: apps/v1
 2 kind: Deployment
 3   metadata:
 4     name: emqx-deployment
 5     labels:
 6       app: emqx
 7   spec:
 8     replicas: 1
 9     selector:
10       matchLabels:
11         app: emqx
12     template:
13       metadata:
14         labels:
15           app: emqx
16     spec:
17       containers:
18         - name: emqx
19           image: emqx/emqx:v4.1-rc.1
20         ports:
21           - name: mqtt
22             containerPort: 1883
23           - name: mqttssl
24             containerPort: 8883
25           - name: mgmt
26             containerPort: 8081
```

Conectando puertos



The screenshot shows a Visual Studio Code interface with a dark theme. The title bar reads "EMQX.yaml - Proyecto - Visual Studio Code". The left sidebar has icons for file operations, search, and other development tools. The main area has tabs for "Get Started", "EMQX.yaml", "docker-compose.yml", and "NodeMCU.ino". The "TERMINAL" tab is active, displaying the following command and its output:

```
PS C:\Users\victo\Desktop\Portafolio 2 Semestre\Sexto Semestre\Computacion tolerante\Computacion\ComputacionTolerante\Proyecto> kubectl port-forward emqx-deployment-64d6bdc847-6cq74 18083:18083
Forwarding from 127.0.0.1:18083 -> 18083
Handling connection for 18083
```

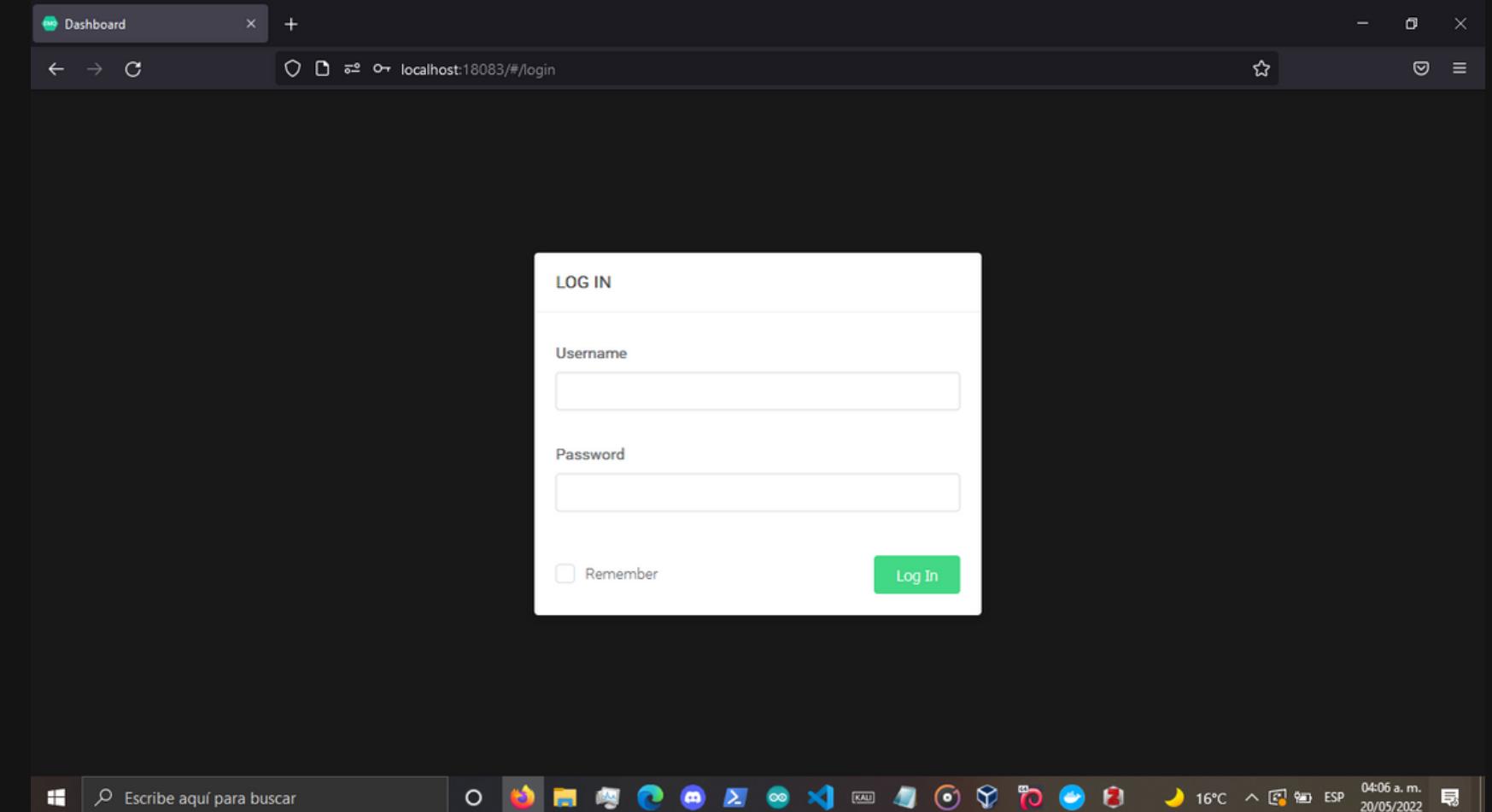
The terminal also shows a list of open PowerShell sessions on the right side.

Dashboard

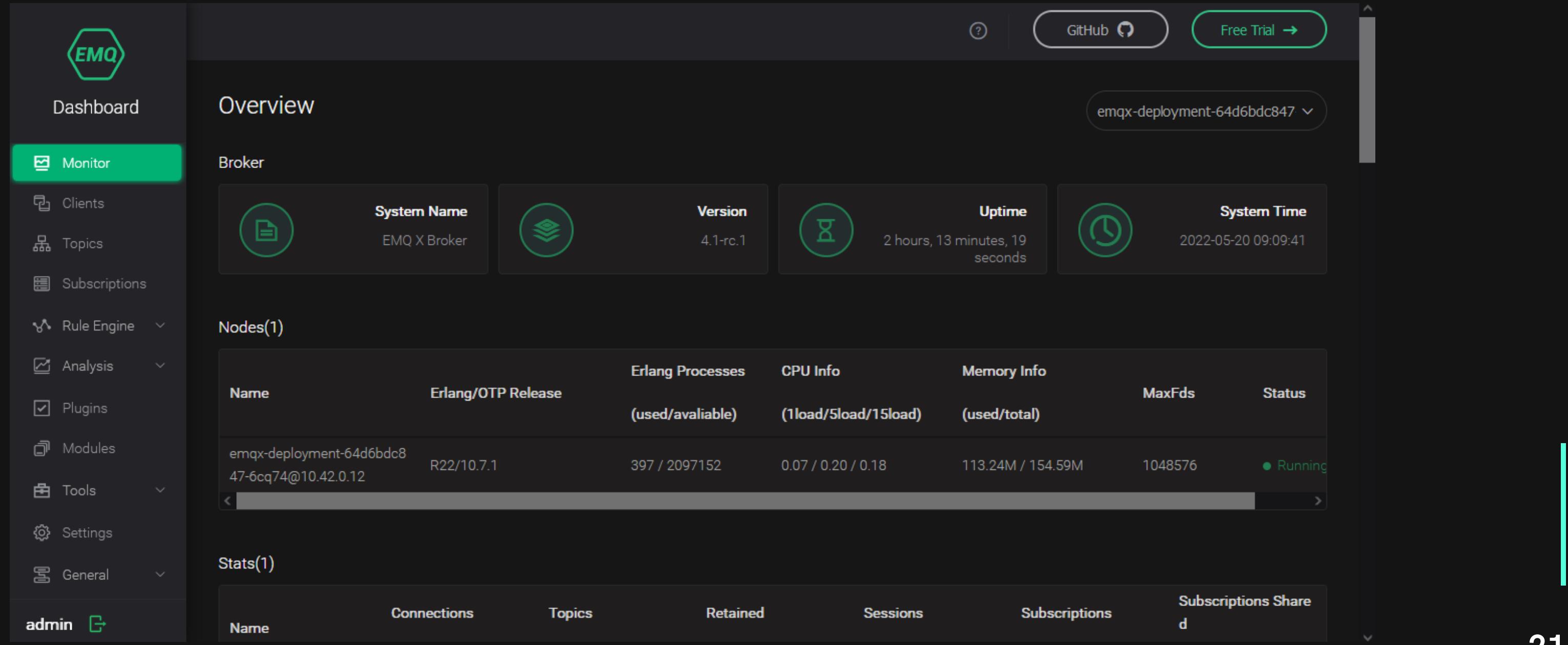
Al haber sincronizado los puertos, nos metemos al Dashboard, con puerto **18083**, en el cuál tendremos que logearnos por primera vez, usaremos el usuario default.

Usuario: **admin**

Password: **public**



Dashboard



The screenshot shows the EMQ X Broker dashboard interface. The top navigation bar includes the EMQ logo, a help icon, GitHub integration, and a free trial button. A dropdown menu shows the current deployment name: emqx-deployment-64d6bdc847.

The main content area is divided into sections:

- Overview**: Displays broker information: System Name (EMQ X Broker), Version (4.1-rc.1), Uptime (2 hours, 13 minutes, 19 seconds), and System Time (2022-05-20 09:09:41).
- Broker**: Monitored metrics for the node: emqx-deployment-64d6bdc847@47-6cq74@10.42.0.12. The table shows Erlang Processes (397 / 2097152), CPU Info (0.07 / 0.20 / 0.18), Memory Info (113.24M / 154.59M), MaxFds (1048576), and Status (Running).
- Nodes(1)**: Shows the single node with its details.
- Stats(1)**: Summary statistics for connections, topics, retained messages, sessions, subscriptions, and shared subscriptions.

The left sidebar contains a navigation menu with the following items:

- Dashboard
- Monitor** (selected)
- Clients
- Topics
- Subscriptions
- Rule Engine
- Analysis
- Plugins
- Modules
- Tools
- Settings
- General

The bottom left corner shows the user is logged in as **admin**.

Dashboard

The screenshot shows the EMQ Dashboard interface. The left sidebar has a dark theme with white icons and text. The 'Clients' option is highlighted with a green background. The main content area is titled 'Clients' and displays a table with columns: Client ID, Username, IP Address, Keepalive (s), Expiry Interval(s), Subscriptions Count, Connect Status, Create, and Operation. A message 'No Data' is shown in the table body. At the bottom of the main content area, there are search and reset buttons, and an 'Expand' button. The top right corner of the main content area has a dropdown menu showing 'emqx-deployment-64d6bdc84'. The top bar includes the EMQ logo, navigation buttons, and a GitHub link.

Client ID	Username	IP Address	Keepalive (s)	Expiry Interval(s)	Subscriptions Count	Connect Status	Create	Operation
No Data								

Dashboard

Monitor

Clients

Topics

Subscriptions

Rule Engine

Analysis

Plugins

Modules

Tools

Settings

General

admin

Escribe aquí para buscar

localhost:18083/#/clients

GitHub

Free Trial

emqx-deployment-64d6bdc84

Search

Reset

Expand

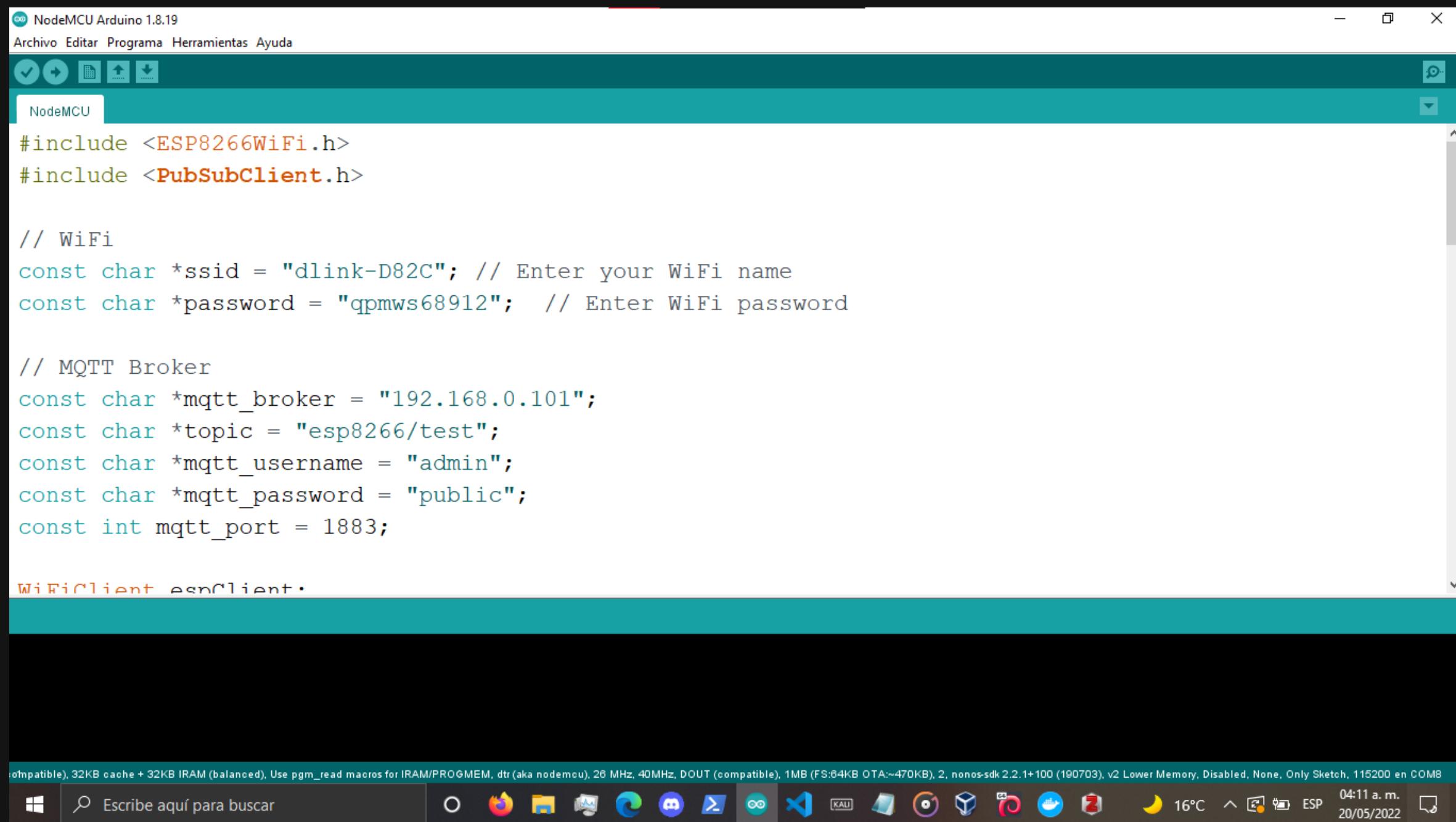
22

16°C 04:10 a.m. 20/05/2022

Dashboard

The screenshot shows the EMQ Dashboard interface. On the left, a sidebar menu includes options like Topics, Subscriptions, Rule Engine, Analysis, Plugins, Modules, Tools (selected), Websocket (highlighted in green), HTTP API, Settings, and General. The main area has a large "Subscribe" button. Below it, a "Messages" section allows sending messages to a topic. A message is currently being typed into the "Messages" input field: `{ "msg": "Hello, World!" }`. The "Topic" dropdown is set to "testtopic". The "QoS" dropdown is set to "0". A "Retained" checkbox is unchecked. A "send" button is visible. At the bottom, there are sections for "Messages already sent" and "Messages received", both currently showing "No Data". The status bar at the bottom of the browser window displays various icons and information, including the date and time.

NodeMCU8266



The screenshot shows the NodeMCU Arduino IDE interface. The title bar reads "NodeMCU Arduino 1.8.19". The menu bar includes "Archivo", "Editar", "Programa", "Herramientas", and "Ayuda". The toolbar has icons for file operations like new, open, save, and upload. The main window displays a sketch titled "NodeMCU" containing C++ code for an ESP8266. The code includes #includes for WiFi and PubSubClient, defines WiFi credentials, sets up an MQTT broker connection, and initializes a WiFi client. The status bar at the bottom provides system information: "Windows", "Escribe aquí para buscar", "O", "16°C", "16:11 a.m.", "ESP", "04/11 a.m.", "20/05/2022", and a battery icon.

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

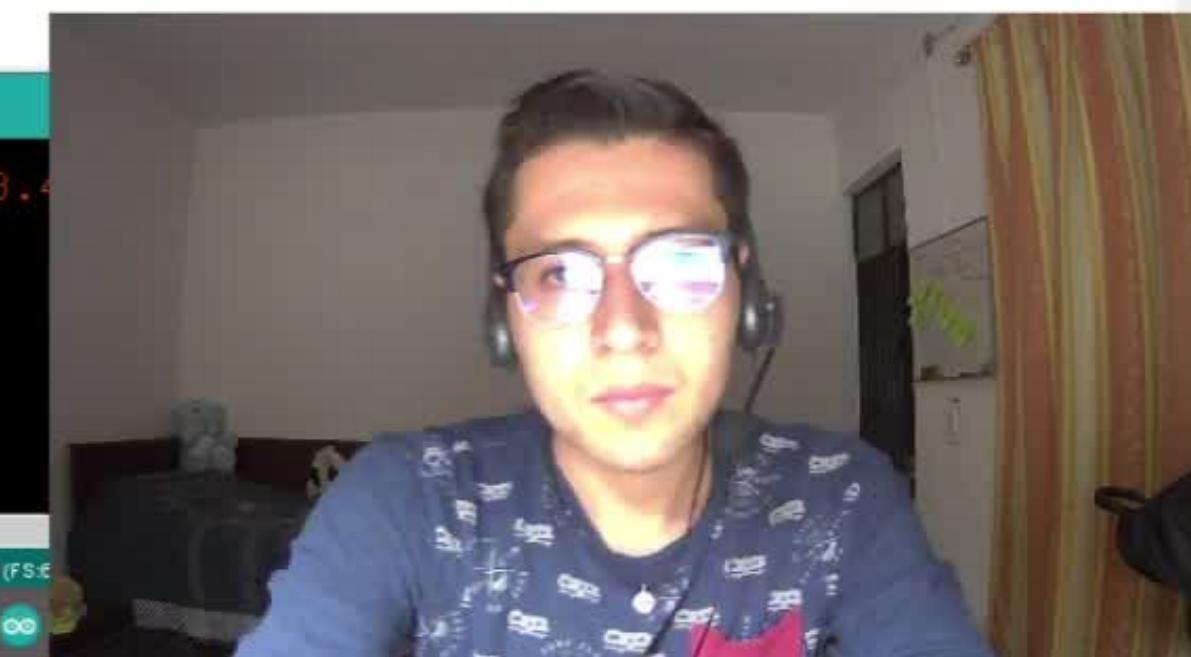
// WiFi
const char *ssid = "dlink-D82C"; // Enter your WiFi name
const char *password = "qpmws68912"; // Enter WiFi password

// MQTT Broker
const char *mqtt_broker = "192.168.0.101";
const char *topic = "esp8266/test";
const char *mqtt_username = "admin";
const char *mqtt_password = "public";
const int mqtt_port = 1883;

WiFiClient espClient;
```

Link de Drive:

[https://drive.google.com/file/d/1v7-_ClrggtQCW6O1r4HuX6lM3AB7Bfwz/view?
usp=sharing](https://drive.google.com/file/d/1v7-_ClrggtQCW6O1r4HuX6lM3AB7Bfwz/view?usp=sharing)



The image shows a screenshot of a NodeMCU Arduino IDE window. The code in the editor is as follows:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
int var;
String results;
// WiFi
const char *ssid = "dlink-D82C"; // Enter your WiFi name
const char *password = "qpmws68912"; // Enter WiFi password

// MQTT Broker
const char *mqtt_broker = "192.168.0.101";
const char *topic = "esp8266/test";
const char *mqtt_username = "admin":
```

The serial monitor window below shows the following output:

```
Subido
Wrote 283776 bytes (207814 compressed) at 0x00000000 in 18.4s
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

At the bottom, a Windows taskbar is visible with icons for File Explorer, Edge, and other applications.

Failure Injection

Se hará uso de la herramienta de Chaos Engineering, [Litmus](#), es open-source, proyecto encubado de la [CNCF \(Cloud Native Cloud Foundation\)](#), con este haremos pruebas de "pod-delete".



Failure Injection

SE INSTALA LITMUS EN EL CLÚSTER:

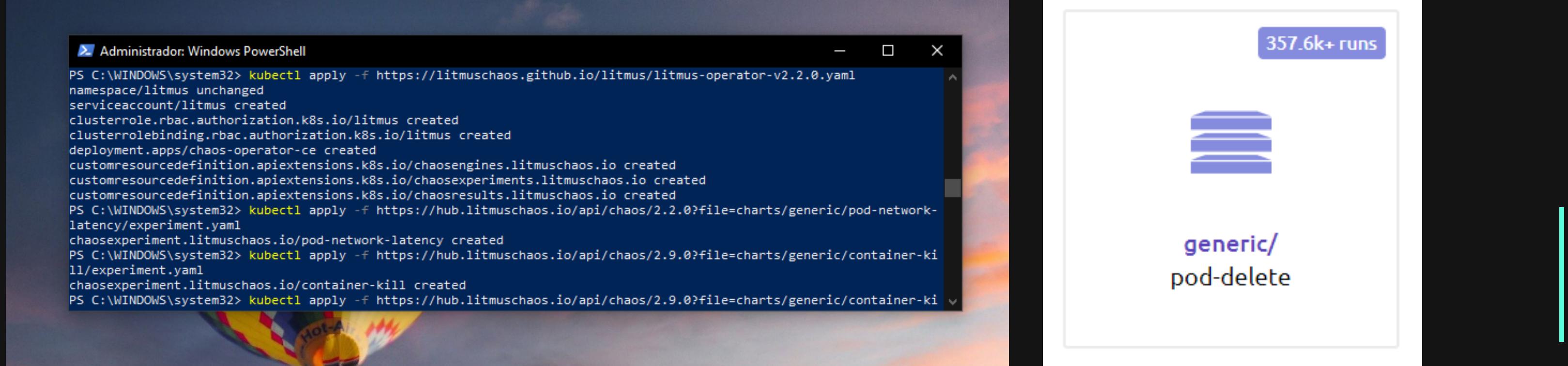
`KUBECTL APPLY -F HTTPS://LITMUSCHAOS.GITHUB.IO/LITMUS/LITMUS-OPERATOR-V2.2.0.YAML`

SE INSTALA EL CHAOS EXPERIMENT "POD-DELETE":

`KUBECTL APPLY -F HTTPS://HUB.LITMUSCHAOS.IO/API/CHAOS/2.9.0?FILE=CHARTS/GENERIC/POD-
DELETE/EXPERIMENT.YAML`

SE CONCEDEN PERMISOS(RBAC) PARA QUE EL EXPERIMENTO TRABAJE:

`KUBECTL APPLY -F HTTPS://HUB.LITMUSCHAOS.IO/API/CHAOS/2.9.0?FILE=CHARTS/GENERIC/POD-
DELETE/RBAC.YAML`



Después, descargaremos el archivo **YAML** que incluye el sitio web del "pod-deletion", para ajustar la configuración y después implementarla en el clúster.

The screenshot shows a Visual Studio Code interface. On the left is a dark-themed sidebar with various icons for file operations like Open, Save, Find, and Refresh. The main area has a dark background with light-colored text. At the top, the title bar reads "generic-container-kill-engine.yaml - 11-Failure injection - Visual Studio Code". Below the title bar, there's a toolbar with icons for File, Edit, Selection, View, Go, Run, Terminal, and Help. The main editor pane displays the following YAML configuration:

```
! generic-container-kill-engine.yaml
apiVersion: litmuschaos.io/v1alpha1
kind: ChaosEngine
metadata:
  name: emqx-deployment-chaos
  namespace: default
spec:
  # It can be active/stop
  engineState: 'active'
  appinfo:
    appns: 'default'
    applabel: 'app=emqx-deployment'
    appkind: 'deployment'
```

Below the editor is a tab bar with "TERMINAL", "PROBLEMS", "OUTPUT", and "DEBUG CONSOLE". The terminal pane at the bottom contains a PowerShell session (PS) running on Windows. The session shows commands being run to apply the YAML file and download a chaos experiment from the LitmusChaos hub. The output of the commands is visible in the terminal window.

```
PS C:\Users\victo\Desktop\Portafolio 2 Semestre\Sexto Semestre\Computacion tolerante\Computacion\ComputacionTolerante\11-Failure injection> kubectl apply -f generic-container-kill-engine.yaml
PS C:\Users\victo\Desktop\Portafolio 2 Semestre\Sexto Semestre\Computacion tolerante\Computacion\ComputacionTolerante\11-Failure injection> kubectl apply -f https://hub.litmuschaos.io/api/chaos/2.9.0?file=charts/generic/pod-delete/experiment.yaml
chaosexperiment.litmuschaos.io/pod-delete created
PS C:\Users\victo\Desktop\Portafolio 2 Semestre\Sexto Semestre\Computacion tolerante\Computacion\ComputacionTolerante\11-Failure injection> kubectl apply -f https://hub.litmuschaos.io/api/chaos/2.9.0?file=charts/generic/pod-delete/rbac.yaml
lure injection>
serviceaccount/pod-delete-sa created
role.rbac.authorization.k8s.io/pod-delete-sa created
PS C:\Users\victo\Desktop\Portafolio 2 Semestre\Sexto Semestre\Computacion tolerante\Computacion\ComputacionTolerante\11-Failure injection> kubectl apply -f generic-pod-delete-engine.yaml
chaosengine.litmuschaos.io/emqx-pod-chaos created
```

At the bottom of the screen, the taskbar shows several pinned icons for Microsoft Edge, File Explorer, Task View, and other applications. The system tray indicates the date as 20/05/2022, the time as 06:01 p.m., and the temperature as 33°C. The number 28 is displayed in the bottom right corner.

Podemos apreciar que el pod ha sido eliminado, y se ha replicado exitosamente, por lo tanto es tolerante a "pod-deletion"

The screenshot shows a Visual Studio Code interface. In the top left, there's a file icon with a red exclamation mark, indicating a problem with the file. The file name is "generic-container-kill-engine.yaml". The code content is:

```
! generic-container-kill-engine.yaml
apiVersion: litmuschaos.io/v1alpha1
kind: ChaosEngine
metadata:
  name: emqx-deployment-chaos
  namespace: default
spec:
  # It can be active/stop
```

Below the editor are four tabs: TERMINAL, PROBLEMS, OUTPUT, and DEBUG CONSOLE. The TERMINAL tab is selected, showing two sets of command-line output from a Windows terminal window:

```
\11-Failure injection> kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
container-kill-gxfzeo--1-9j5cf   0/1    Completed  0          17m
emqx-deployment-chaos-runner   0/1    Completed  0          17m
pod-delete-4eui8j--1-whckg     0/1    Completed  0          11m
emqx-pod-chaos-runner         0/1    Completed  0          11m
emqx-deployment-64d6bdc847-6cq74 1/1    Terminating 2 (70m ago) 13h
emqx-deployment-64d6bdc847-m6hlx 1/1    Running   0          29s

PS C:\Users\victo\Desktop\Portafolio 2 Semestre\Sexto Semestre\Computacion tolerante\Computacion\ComputacionTolerante\11-Failure injection>
kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
container-kill-gxfzeo--1-9j5cf   0/1    Completed  0          17m
emqx-deployment-chaos-runner   0/1    Completed  0          17m
pod-delete-4eui8j--1-whckg     0/1    Completed  0          11m
emqx-pod-chaos-runner         0/1    Completed  0          11m
emqx-deployment-64d6bdc847-m6hlx 1/1    Running   0          33s
```

The status bar at the bottom shows the current file is "main*", the cursor is at "Ln 5, Col 21", the encoding is "UTF-8", and the date and time are "07:55 p. m. 20/05/2022".

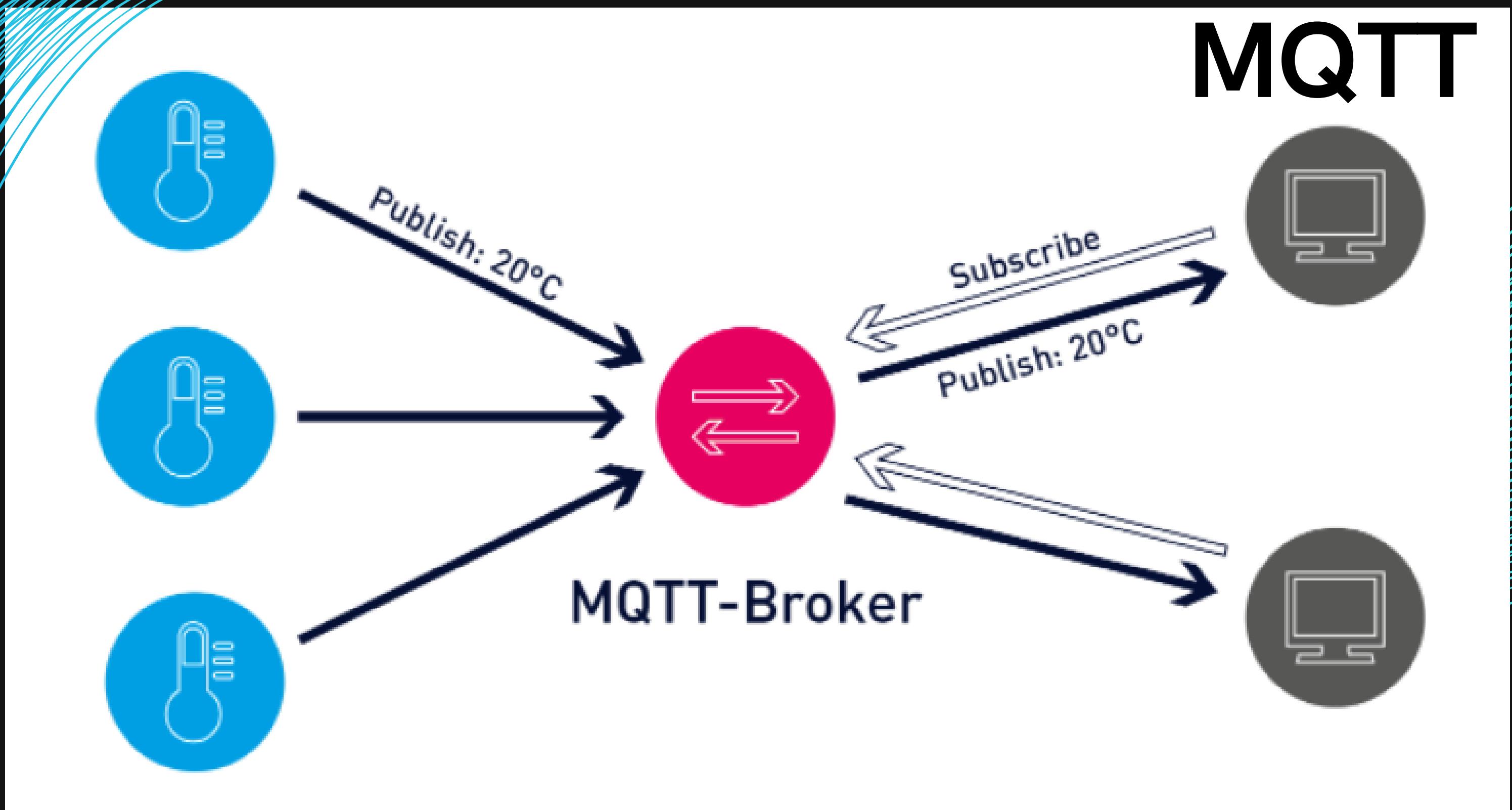
Reto

Se ha propuesto el reto de conectar más de una placa de desarrollo con las cuales:

- Se prenderán sus focos con un solo mensaje.
- Uso de sensores para enviar mensaje.



MQTT



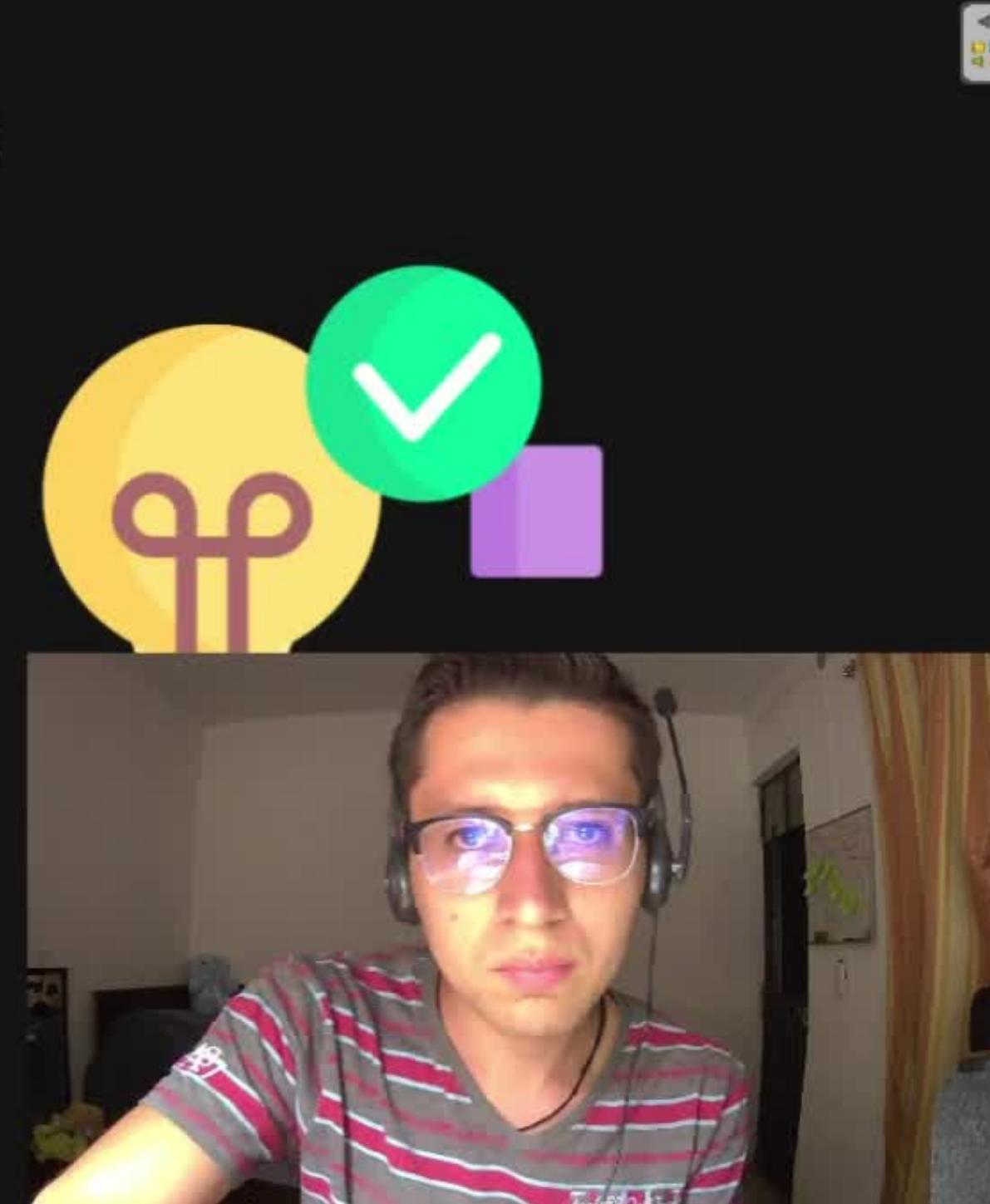
LINK DE DRIVE:
[HTTPS://DRIVE.GOOGLE.COM/FILE/D/1DDA4RBIDVZXG3WM8QDMOKXUMUQB20X7B/VIEW?USP=SHARING](https://drive.google.com/file/d/1DDA4RBIDVZXG3WM8QDMOKXUMUQB20X7B/view?usp=sharing)



Reto

Se ha propuesto el reto de conectar más de una placa de desarrollo con las cuales:

- Se prenderán sus focos con un solo mensaje.
- Uso de sensores para enviar mensaje.



localhost:18083/#/websocket

Dashboard

MQ

Monitor

Clients

Topics

Subscriptions

Rule Engine

Analysis

Plugins

Modules

Tools

Websocket

HTTP API

Alarms

Settings

admin

Escribe aquí para buscar

Messages already sent

Messages received

Messages	Topic	QoS	Time
0	esp8266/test	0	2022-05-27 01:12:23
1	esp8266/test	0	2022-05-27 01:12:01

Messages	Topic	QoS	Time
Valor fotos istancia: 0	esp8266/photo2	0	2022-05-27 01:14:45
Valor fotos istancia: 37	esp8266/photo	0	2022-05-27 01:14:44
Valor fotos istancia: 5	esp8266/photo2	0	2022-05-27 01:14:43
Valor fotos istancia: 37	esp8266/photo	0	2022-05-27 01:14:42
Valor fotos istancia: 5	esp8266/photo2	0	2022-05-27 01:14:41
Valor fotos istancia: 37	esp8266/photo	0	2022-05-27 01:14:40
Valor fotos istancia: 0	esp8266/photo2	0	2022-05-27 01:14:39

GitHub

Free Trial

25°C 01:14 a.m. 27/05/2022

The screenshot shows the EMQ MQTT Broker dashboard. On the left sidebar, under the 'Tools' section, the 'Websocket' button is highlighted in green. The main area displays two tables: 'Messages already sent' and 'Messages received'. The 'Messages already sent' table has two entries: one at 2022-05-27 01:12:23 with topic 'esp8266/test' and QoS 0, and another at 2022-05-27 01:12:01 with the same details. The 'Messages received' table lists seven messages from 'esp8266/photo2' to 'esp8266/test' with various QoS levels (0, 1, 2) and timestamps between 2022-05-27 01:14:39 and 01:14:45. The interface includes a search bar at the bottom and a system tray with weather information and system icons.



GRACIAS POR SU ATENCIÓN