BigData integration and IoT

# OBJECTIVES

After finishing this tutorial, students will be able to:

* Describe the characteristics of network protocols used for the Internet of Things (IoT), specific MQTT;
* Understand the functionality of the MQTT protocol;
* Create a small application that uses the MQTT protocol using Python programing language and PAHO-MQTT library.

# THEORETICAL NOTIONS

## The publisher/subscriber programming pattern

The publisher/subscriber pattern is a Cloud design pattern that enables an application to announce events to multiple interested consumers asynchronously, without coupling the senders to the receivers. In cloud-based and distributed applications, components of the system often need to provide information to other components as events happen.

Asynchronous messaging is an effective way to decouple senders from consumers and avoid blocking the sender to wait for a response. However, using a dedicated message queue for each consumer does not effectively scale to many consumers. Also, some of the consumers might be interested in only a subset of the information. How can the sender announce events to all interested consumers without knowing their identities?

## The solution

Introduce an asynchronous messaging subsystem that includes the following:

* An input messaging channel is used by the sender. The sender packages events into messages, using a known message format and sends these messages via the input channel. The sender in this pattern is also called the *publisher*.
* One output messaging channel per consumer. The consumers are known as *subscribers*.
* A mechanism for copying each message from the input channel to the output channels for all subscribers interested in that message. This operation is typically handled by an intermediary such as a message broker or event bus.

The following diagram shows the logical components of this pattern:

Diagram

Description automatically generated

Pub/sub messaging has the following benefits:

* It decouples subsystems that still need to communicate. Subsystems can be managed independently, and messages can be properly managed even if one or more receivers are offline.
* It increases scalability and improves the responsiveness of the sender. The sender can quickly send a single message to the input channel, then return to its core processing responsibilities. The messaging infrastructure is responsible for ensuring messages are delivered to interested subscribers.
* It improves reliability. Asynchronous messaging helps applications continue to run smoothly under increased loads and handle intermittent failures more effectively.
* It allows for deferred or scheduled processing. Subscribers can wait to pick up messages until off-peak hours, or messages can be routed or processed according to a specific schedule.
* It enables simpler integration between systems using different platforms, programming languages, or communication protocols, as well as between on-premises systems and applications running in the cloud.
* It facilitates asynchronous workflows across an enterprise.
* It improves testability. Channels can be monitored and messages can be inspected or logged as part of an overall integration test strategy.
* It provides a separation of concerns for your applications. Each application can focus on its core capabilities, while the messaging infrastructure handles everything required to reliably route messages to multiple consumers.

## MQTT protocol

MQTT is the most commonly used messaging protocol for the Internet of Things (IoT). MQTT stands for MQ Telemetry Transport. The protocol is a set of rules that defines how IoT devices can publish and subscribe to data over the Internet. MQTT is used for messaging and data exchange between IoT and industrial IoT (IIoT) devices, such as embedded devices, sensors, industrial PLCs, etc. The protocol is event-driven and connects devices using the publish /subscribe (Pub/Sub) pattern. The sender (Publisher) and the receiver (Subscriber) communicate via Topics and are decoupled from each other. The connection between them is handled by the MQTT broker. The MQTT broker filters all incoming messages and distributes them correctly to the Subscribers.

# APPLICATION

We would like to create a chat application using the MQTT message. This will be the interactive and fun part of this course. The application will:

* Subscribe to a topic BigData/IoT/chat/#;
* On every message on that topic, the application will print the message and the topic name;
* Read a message form STDIN and publish the message to the topic BigData/IoT/chat/NAME, where NAME is your name, or an alias of your name.

Step 1. //done by the teacher

1.1. Login to your Azure account

1.2. Create a new VM with Ubuntu OS;

1.3. ssh to the created VM;

1.4. Install & Configure Docker

sudo apt-get update

sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg2 \

software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

sudo apt-get update

sudo apt-get install \

apt-transport-https \

ca-certificates \

curl \

gnupg2 \

software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository \

"deb [arch=amd64] https://download.docker.com/linux/ubuntu \

$(lsb\_release -cs) \

stable"

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

1.5. create docker-compose.yml file

version: '3'

services:

mosquitto:

image: eclipse-mosquitto:1.6

ports:

- 1883:1883

- 8883:8883

volumes:

- mosquitto-data:/mosquitto/data

- mosquitto-logs:/mosquitto/logs

- mosquitto-conf:/mosquitto/config

restart: unless-stopped

volumes:

store:

config:

mosquitto-data:

mosquitto-logs:

mosquitto-conf:

1.6. Start the Eclipse-Mosquito container

**azureuser@mqttBorkerBigData**:**~**$ sudo docker-compose up -d

Creating network "azureuser\_default" with the default driver

Creating azureuser\_mosquitto\_1 ... done

Step 2 //done by students

2.1. Open VSCode

2.2. Create a file named IoTApp.py

from typing import NewType

import paho.mqtt.client as mqtt

import time

brokerEclipse = "mqtt.eclipseprojects.io"

brokerHive = "broker.hivemq.com"

brokerLocalHost = "localhost"

brokerAzure = "20.5.77.27"

def on\_log(client,userdata,level,buf):

print("log: " + buf)

def on\_connect(client,userdata, flags,rc):

if rc==0:

print("Connected OK!")

client.subscribe("BigData/IoT/chat/#")

else:

print("Connection error!",rc)

def on\_disconnect(client,userdata,flags,rc=0):

print("Disconnected with status code:" + str(rc))

def on\_message(client,userdata,msg):

topic = msg.topic

decodedMessage = str(msg.payload.decode("utf-8"))

print("Topic: " + topic + "Message received: " + decodedMessage)

client = mqtt.Client("ClientBogdan") # creaza un client cu ID-ul Test

#client.on\_log = on\_log

client.on\_disconnect = on\_disconnect

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect(brokerAzure, 1883, 60)

client.loop\_start()

while 1:

msg = input("Mesaj: ")

client.publish("BigData/IoT/chat/Bogdan", str(msg))

time.sleep(2)

2.3. Create a virtual environment for python

python<version> -m venv <virtual-environment-name>

2.4. Activate the virtual environment

source env/bin/activate

2.5. Run the python script

(env) (base) bogdan@Bogdans-MacBook-Pro IoT % python IoTApp.py

Mesaj: Connected OK!

Salut

Topic: BigData/IoT/chat/BogdanMessage received: Merry Christmas!

Mesaj: And a happy new Year!