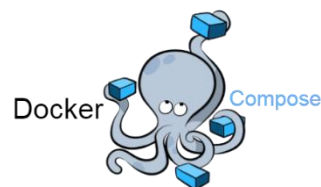


TP MQTT Docker WINDEV

Module : Virtualisation 2 : Docker



1. Sommaire

2. Objectifs.....	1
3. Installation du conteneur Broker MQTT	1
3.1. Eclipse Mosquitto.....	1
3.2. Répertoire de travail et suivi de version Git et GitHub.....	2
3.3. docker-compose Mosquitto	2
3.4. Configuration de Mosquitto	2
4. Teste du serveur Broker avec Postman	3
5. Utilisation de MQTT sur WebSocket	5
5.1. Configuration de Mosquitto	6
5.2. Installation d'un client MQTT avec une interface web dans un container (mqtt-web)	6
5.3. Configuration de la connexion à la WebSocket.....	6
5.4. Test de fonctionnement.....	7
6. Application WINDEV	7
6.1. Application Exemple WD MQTT	7
6.2. Analyse du code de l'application WD MQTT	8

2. Objectifs

- Mettre en application :
 - MQTT
 - Docker et docker-compose
 - WINDEV
 - git

3. Installation du conteneur Broker MQTT

3.1. Eclipse Mosquitto

⇒ Résumer ce qu'est **Eclipse Mosquitto** :

⇒ Donner la définition d'un **Broker** en informatique :

3.2. Répertoire de travail et suivi de version Git et GitHub

⇒ Créer un dossier « **MQTTBroker** » dans le dossier de ce TP.

⇒ Initialiser un dépôt git local en lien avec un dépôt **GitHub**.

Ce dépôt aura pour nom **MQTTBroker** et sera **public**. La notation tiendra compte de l'existence et du contenu et de l'horodatage de ce dépôt.

Pour chaque commit, indiquer dans le commentaire la référence #numéro et donner une explication.



Commit #1

⇒ Ajouter un fichier README.md, Avec comme Titre principal « **MQTTBroker** » puis ajouter un descriptif du dépôt.



Commit #2

3.3. docker-compose Mosquitto

⇒ Dans le dossier **MQTTBroker**. Créer le fichier **docker-compose.yml** avec le contenu suivant :

```
version: '3.8'

services:
  mqtt:
    image: eclipse-mosquitto
    container_name: mqtt_broker
    ports:
      - "1883:1883" # TCP
    volumes:
      - ./mosquitto/config:/mosquitto/config
      - ./mosquitto/data:/mosquitto/data
      - ./mosquitto/log:/mosquitto/log
    restart: unless-stopped
```

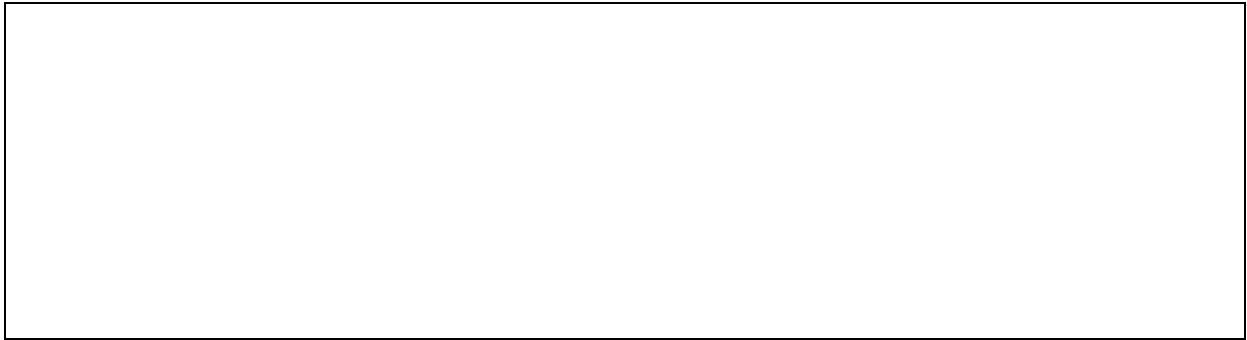
⇒ Créer les dossiers nécessaires pour les volumes configurés.

3.4. Configuration de Mosquitto

⇒ Dans le dossier **config** ajouter un fichier **mosquitto.conf** avec le contenu suivant :

```
listener 1883
allow_anonymous true
```

⇒ Exécuter le docker-compose, relever la ligne qui indique que le container est démarré.



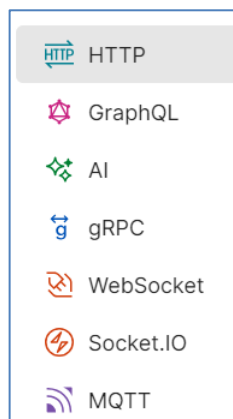
⇒ Vérifier l'état dans Docker Desktop.



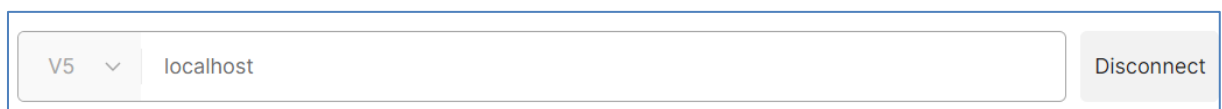
4. Teste du serveur Broker avec Postman

⇒ Installer, si nécessaire Postman.

⇒ Créer une requête MQTT.



⇒ Connecter

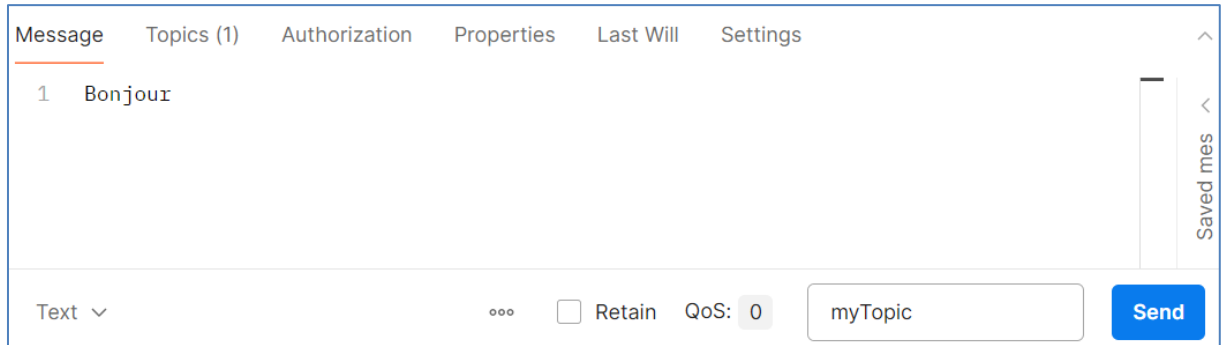


⇒ Ajouter un **sujet (topic)** avec le nom « myTopic » et s'**abonner (subscribe)**

MessageTopics (1)AuthorizationPropertiesLast WillSettings

TOPICS	+		QOS	SUBSCRIBE	DESCRIPTION
myTopic		...	0	<input checked="" type="checkbox"/>	Test MQTTBroker
Add topic			0	<input type="checkbox"/>	Add description

⇒ **Publier / envoyer un message :**

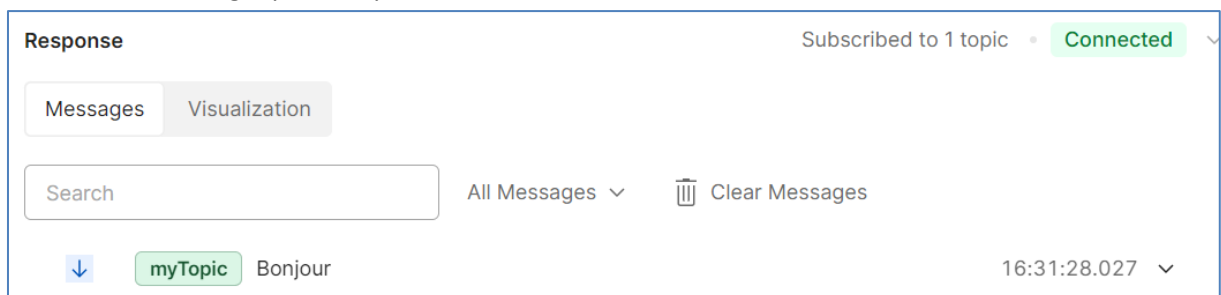


Message Topics (1) Authorization Properties Last Will Settings

1 Bonjour

Text ▾ ... ☐ Retain QoS: 0 myTopic **Send**

⇒ **Visualiser le message qui a été publié :**



Response Subscribed to 1 topic **Connected**

Messages Visualization

Search All Messages ▾ Clear Messages

↓ myTopic Bonjour 16:31:28.027 ▾

⇒ Dans WireShark en appliquant le filtre qui convient, relever les échanges lors de la connexion et la publication d'un message MQTT.

⇒ Quel est le protocole de transport utilisé et quel est le port habituellement associé à MQTT ?

⇒ Activer des logs en ajoutant dans le fichier de config de Mosquitto :

```
log_dest file /mosquitto/log/mosquitto.log
log_type all # Active tous les logs (messages, connexions, erreurs, etc.)
```

⇒ Relever les logs du démarrage du serveur, de la connexion d'un client, de l'inscription / abonnement (**subscribe**) et de la réception d'un message :

⇒ Donner une explication des 3 niveaux de qualité de service (Qos) :



Commit #4



Autoévaluation

5. Utilisation de MQTT sur WebSocket

⇒ Comment fonctionne une WebSocket ?

⇒ Quelle est l'avantage principal d'une WebSocket comparé à une socket TCP ?

5.1. Configuration de Mosquitto

⇒ Ajouter dans la configuration de Mosquitto :

```
# Activation des WebSockets
listener 9001
protocol websockets
```

⇒ Ajouter la socket pour le container :

```
- "9001:9001" # Websocket
```

5.2. Installation d'un client MQTT avec une interface web dans un container (mqtt-web)

⇒ Ajouter dans docker-compose :

```
mqttx-web:
  image: emqx/mqttx-web
  container_name: mqttx_web
  ports:
    - "8081:80" # Interface Web
  depends_on:
    - mqtt
  restart: unless-stopped
```



Commit #5



Autoévaluation

5.3. Configuration de la connexion à la WebSocket

⇒ Paramétrer une connexion sur Mosquitto via la WebSocket. Donner une capture d'écran de la configuration qui fonctionne :

5.4. Test de fonctionnement

⇒ Tester en utilisant Postman : réception et envoi de messages dans les deux sens sur un topic :



Autoévaluation

6. Application WINDEV

6.1. Application Exemple WD MQTT

- ⇒ Installer l'exemple « **WD MQTT** »
- ⇒ Tester l'application avec le Broker et le client Postman. Les deux clients s'abonnent au même topic et publie tous les deux des messages.

6.2. Analyse du code de l'application WD MQTT

⇒ Où est déclarer la variable **mqttSession**, donner le nom de la variable ?

⇒ Quels sont les propriétés de **mqttSession** à initialiser pour établir la connexion ?

⇒ Donner la fonction et les paramètres pour l'abonnement.

⇒ Donner la fonction et les paramètres pour la publication.

Fonctionnement des callbacks

⇒ Quelles sont les fonctions nécessitant en paramètre une fonction, et pourquoi ?

⇒ Les fonctions de callback sont-elles internes ou globales ?



Autoévaluation