

Msg brocker MQTT

FORMATEUR : CHACHIA ABDELILAH

CONTACT: achachia2003@yahoo.fr

SOMMAIRE

- 1. Introduction**
- 2. Publier-S'abonner**
- 3. Serveur-client**
- 4. Senario pratique**

Introduction

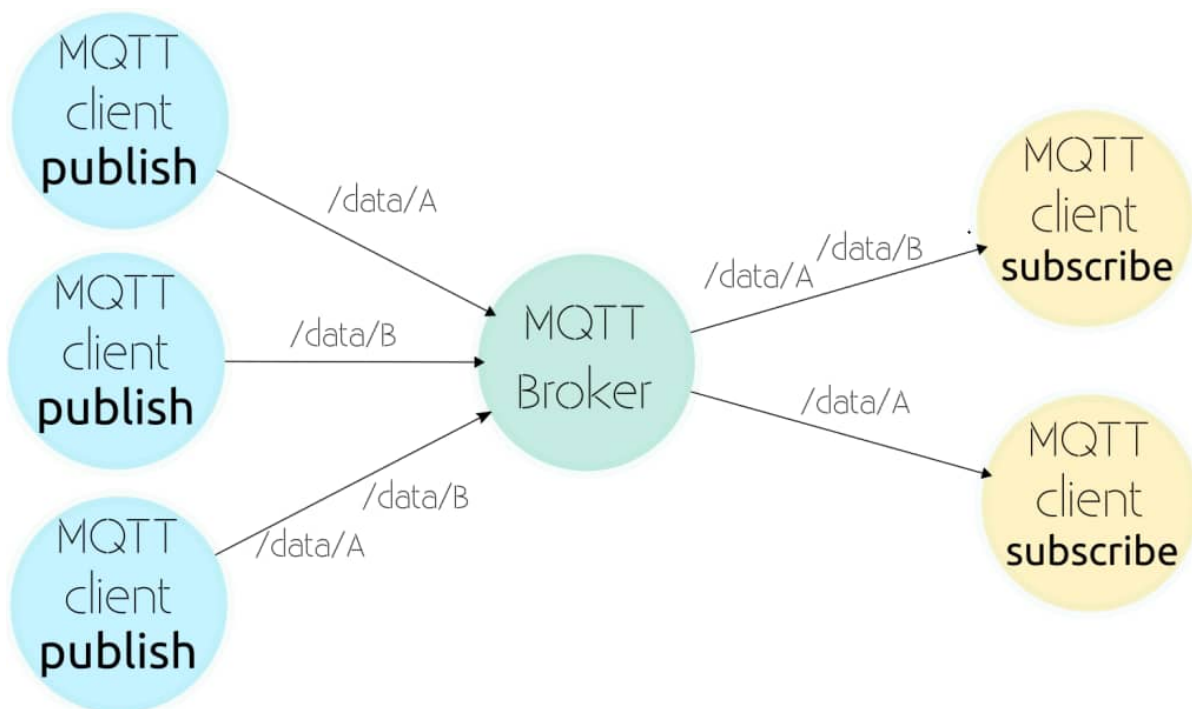
Le MQTT (ou Message Queuing Telemetry Transport) a été créé en 1999 dans le but de surveiller un oléoduc dans le désert. Il a été conçu dans l'optique d'être léger en bande passante et économe en ressources énergétiques dans le temps.

Le MQTT est maintenant un des protocoles les plus utilisés par les objets IOT (Internet Of Things - objets connectés).

Au cœur de tous les projets MQTT centrés sur des objets connectés se trouvent un serveur central, appelé Broker. Tous les objets et services s'y connectent en tant que clients.

Le Broker transmet les messages entre les clients. Les clients peuvent envoyer des messages en tant que publicateurs et recevoir des messages en tant que souscripteur.

Les messages contiennent un sujet qui décrit le contenu du message (par exemple: la météo à Paris). Les souscripteurs reçoivent chacun une copie du message s'ils ont souscrit au sujet du message publié.



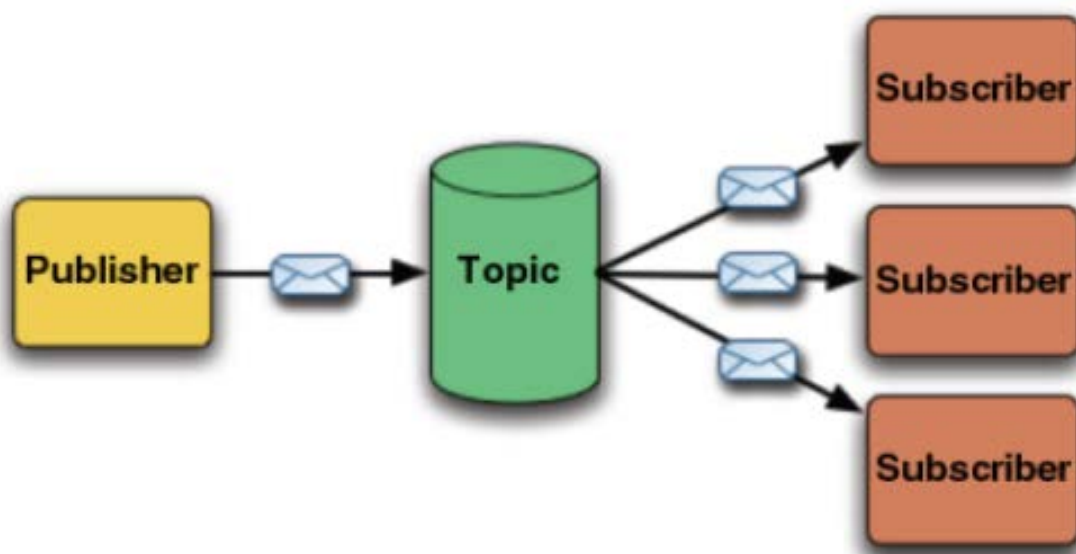
il s'agit d'un puissant protocole de transport de messagerie principalement utilisé dans les contextes de communication machine à machine (M2M) et Internet des objets (IoT).

MQTT est préféré dans ces contextes, car il est facile à mettre en œuvre et convient parfaitement aux appareils aux ressources limitées.

MQTT est un protocole de transport de messagerie client-serveur de publication-abonnement .

Publier-S'abonner

Dans l'architecture de publication/abonnement , les expéditeurs (éditeurs) ne transmettent pas les messages directement à des destinataires spécifiques (abonnés), mais classent plutôt les messages publiés en « catégories » (appelées sujets)(topic) sans savoir quels abonnés il peut y avoir.



Les éditeurs et les abonnés ne se contactent jamais directement. La connexion entre eux est gérée par un troisième composant appelé broker .

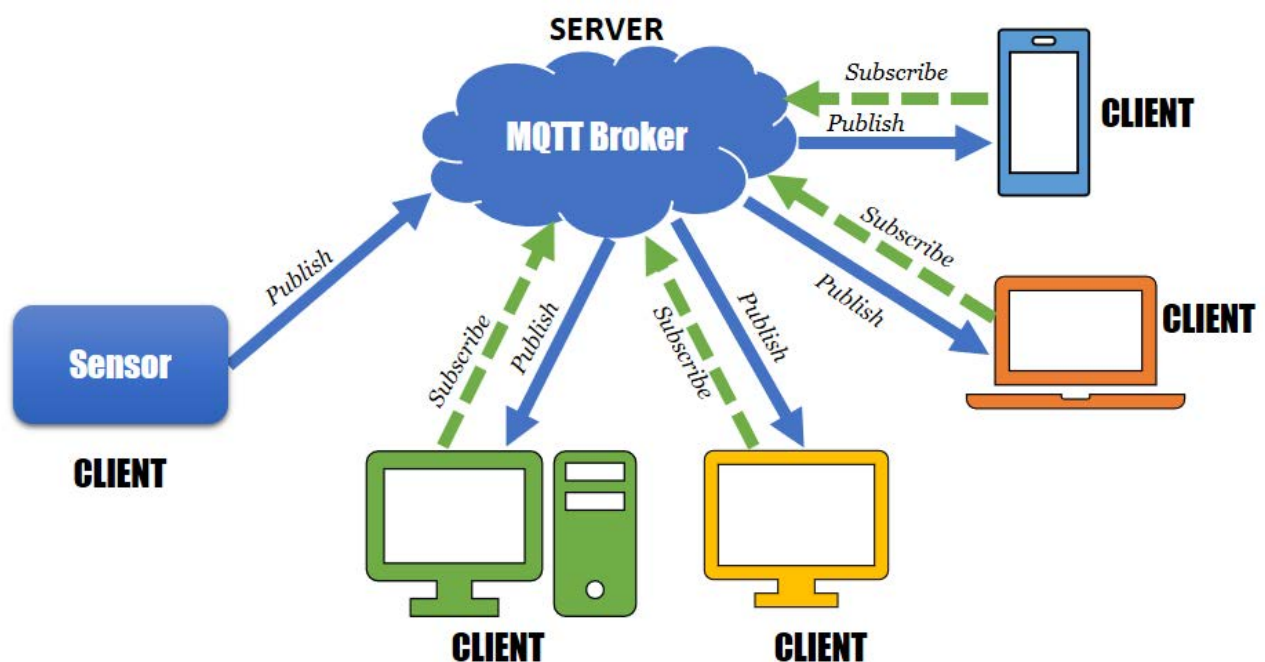
Un exemple pratique de modèle de publication/abonnement dans la vie réelle pourrait être le journal : les journalistes (éditeurs) écrivent plusieurs articles dans le journal (courtier) mais ils ne savent pas combien et quels lecteurs (abonnés) liront cet article.

Serveur client

Dans l' architecture Client-Serveur , un client se connecte à un serveur pour l'utilisation d'un service.

Dans le contexte MQTT, un client MQTT est un appareil qui se connecte à un courtier MQTT sur un réseau. Le service fourni par le courtier MQTT (serveur) est la possibilité de publier et/ou de s'abonner sur un ou plusieurs sujets.

Dans MQTT, un client peut être un éditeur et un abonné ou les deux.



Commençons à communiquer : Connexion

Avant de commencer l'échange de messages sur les sujets, le client doit initier la communication en envoyant le message CONNECT au courtier(Brocker).

Avec ce message, un client se présente à un courtier en fournissant les principales informations suivantes :

ClientID

ClientID est un identifiant unique utilisé par les courtiers pour identifier le client et stocker des informations (appelées session) à son sujet.

Un ClientID vide signifie une connexion « anonyme » : par conséquent, le broker ne mémorise aucune information sur le client.

CleanSession

Si CleanSession est défini sur false et que le courtier a des informations stockées pour ce client, le courtier utilise la session existante et remet les messages précédemment mis en file d'attente au client.

Au lieu de cela, si l'indicateur est défini sur true , cela signifie supprimer toutes les sessions et tous les messages existants pour ce client (obligatoire si le ClientId est vide).

KeepAlive

Cet intervalle, exprimé en secondes, définit la durée maximale pendant laquelle le courtier et le client peuvent rester en contact sans envoyer de message. Le client doit envoyer des messages PING réguliers, pendant la période KeepAlive , au courtier pour maintenir la connexion active.

Nom d'utilisateur et mot de passe (facultatif)

Le client peut envoyer un nom d'utilisateur et un mot de passe pour améliorer la sécurité des communications.

WillMessage (facultatif)

Un client peut spécifier son dernier message Will sous la forme d'un message MQTT et d'un sujet. Le courtier enverra ce message, au nom du client, lorsque le client se déconnectera "mal".

Sujet (Topic)

Comme mentionné précédemment, le courtier MQTT utilise le sujet pour décider quel abonné reçoit quel message.

Un client ne crée pas un sujet avant de l'utiliser.

Un serveur accepte chaque sujet valide sans aucune initialisation.

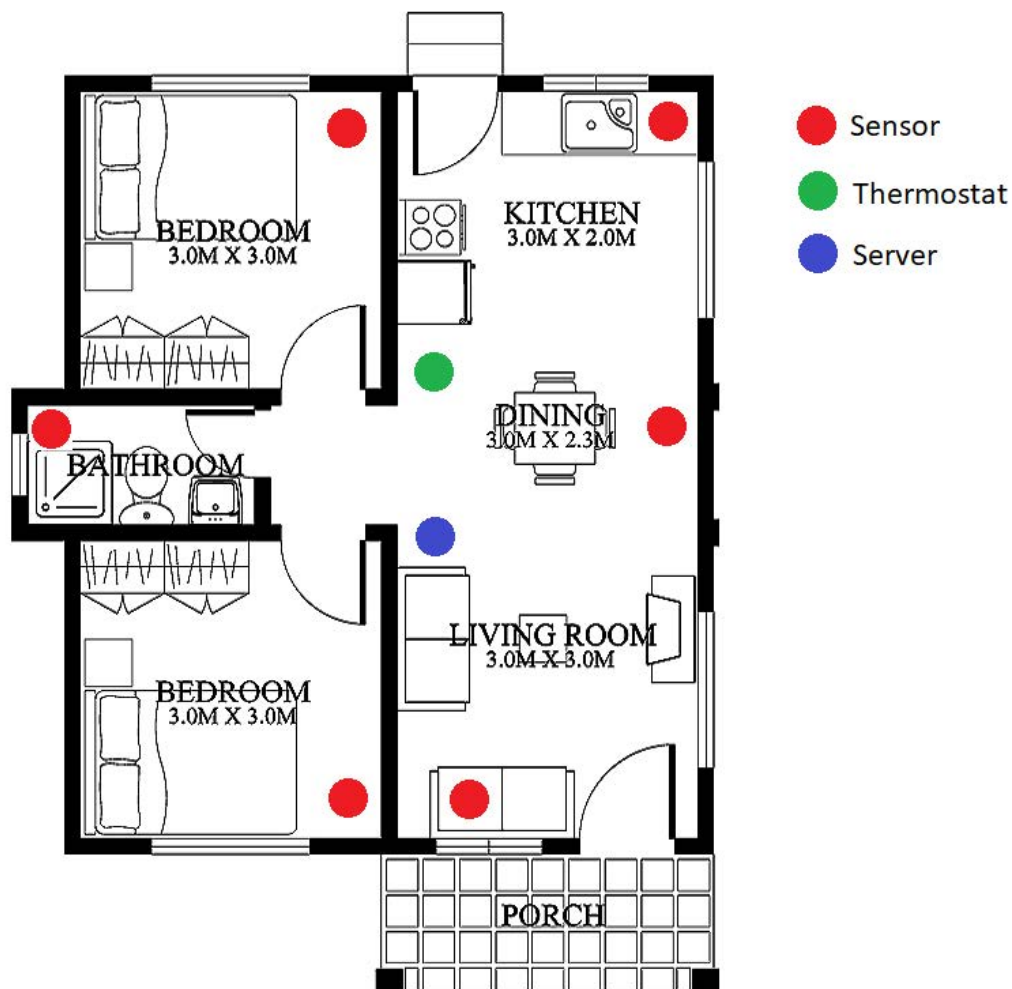
Le sujet est une chaîne avec les caractéristiques suivantes :

Au moins 1 caractère de longueur Sensible aux majuscules et minuscules

Composé d'un ou plusieurs niveaux séparés par " / "

Scenario pratique

Supposons que vous ayez des capteurs de température dispersés dans votre maison. Ces capteurs communiquent avec le thermostat à l'aide de MQTT et le thermostat utilise ces informations pour réguler le système de chauffage de la maison.



Chaque capteur (éditeur) communique sa température en publiant sur un sujet MQTT spécifique et le thermostat (abonné) surveille les températures en vérifiant ces sujets ;
ci-dessous un exemple de la structure des sujets :

maMaison/rez-de-chaussee/cuisine
maMaison/rez-de-chaussee/salle-a-manger
maMaison/rez-de-chaussee/salon
maMaison/rez-de-chaussee/chambre1
maMaison/rez- de-chaussee/chambre2
maMaison/rez- de-chaussée/salle-de-bain

Publier

Un client écrit des données sur un sujet à l'aide du message PUBLIER .
Ce message contient les informations suivantes :

Nom du sujet

Payload : contenu du message. Le protocole MQTT est indépendant des données . La charge utile est simplement une chaîne alphanumérique qui doit être interprétée par les clients.

Niveau QoS : indique le niveau de qualité de service (0, 1, 2). Nous en discuterons plus tard.

Conserver l'indicateur : définit si le message doit être enregistré par le courtier en tant que dernière bonne valeur connue pour le sujet. Lorsqu'un nouveau client s'abonne à ce sujet, il reçoit le dernier message retenu.

Ci-dessous un exemple de capteur de cuisine publiant sa température de lecture sur le thème de la cuisine :

Nom du sujet : maMaison/rez-de-chaussée/cuisine

Charge utile : 21,5

Niveau de QoS : 1

Conserver l'indicateur : FALSE

La température, à l'intérieur de la charge utile, est exprimée en degré Celsius (°C) avec un nombre décimal. L'abonné (thermostat) a besoin de connaître cette représentation afin d'interpréter correctement les données de température.

S'abonner

Pour recevoir des messages sur des sujets, le client envoie un message SUBSCRIBE au courtier.

Ce message contient une liste d' abonnements composée par :

Nom du sujet

Niveau de qualité de service

Ci-dessous un exemple de thermostat s'abonnant sur tous les sujets de capteurs .

Nom du sujet : mamaison/rez-de-chaussée/cuisine

Niveau QoS : 1

Nom du sujet : mamaison/rez-de-chaussée/salle à manger

Niveau QoS : 1

...

Nom du sujet : mamaison/rez-de-chaussée/salle de bains

Niveau QoS : 1

Se désabonner

Pour supprimer des abonnements existants à une rubrique, le client envoie un message UNSUBSCRIBE au courtier.

Le contenu est le même que le message SUBSCRIBE : une liste d'abonnements.

QoS : qualité de service

Le niveau de QoS est un accord entre l'expéditeur et le destinataire sur la garantie de livraison d'un message.

Mais pourquoi est-ce si important ? Il améliore la fiabilité .

Le protocole MQTT gère la retransmission des messages et garantit la livraison, ce qui rend la communication dans les réseaux non fiables un peu moins complexe.

Il y a 3 niveaux :

0 — Au plus une fois

Aucune garantie de livraison. Ce niveau s'appelle « feu et oublie ».

Utilisez-le lorsque vous disposez d'un canal de communication stable et lorsque la perte de messages est acceptable.

1 — Au moins une fois

Garantit qu'un message est remis au moins une fois au destinataire.

Utilisez-le lorsque les clients peuvent tolérer les messages en double. C'est le plus utilisé .

2 — Exactement une fois

Garantit que le message n'est reçu qu'une seule fois par le destinataire.

Utilisez-le lorsque votre application est critique et que vous ne pouvez pas tolérer la perte et les messages en double.

En augmentant le niveau de QoS, vous augmenterez la fiabilité de la communication, mais vous diminuerez les performances .

Commençons : connectez-vous au courtier MQTT

Tout d'abord, vous avez besoin d'un courtier MQTT.

Il existe plusieurs sites Web offrant un service de courtage MQTT gratuit et/ou d'entreprise. Ci-dessous, une liste :

1. **HiveMQ** (<https://www.hivemq.com/>)
2. **Moustique**
3. **CloudMqtt**
4. **Adafruit IO**

À des fins de démonstration, j'utiliserai le courtier MQTT public HiveMQ gratuit . Mais, en général, je suggère Adafruit IO , car il fournit plusieurs tableaux de bord sympas et bien d'autres fonctionnalités.

The screenshot shows the HiveMQ website homepage. At the top is a navigation bar with the HiveMQ logo and links for Product, Cloud, Developers, MQTT, Solutions, Blog, and Company. The main hero section features the headline "Reliable Data Movement for Connected Devices" and a sub-headline stating that HiveMQ's MQTT broker makes it easy to move data efficiently. Below this are three buttons: "Learn more", "Get HiveMQ", and "Contact Us". A secondary banner promotes "HiveMQ Cloud for free" with a "Get started now!" button. The page is divided into two main content areas. The left area, titled "Develop Efficient IoT Solutions", explains that IoT applications generate a lot of data and that HiveMQ's broker is designed for cloud-native deployments to optimize bandwidth and reduce costs. To the right of this text is a diagram of a cloud connected to a network of nodes. The bottom section, titled "Connect Any Device", highlights the challenge of connecting and moving data between devices and backend systems, noting that HiveMQ connects any device to a backend system reliably and securely via the MQTT protocol. This section is accompanied by a diagram showing a central "HIVEMQ MQTT Broker" icon connected to various IoT device icons like a satellite, a car, a house, a factory, a robot, and a server.

HIVEMQ

Product Cloud Developers MQTT Solutions Blog Company

Reliable Data Movement for Connected Devices

HiveMQ's MQTT broker makes it easy to move data to and from connected devices in an efficient, fast and reliable manner. enable new digital businesses.

Learn more Get HiveMQ Contact Us

HiveMQ Cloud for free
Experience how easy it is to deploy and manage an MQTT broker. [Get started now!](#)

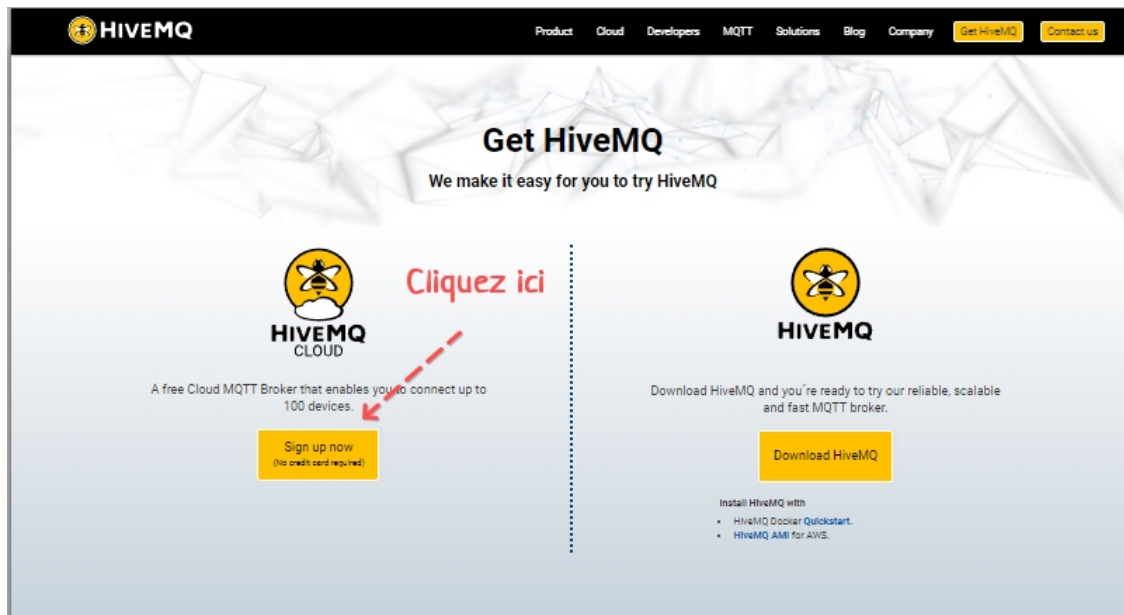
Develop Efficient IoT Solutions

IoT applications can generate a LOT of data. It is critical to select a technology that is designed to move IoT data across networks and cloud platforms. HiveMQ's MQTT broker is designed for cloud native deployments to make optimal use of cloud resources. It's use of MQTT reduces network bandwidth required for moving data. Efficient IoT solutions mean lower total costs of operation.

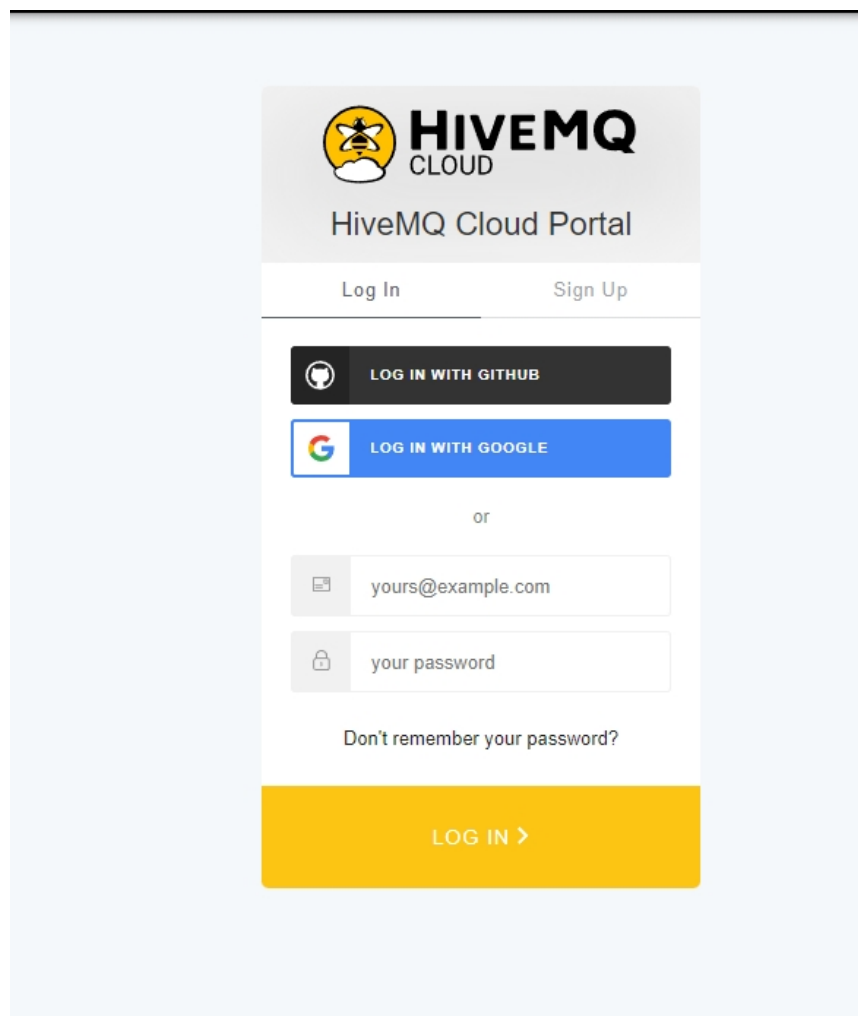
Connect Any Device

A key challenge for any IoT solution is connecting and moving data to and from devices. HiveMQ connects any device and backend system in a reliable and secure manner via the IoT standard protocol MQTT.

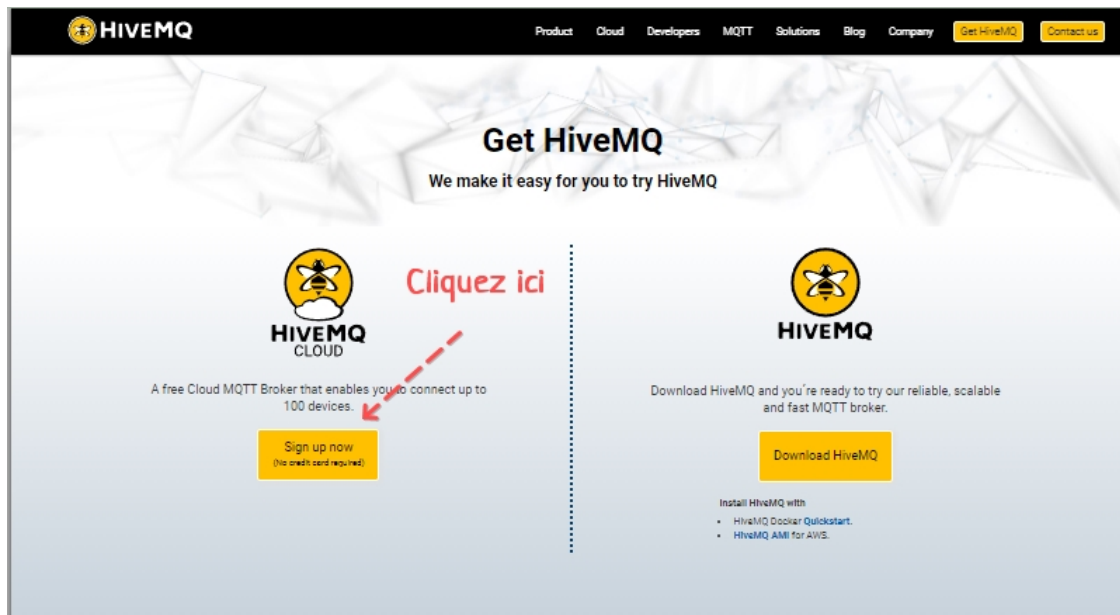
1.



2.




1.



The image shows the HiveMQ landing page. At the top is a black navigation bar with the HiveMQ logo and links for Product, Cloud, Developers, MQTT, Solutions, Blog, Company, Get HiveMQ, and Contact us. The main heading is "Get HiveMQ" with the subtext "We make it easy for you to try HiveMQ". Below this, there are two columns. The left column is for "HiveMQ Cloud", described as a free Cloud MQTT Broker that enables you to connect up to 100 devices. It has a "Sign up now (no credit card required)" button. A red dashed arrow points from the text "Cliquez ici" to this button. The right column is for downloading HiveMQ, described as a reliable, scalable, and fast MQTT broker. It has a "Download HiveMQ" button. Below this button, it says "Install HiveMQ with" followed by a list: "HiveMQ Docker Quickstart" and "HiveMQ AMI for AWS".

Get HiveMQ

We make it easy for you to try HiveMQ




HiveMQ Cloud

A free Cloud MQTT Broker that enables you to connect up to 100 devices.

[Sign up now](#)
(no credit card required)

Cliquez ici



HiveMQ

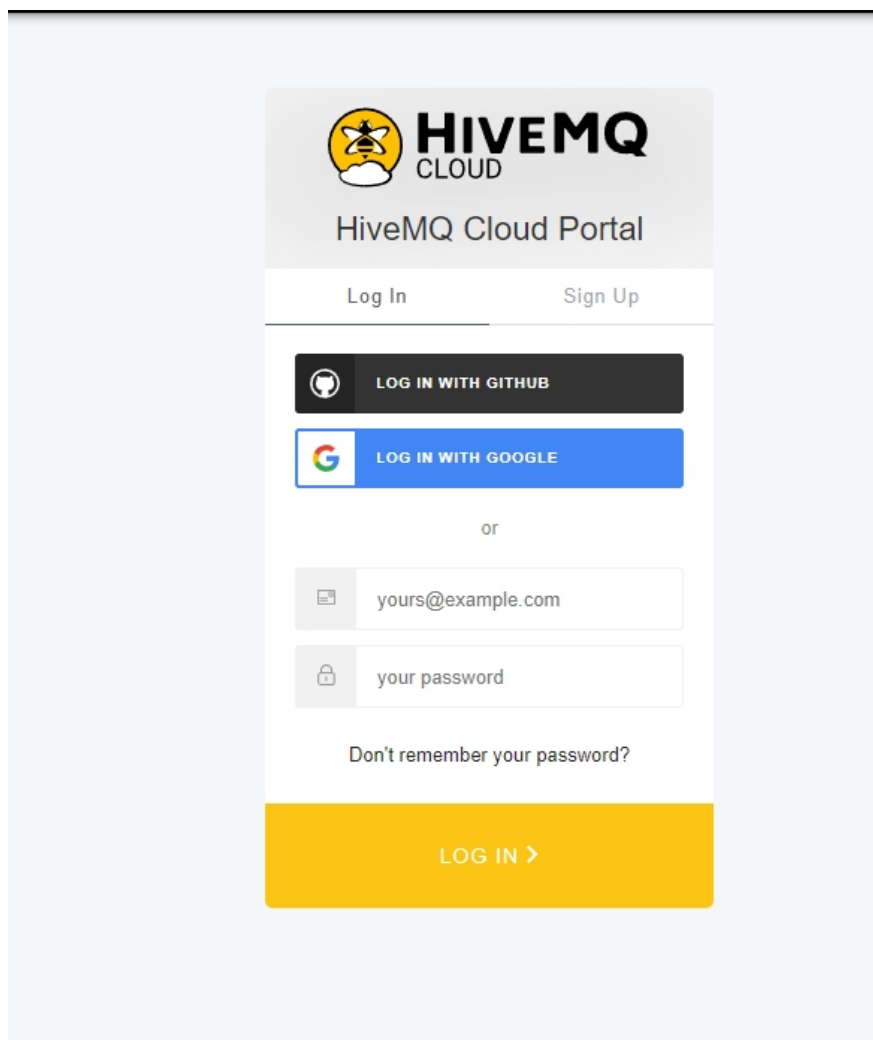
Download HiveMQ and you're ready to try our reliable, scalable and fast MQTT broker.

[Download HiveMQ](#)

Install HiveMQ with

- [HiveMQ Docker Quickstart](#).
- [HiveMQ AMI for AWS](#).

2.



The image shows the HiveMQ Cloud Portal login form. At the top is the HiveMQ Cloud logo and the text "HiveMQ Cloud Portal". Below this are two tabs: "Log In" and "Sign Up". The "Log In" tab is selected. Below the tabs are two buttons: "LOG IN WITH GITHUB" and "LOG IN WITH GOOGLE". Below these buttons is the text "or". Below "or" are two input fields: one for email (containing "yours@example.com") and one for password (containing "your password"). Below the password field is the text "Don't remember your password?". At the bottom is a large yellow button labeled "LOG IN >".

HiveMQ Cloud Portal

Log In Sign Up

[LOG IN WITH GITHUB](#)

[LOG IN WITH GOOGLE](#)

or


Don't remember your password?

[LOG IN >](#)

3.Liste des clusters

Your Clusters

Create New Cluster



FREE

Perfect for testing and small use cases

URL

PORT (TLS)

STATUS

STARTED

Running

25/03/2022 21:38

Manage Cluster

4.Détails du cluster

Cluster Details

Back to clusters

Overview

Access Management

Getting started

Cluster Overview

Connection settings

Cluster Information

Cluster Capacity

Cluster URL:

Port (TLS):

Port (Websocket + TLS):

Cluster Type:

Cloud Provider:

MQTT Client Sessions (*):

Data Traffic (*):

Data Retention Time:

Max Message Size:

Edit Cluster URL

Free

Amazon Web Services

0 / 100

0 B / 10 GB

3 Days

5 MB

Delete Cluster

Upgrade cluster

5. Gestion des accès

Détails du cluster

[Retour aux clusters](#)

Aperçu

Gestion des accès

Commencer

Identifiants MQTT

Définissez les informations d'identification que vos clients MQTT peuvent utiliser pour se connecter à votre cluster HiveMQ Cloud.

Veuillez consulter la [documentation HiveMQ](#) pour des exemples sur la façon d'utiliser les informations d'identification pour connecter un client MQTT à votre cluster.

Nom d'utilisateur

Au moins 5 caractères

Mot de passe

☐ Montrer le mot de passe

Au moins 8 caractères, chiffres, lettres majuscules et minuscules.

Confirmez le mot de passe

☐ Montrer le mot de passe

Ajouter

Identifiants MQTT actifs

Ces informations d'identification permettent aux clients MQTT de publier et de s'abonner à votre cluster HiveMQ Cloud.

Nom d'utilisateur	Mot de passe	Actions
docfile	*****	<div>Supprimer</div>
test00	*****	<div>Supprimer</div>

6. Liste des clients MQTT

Détails du cluster

[Retour aux clusters](#)

Aperçu


Gestion des accès

Commencer


Connectez vos premiers clients MQTT

Sélectionnez votre outil ou langage de programmation préféré pour obtenir des instructions de connexion étape par étape.


Outils




outil de ligne de commande **mqtt-cli**




Outil d' interface graphique **MQTT.fx**



outil de ligne de commande **mosquitto_pub/sub**




Outil de navigation **HiveMQ Websocket Client**




Arduino ESP8266
Arduino IDE utilisant ESP8266


Langages de programmation




Java
hivemq-mqtt-client




Python
Paho Python




JavaScript
mqtt.js




Java (Websocket)
hivemq-mqtt-client



C
Paho C



Allez
Paho Allez



Fléchette MQTT.Dart

MQTT Client js

```
index.js x
1  var mqtt = require('mqtt')
2
3  var options = {
4    host: '7d06b1d0322945d398580731e19e7e30.s1.eu.hivemq.cloud',
5    port: 8883,
6    protocol: 'mqtts',
7    username: 'userCloudReplit',
8    password: 'userCloudReplit69'
9  }
10
11 //initialize the MQTT client
12 var client = mqtt.connect(options);
13
14 //setup the callbacks
15 client.on('connect', function () {
16   console.log('Connected');
17 });
18
19 client.on('error', function (error) {
20   console.log(error);
21 });
22
23 client.on('message', function (topic, message) {
24   //Called each time a message is received
25   console.log('Received message:', topic, message.toString());
26 });
27
28 // subscribe to topic 'my/test/topic'
29 client.subscribe('dev');
30
31 // publish message 'Hello' to topic 'my/test/topic'
32 client.publish('dev', 'Hello-client-nodejs-cloud-replit-userCloudReplit');
```

Sortie

```
abdel-dev@DESKTOP-TBJ97MB MINGW64 ~/Desktop/Ionic/Langage programmation/MQTT/mqt
t.js-client-example (master)
$ node index.js
Connected
Received message: dev Hello-client-nodejs

abdel-dev@DESKTOP-TBJ97MB MINGW64 ~/Desktop/Ionic/Langage programmation/MQTT/mqt
t.js-client-example (master)
$ node index.js
Connected
Received message: dev Hello-client-nodejs-machine local
Received message: dev Hello-client-nodejs-cloud-replit
Received message: dev Hello-client-nodejs-cloud-replit-userCloudReplit

ser-setup.php
l/bin --filename=composer
```

Code source Github

<https://github.com/achachia/MQTT-Next-U/tree/MQTT-Client-Node-js>