**Le modèle publish/subscribe**

1. **Introduction** :

Le monde des applications communicants à travers un réseau est en plein mutation et tend vers des écosystèmes reposant sur l’Open source car leur implémentation pénètre de plus en plus les entreprises vu les solutions propriétaires couteuses. Cette communication consiste en une transmission de requête/réponse pour le modèle Client/serveur et de message pour le modèle publication/abonnement (Pub/Sub) entre divers systèmes avec utilisation d’un protocole. En effet Pub/Sub est un modèle d’échange de message utilisant un protocole de messagerie. Les protocoles de messageries utilisables définies les accords entres les points de terminaison (nœuds), la qualité des services et les formats de données. Ils sont responsable d’établissement de la connexion, des réglages des paramètres et les règles de garantie de livraison.

Cependant, beaucoup d’objet étant connecter au service Cloud, le modèle de communication publication/abonnement offre une topologie de réseau dynamique, une communication orienté contenu et prennent en compte les contraintes (fréquentes de connexions, nombres de système destinateur, criticités des ressources) ce qui le permet d’être utilisables par les applications telles que Internet des Objets (IoT), etc. Les protocoles les plus populaires utilisant ce modèle sont : AMQP, MQTT, PSync, NDN-Lite….

Dans ce chapitre nous décrirons c’est quoi Pub/Sub, son importance pour les systèmes distribués tel que IoT et quelques protocoles qui utilisent ce modèle.

1. **Définition et description** :
2. **Définition** :

Le model publication/abonnement ou Publish/subscribe est un mécanisme de publication de message pour les producteurs et d’abonnement à ces derniers pour les consommateurs. C’est un modèle dans lequel aucun consommateur ni producteur n’est ciblé.

1. **Description** :

La communication Pub/Sub n’a pas besoin de connaître l’adresse source ou de destination. Les sources et les destinateurs sont appelés respectivement les producteurs et les consommateurs (abonnés) de messages.

Les producteurs communiquent avec les abonnés de manière asynchrone à travers un courtier en diffusant des catégories de messages sans tenir compte de la façon dont ces messages seront traités. Pub/Sub diffuse ensuite des événements à tous les services qui y sont intéressés. Cette manière d’intégration augmente la flexibilité et la stabilité du système dans son ensemble car pas besoin des appels procédures ni attendre que les abonnés reçoivent les données. C’est un modèle dynamique et découplé d’où son adaptation au système distribué à grande échelle et peut être implémenter comme pattern de mémoire partagée [1]. Ce paradigme est fonctionnel et flexible en systèmes distribués avec la communication centré sur le contenu.

Subscriber

Publisher

La communication Pub/Sub met en relation trois entités dont le producteur (*Producer*), le consommateur (Consumer) et le courtier (*broker*).

1. **Les producteurs** :

Récoltent les informations et les publient au courtier, au lieu de destiner les messages à un consommateur il les associe à une catégorie de message. Il peut être une application.

1. **Les consommateurs** :

Envoient une suscription événementielle à une catégorie de message et toutes informations qui répondent aux contrainte de cet abonnement sont transmises par le courtier et plusieurs consommateurs peuvent s’intéresser aux mêmes informations. Il peut être aussi une application.

1. **Le courtier** :

Il est l’entité qui assure la livraison des messages aux consommateurs au moment opportun. Il peut posséder une petite mémoire de stockage temporaire des messages. Un exemple de courtier utilisant le protocole Pub/Sub AMQP est RabbitMQ.

1. **Les catégories de Pub/Sub** :

Pub/Sub est un modèle de communication asynchrone pour les applications distribuées.

Ce modèle est généralement divisé en deux catégories : à base de sujet et à base de contenu.

1. **Pub/Sub centré sur le sujet** :

Dans cette catégorie les éditeurs sont tenus d'étiqueter chaque événement avec un nom de sujet ; les abonnés s'abonnent à tous événements sous un sujet particulier. Chaque événement appartient à l’un d’un ensemble fixe de sujet. Cloud Pub/Sub est un protocole qui utilise ce modèle.

1. **Pub/Sub centré sur le contenu** :

Dans les systèmes basés sur le contenu, chaque abonné définit une souscription selon la structure interne des événements et chaque producteur de données attribue un nom unique et sémantiquement significatif à chaque paquet de données qu'il génère. C’est un système utilisé dans le routage statique car même si les intérêts changes dynamiquement la technique de routage reste inchangée. NDN (named data network) est un protocole qui utilise cette manière de distribution de données.

1. **Pourquoi Pub/Sub** :

Pub/Sub est un modèle centré courtier dans lequel il n’y a pas de saturation de plage d’adressage. Il est adapté aux systèmes distribués et aux systèmes ayant en communication plusieurs milliers d’autres appareils fixe ou mobile comme Internet of Things (IoT). C’est une architecture pour un réseau qui ne cesse de se développer et qui accueille chaque jour de nouvel utilisateur qui échange des contenus continuellement car elle offre à ses consommateurs la possibilité d’abonnement aux données permettant aux applications grand public (consommateurs) de recevoir efficacement des mises à jour. Il est extensible, à faible consommation d’énergie et est sans serveur central ce qui assure la disponibilité des données et la tolérance aux pannes des équipements.

1. **Les protocoles qui utilisent Pub/Sub** :

Les protocoles les plus populaire utilisant le modèle de communication publication/abonnement des messages sont :

1. **Le protocole MQTT**

MQTT (Message Queuing Telemetry Transport) est un protocole léger, ouvert et simple pour transport de messagerie de publication/abonnement et Client/server. Il est conçu pour être facile à mettre en œuvre et est utilisable dans des environnements contraints tels que l’internet des objets. Il est basé sur TCP/IP, idéal pour la transmission des données utilisant une faible bande passante et est à faible consommation énergétique.

La principale caractéristique de MQTT est sa légèreté, ce protocole ne requiert que des ressources minimales et peut donc être utilisé sur de petits microcontrôleurs.

Son objectif est de faire dialoguer des équipements qui ne disposent que peu de ressources pour assurer une connexion permanente. Pour Fabien Pereira Vaz, de Paessler AG, MQTT se démarque par sa souplesse et sa simplicité de mise en œuvre, en plus d'assurer une transmission de données bidirectionnelle. Le protocole prend ainsi de l'importance dans les technologies opérationnelles de l'industrie et c’est l’un des protocoles le plus utilisé dans l’internet des objets. Il utilise un courtier pour le relais des messages. Son processus se divise en quatre étapes distinctes : connexion, authentification, communication, terminaison et les accords définit pour la garantie de livraison son sont les mêmes que AMQP qui sont : plus d’une fois, moins d’une fois et exactement une fois.

1. **NDN** :

NDN (Named Data Network) est une nouvelle architecture qui fournit une communication centrée sur les données au niveau de la couche réseau. NDN implémente un modèle de communication demande-réponse asynchrone qui dissocie naturellement les producteurs et les consommateurs de données. Il définit deux types de paquets de couche réseau : intérêt et donnée. Chaque producteur de données attribue un nom unique et sémantiquement significatif à chaque paquet de données qu'il génère. Chaque consommateur émet un paquet d'intérêt avec un nom de données ou un préfixe de nom, qui est transmis en fonction du nom (préfixe). Pour chaque intérêt reçu, les producteurs NDN utilisent des stratégies de transfert pour décider où transmettre l'intérêt en tenant compte des politiques d'utilisation de la table de transfert et de la mesure des décisions de transfert précédentes.

1. **Pub/Sub sur google Coud** :

C’est un ensemble d’application implémenté comme un ensemble de service connecté par la messagerie asynchrone. Le producteur crée le message et l’envoie pour un topic laquelle est appelé ressource, le message est stocké jusqu’à l’acquittement de tous les consommateurs. Pour retirer ce message une application consommatrice doit créer un abonnement pour ce topic et Pub/Sub Cloud livre le message à tous ce qu’y sont abonnés. L’acquittement des messages permettent la non livraison des messages plus d’une fois. Cette communication peut s’effectuée entre un producteur et plusieurs consommateurs, plusieurs producteurs et un seul consommateur et plusieurs producteurs et plusieurs consommateurs. Une application productrice peut-être n’importe quelle application capable d’envoyer une requête http aux APIs du google. Quant aux applications consommatrices, elles doivent être non seulement capable d’envoyer une requête https, mais aussi accepter la requête POST de https. C’est un modèle idéal pour un environnement ou il y a beaucoup d’échange de message.

1. **Le protocole AMQP**

AMQP (Advanced Message Queuing Protocol) est un protocole réseau ouvert (open source) pour la messagerie. Il agit au niveau de la couche application autorisant un échange de message entre deux parties avec utilisation d’une file d’attente (Queuing). Il est standard et peut être utilisé pour la mise en place d’une topologie d’échange de type publication/abonnement. L’objectif principale de ce protocole est de mettre en place un serveur de messagerie orienté message en utilisant la fille d’attente et l’interopérabilité pour permettre une production/consommation sans un problème de langage de programmation voire de système d’exploitation et il est efficace dans l’interconnexion des objets hétérogènes. AMQP est plus adapté aux situations exigeantes : une faible connexion, la fiabilité communication et une importante sécurité. Il est utilisé dans activeMQ par la fondation Apache pour fournir son implémentation dans plusieurs langages tels que : C, C++, java... et d’autre plateforme comme IBM (MQLight) [3].

1. **Pub/Sub et Client/serveur** :

Pub/Sub est une architecture basée sur un composant central courtier (peut être un serveur). C’est un modèle dans lequel un consommateur n’a pas besoin de connaître les producteurs de donnée et un producteur n’a pas non plus besoin de connaître un consommateur de donnée.

Client/serveur est un modèle de communication d’internet de requête/réponse. Un client demande des informations ou des services et le serveur répond par les données ou services demandés.

Pub/Sub

consumer

producer

requête

serveur

client

consumer

réponse

consumer

producer

|  |  |
| --- | --- |
| Pub/Sub | Client/server |
| Centré sur le contenu | Centré sur l’adresse IP |
| Routage basé sur le contenu | Routage basé sur IP |
| Élimination du DNS | Impossible de fonctionner sans DNS |
| Distribué les données | Partages de donnée |
| Pas de serveur central pour les informations | Toujours un serveur central utilisé |
| Qos plusieurs vers plusieurs | Qos un client et un serveur |
| Communication asynchrone | Communication synchrone |
| Les deux entités peuvent commencer la communication | Client debute la communication |

1. **Protocole à implémenter AMQP** :

Parmi les protocoles cités nous implémentons AMQP car c’est un protocole qui assure l’échange des messages chiffrés et assure l’interopérabilités entre les applications (il n’impose pas aux producteurs et consommateurs de comprendre le même langage de programmation). Il utilise l’architecture publication/abonnement utilisé dans les systèmes de messageries ainsi que pour la gestion des périphérique IoT**.**

Une communication AMQP est un transfert de message pair-à-pair entre les nœuds d’un réseau. Elle est constituée d’un ensemble de conteneur qui peut contenir plusieurs nœuds. Un conteneur peut être un broker ou application cliente. Un nœud peut être un producteur ou un consommateur responsable d’un stockage ou d’une livraison de message.

Une communication entre deux conteneurs nécessite un établissement de connexion. Les messages envoyés lors d’une connexion sont fragmentés en un ou plusieurs trames. Une connexion est alors constituée de l’envoie d’une séquence de trame ordonnée car la réception de la dernière trame implique que les premières trames sont déjà reçues.

1. **Architecture AMQP** :
2. **Les entités de communications** :
3. **Nœuds** : les nœuds sont les entités responsables de production, de stockage, de livraison ou de consommation des messages. Il peut être un producteur, un consommateur ou une queue.

* **Source ou producteur** : est un nœud qui crée et envoie les messages.
* **Cible ou consommateur** : est un nœud qui récupère et traite les messages.
* **File d’attente (queue)** : est la destination ciblée pour le stockage des messages. Elle peut être plusieurs, chacune ayant un identifiant unique qui la différencie des autres et est réalisée à l’aide d’une mémoire. Elle permet de distribuer les messages aux différents consommateurs. Un message est stocké dans une file d’attente à condition que les règles de comparaison entre l’identifiant de la file d’attente de celui du message soient respecté.

Broker

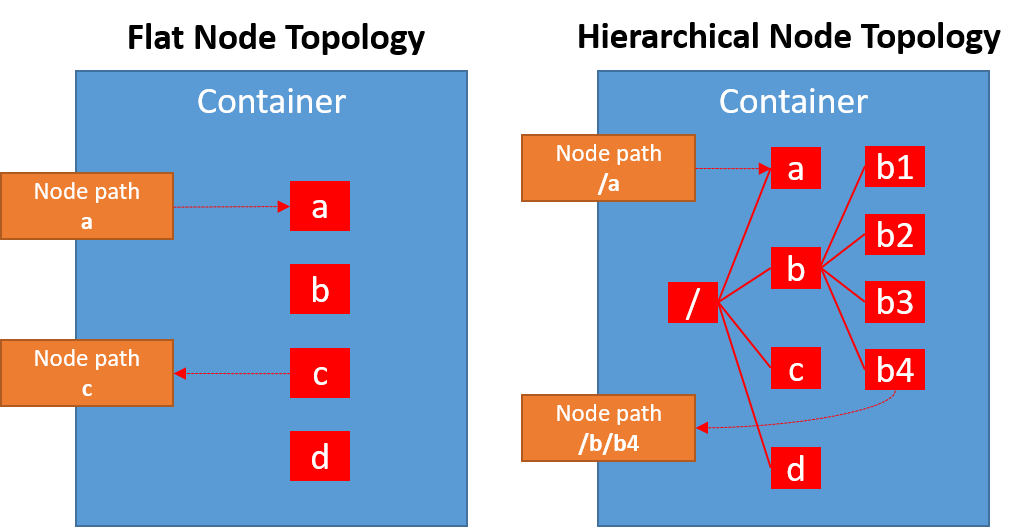
Queue1

**Exchange**

Queue2

Queue3

1. **Conteneurs** : est un ensemble de nœud ou une application client avec ces producteurs et/ou ces consommateurs, ou un broker avec ces queues. Le placement des nœuds dans un conteneur peut être plat ou hiérarchique [3] :



1. **Unité de communication AMQP** :

Une trame est l’unité de base échangé de AMQP et chaque trame est composée de trois parties distinctes :

Header

Body

Extended header

* Frame header (8 bits) : il est de taille fixe et contient la version protocole AMQP utilisé.
* Extended header (taille variable) : Le traitement de cette zone dépend du type de trame car il contient des données supplémentaires.
* Frame body (taille variable) : c’est le corps de la trame, il contient les données à transmettre et peut avoir différentes formes telles que : *open/close* (la trame d’établissement ou de fin de connexion), *begin/end* (établissement ou finalisation d’une session), *attach/dettach* (concerne les liens), *transfert* (pour envoyer un message), *disposition* (utilisé pour informer le récepteur des changements d’état de livraison).

1. **Connexion entre pair** :

Tout connexion AMQP commence par les échanges TCP, ensuite chaque pair doit envoyer un paquet *open* qui indique ses limites et ses capabilités pour l’établissement de deux chaines de connexion unidirectionnelle appelé chacun un lien AMQP. Cet échange TCP concerne les informations sur la version du protocole et de la sécurité et ce lien peut être par exemple producteur-queue-consommateur. Ces deux liens de sens opposé constituent une connexion bidirectionnelle avec le control de flow d’où une session et sont obtenues.

Session

conteneur

conteneur

Sessi sessionon

lien

lien

La session est éphémère c’est-à-dire qu’après la déconnexion les pairs doivent refaire le même processus.

Pour la fermeture de la connexion les deux points de communications doivent envoyer un paquet *close* avec un code de fermeture de session. Pour ça chaque pair doit attendre avec un délai raisonnable pour la réception de ce paquet pour éviter toute erreur car c’est le dernier paquet envoyé.

1. **Contrôle de flow** :

Le point de terminaison de session attribue à chaque trame sortante un identifiant de transfert implicite à partir d'une séquence. Chaque point de terminaison de session maintient l'état suivant pour gérer les trames de transfert entrantes et sortantes. Le contrôle de flow détermine l’identifiant du prochain paquet entrant attendu et sortant, le nombre de paquet entrant et sortant tout en respectant la taille de la fenêtre (la taille maximale de paquet recevable par le consommateur) ce qui facilite la communication avec une faible bande passante. Cette connexion étant l’envoie d’une séquence ordonnée de trame il vérifie le numéro de séquence car le contrôle de flow est basé sur le nombre de paquet échangé. Les deux points de communication incrémentent l’identifiant du paquet précédemment reçu comme l’identifiant du prochain paquet à envoyer et peut aussi décrémente la taille de la fenêtre.

Il exige aussi le contrôle de flow des liens car il possède un état qui indique si le lien est disponible pour l’envoie des messages et pour le mettre à jour l’envoyeur doit le modifier pour le receveur.

1. **Etat de la connexion et de la réception des messages** :
2. **Etat de la connexion** :

Tout connexion AMQP commence par les échanges TCP,

Il s’agit des étapes principales indiqué par les paquets de comment la communication se passe entre les pairs :

* Open/Close : cette étape implique l’envoie des paquets d’initiation ou de fermeture de la connexion ce qui veut que la connexion est soit ouverte ou fermée.
* Begin/End : pour l’établissement ou la fermeture de la session les pairs s’envoi ces paquets. A ce stade la session est ouverte ou fermée et la session fait partie de la connexion TCP établie.
* Attach/Detach : indique l’attachement ou le détachement d’un lien a une session.
* Send/Receive : indique l’envoi des paquets et leurs réceptions sur les deux liens établis.

1. **Etat de la réception des messages** :

Les messages reçus peuvent être de différente forme tel que :

* *Accepted* : le message est bien reçu et est traité par le consommateur
* *Rejected* : les message invalide que le consommateur ne peut pas traiter donc il est rejeté.
* *Released* : les message invalide qui doivent re-envoyé pour qu’ils puissent être traité.
* *Modified* : les messages non traités car ils ont été modifiés.
* *Received* :

1. [**La qualité de services (QoS)**](https://www.ibm.com/docs/en/ibm-mq/9.2?topic=reliability-message-delivery-amqp#q123795___qos) :

AMPQ est efficace, portable, multiport et réputé par sa fiabilité de livraison. Il fonctionne bien dans les environnements multi client et obéit à des règles strictes, garantissant l'interopérabilité des clients de différents fournisseurs. Ce protocole offre trois manières de communication producteur/consommateur grâce au principe de réglage automatique (*setled*) de ces entités de communication. Ces trois manières sont :

* ***at-most-once*** : un seul envoi avec la possibilité qu'il soit manqué ; si un production choisit ce réglage il supprime sa copie du message avant que les files en acquitte. De même si un consommateur choisit cette option la file n’attend pas l’acquittement de leur part avant de le jeter.

consommateur

producteur

Transfert (delivery\_tag,setled, …)

* ***at-least-once*** : livraison garantie avec la possibilité de duplication de message, contrairement au premier le producteur et la file d’attente attend respectivement l’accusé de réception de la file d’attente et le consommateur.

consommateur

producteur

Transfert (delivery\_tagsetled , …)

ACK (setled, …)

* ***exactly-once*** : livraison unique garantie. Car avant l’envoie du message le producteur et le consommateur s’envoient quelque message et acquittement.

1. **Fonctionnement** :

AMQP est un protocole de communication entre les applications à travers un réseau. Il permet une communication asynchrone car l’émetteur et le consommateur n’ont pas à agir au même rythme et le consommateur n’est pas non plus obligé de traiter l’information et d’en accuser la réception. Il s’appuie sur le message broker pour la distribution des messages aux différentes destinations. Les messages étant stockés dans une file d’attente, le producteur peut ainsi travailler sans avoir de période d’inactivité.

Les messages sont distribués les différentes files d’attentes en utilisant les identifiant de ces files et le *binding* des messages.

Broker

Queue

Echange

Consommateur

Producteur

Queue

Queue

**Exemple d’application** :

Cet exemple met en place un producteur, un consommateur et un serveur RabbitMQ avec le langage Java en localhost.

**RabbitMQ** : est un server (ici broker et gestionnaire de queues) localhost de simulation du protocole AMQP qui écoute par défaut sur le port 15672 pour les versions 3.x et pour la manipulation de ce serveur à travers une interface utilisateur, nous avons activés les plugins de gestions en saisissant dans le terminal RabbitMQ : « rabbitmq-plugins enable rabbitmq\_management ».

Publie Consomme

**Producteur**

**Consommateur**

**RabbitMQ**

Vérifie

Dans cet exemple nous avons utilisés rabbitmq-server-3.8.16.exe, Erlang (otp\_win64\_23.3.exe) et Eclipse (nécessite l’importation des deux librairies amqp-client.jar et slf4j-simple.jar).

Apres avoir lancer le programme producteur (**Prod**), il produit un message et ce message sera stocké dans la queue définie et visible dans **RabbitMQ**. Le programme consommateur (**Cons**) va ensuite vérifier la présence d’un message dans la même queue et le consomme (affiche le message trouvé) s’il y en a.

**NB** : Une exécution continue de **Prod** produit plusieurs messages qui seront stockés dans la même queue.

1. **Transmission des messages aux consommateurs** :

Après la production des messages, chaque message est transmis au broker par les producteurs et le broker à son tour livre les messages aux consommateurs. Le broker AMQP est constitué de l’*exchange* et les queues. Le transfert de message entre *l’exchange* et les queues s’effectue par la correspondance entre la *routing\_key* du message et le *binding\_key* des files d’attente.

Le protocole AMQP comporte quatre types *d’exchange*, ces quatre types deroutages sont :

* Direct exchange ou unicast : le message est livré à une seule file d’attente si la *routing\_key* correspond au *binding\_key* donc un seul destinateur. C’est une communication one-to-one.
* Fanout exchange ou broadcast : c’est une communication one-to-many car ce type d’exchange ne fournit pas de *routing\_key* donc le message est livré à toutes les files d’attentes.
* Topic exchange ou multicast : c’est comme direct exchange sauf que la comparaison entre les *keys* ne concerne qu’une partie des keys. C’est possible d’avoir une communication many-to-many.
* Header exchange : utilise l’en-tête des messages à la place de la routing\_key. Les que contient cet entête seront comparées au *binding\_key*.

Par défaut le c’est-à-dire default exchange, le *binding\_key* d’une queue est utilisé comme le *routing\_key* ce qui le stockage direct des messages dans cette queue.

Si un ou plusieurs consommateurs s’abonne à une queue pour un type de message. Le message leurs serait livré une fois son stockage dans la file d’attente.

1. **Inconvénient :**

Ce protocole n’est pas compatible à ses versions précédentes et utilise beaucoup de bande passante.

1. **Sécurité**:

AMQP est protocole d’encodage binaire qui utilise un protocole de sécurisation SSL/TLS et de l’authentification SASL. Chacun de ces deux protocoles consiste en quelque échange de frame header suivies des négociations de TLS ou de SASL. Cette négociation est un échange de certificat de sécurité pour les échanges sur réseau informatique.

SSL/TLS permet de crypter les informations sur les différentes chaînes pour assurer la confidentialité des données.

SASL est un mécanisme d’authentification qui vérifie les différents accès aux informations et peut être par exemple (nom d’utilisateur et mot de passe).

1. **Conclusion** :

Les services de communication de nouvelle génération doivent pouvoir coopérer pour répondre à des besoins spécifiques tout en gardant leur autonomie. Les réseaux sont mis en place notamment dans le but de transférer des données d’un système à un autre ou de fournir des ressources partagées des serveurs, des bases de données ou d’une imprimante. Ceci nécessite de maîtriser leurs architectures et de partager ces architectures. Pub/Sub est une norme qui préconise comment les équipements devraient communiquer entre eux tout en permettant aux systèmes ouverts capable de respecter son architecture d’être apte à échanger des informations avec d'autres équipements hétérogènes et issus de constructeurs différents. Les protocoles de cette architecture fourni une connexion logique qui est à la base de ces échanges entre ces systèmes informatiques indépendants afin de répondre aux contraintes des applications et systèmes actuelles mais aussi d’en soulever d’autre.

[1] Cudennec, Loïc. "Merging the publish-subscribe pattern with the shared memory paradigm." *European Conference on Parallel Processing*. Springer, Cham, 2018.

[2] <https://www.ibm.com/docs/en/ibm-mq/9.2?topic=reliability-message-delivery-amqp>

[3] <https://dzone.com/refcardz/amqp-essentials#section-1>

Lien que vous m’aviez fourni

<http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf>