



RabbitMQ

Formation Ministère de l'intérieur



Sommaire

- 1. Introduction – MOM**
- 2. RabbitMQ**
- 3. Types d'Exchange**
- 4. Installation**
- 5. Configuration.**
- 6. Sécurité / Policy.**
- 7. RabbitMQ et le clustering.**
- 8. Trace et debug.**
- 9. Administration.**
- 10. Best practices**

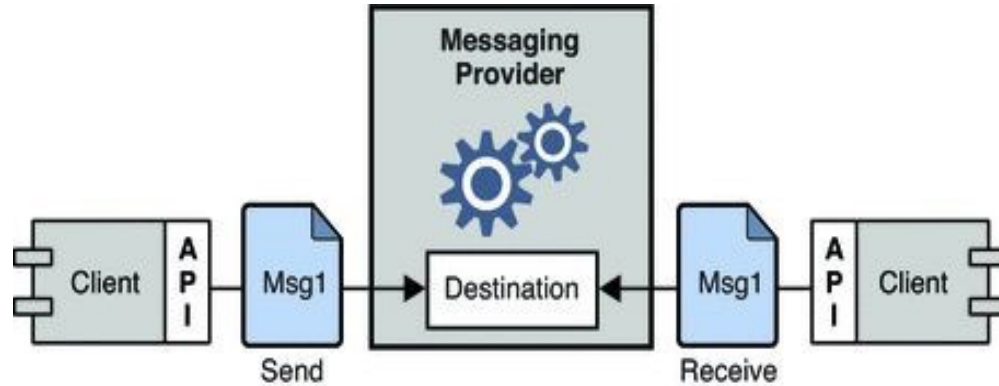
1-Introduction



Sommaire Chapitre

- 1. Présentation MOM.**
- 2. Installation.**
- 3. Exemple de code client.**

Présentation MOM



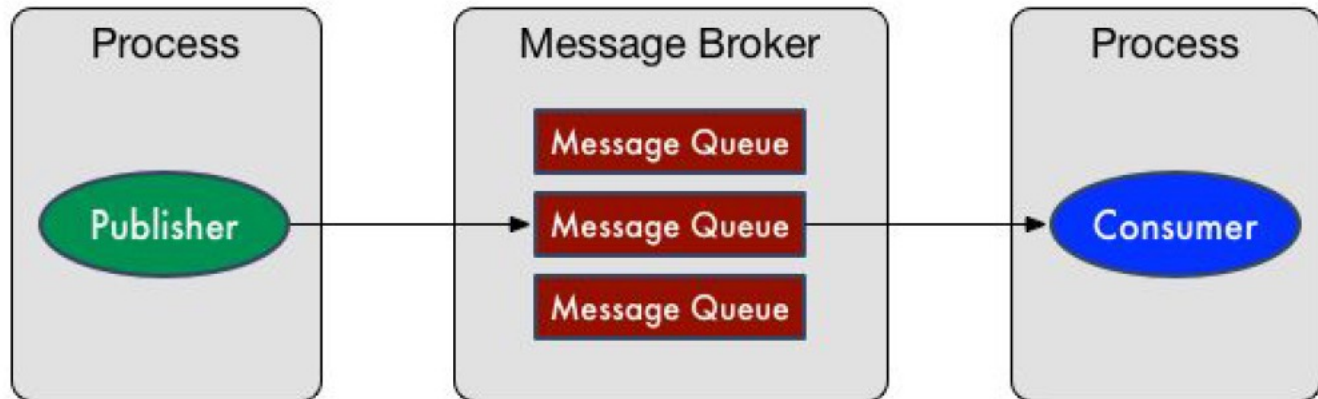


Message Queue

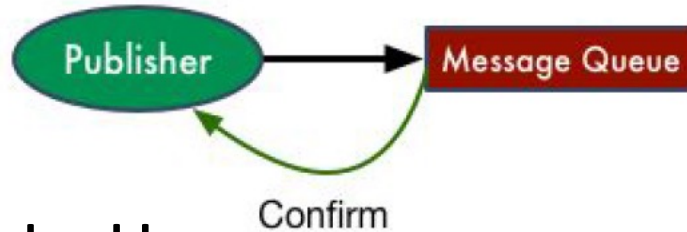


En général basé sur : First In First Out FIFO

Message Broker



Fiabilité d'envoi d'un message



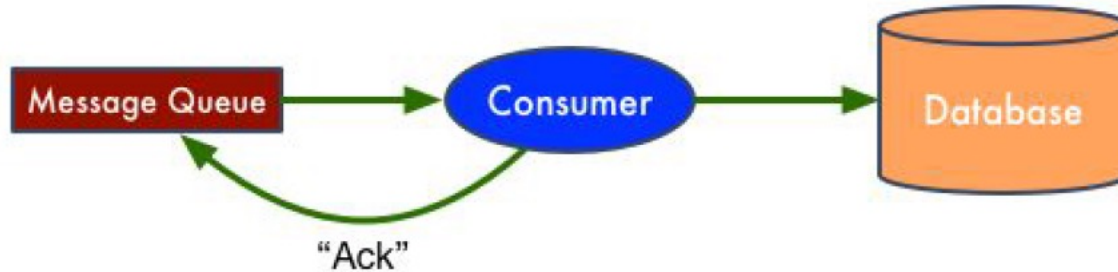
Une Queue durable :

- Sauvegarde les messages dans un espace de stockage permanent.
- Important pour la récupération

Une Queue non durable :

- Garde les messages dans la mémoire,
- Assure un débit plus rapide des messages
- Certains messages n'ont pas besoin de persistance

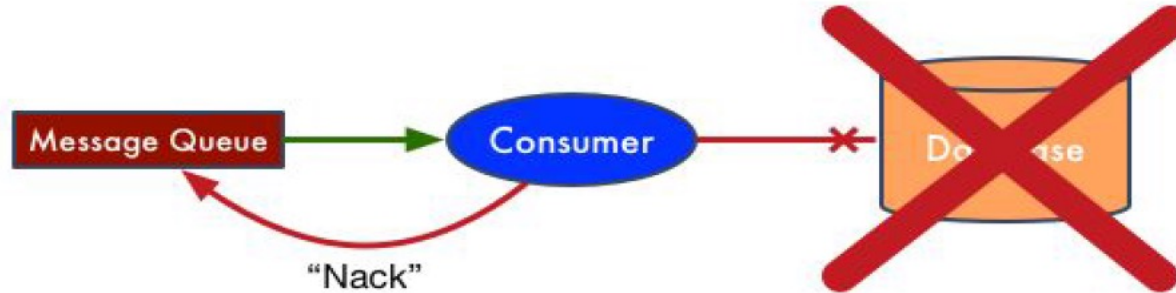
Diffusion fiable d'un message pour le consommateur



"Ack" ⇒ Acknowledgement

- Message supprimé de la file d'attente seulement après "Ack" du consommateur,
- Utilise "Ack" pour garantir que le message est consommé

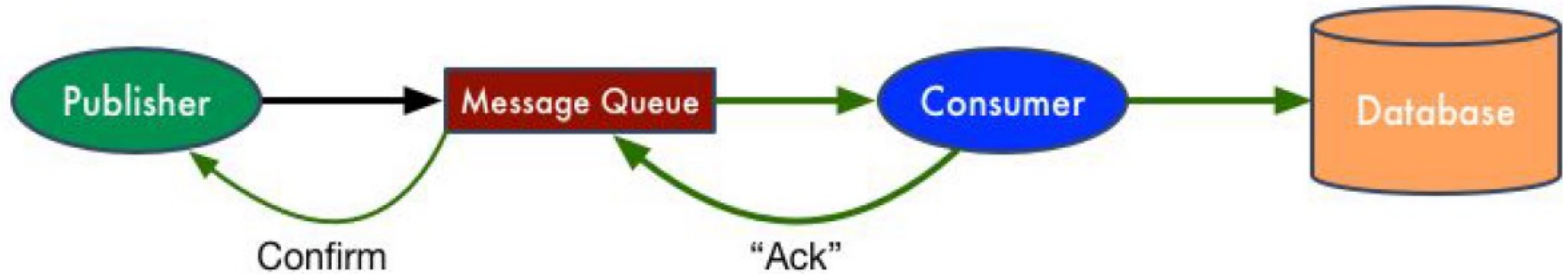
Si Problème ?



“Nack” ⇒ No Acknowledgement

- Utilise “Nack” Quand le consommateur échoue de traiter le message.
- Le message est redilivré à un autre consommateur après le “Nack”

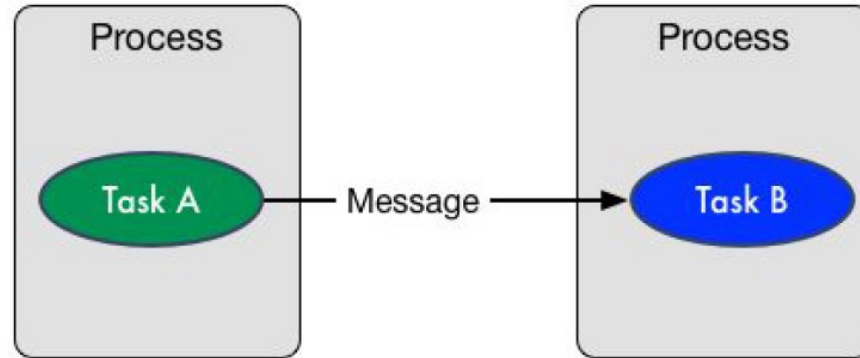
Fiabilité de la messagerie



- **Ack** ⇒ Acknowledge message
 - "Nack" ⇒ No Acknowledgement
 - Utiliser le protocole "Ack/Nack" pour assurer la fiabilité de messagerie.

Quels problèmes à résoudre ? "Message Queues" >>

Processus étroitement liés :



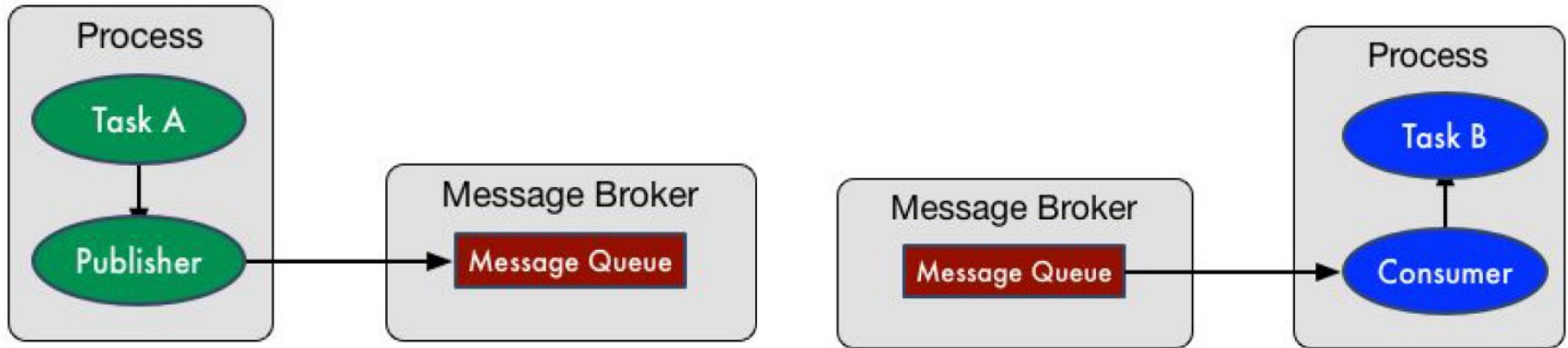
Task A ne peut pas envoyer un message au Task B si "Task B" n'est pas disponible pour le recevoir.

- **HYPOTHESES:** Task A n'a pas besoin d'une réponse en temps réel de la Task B



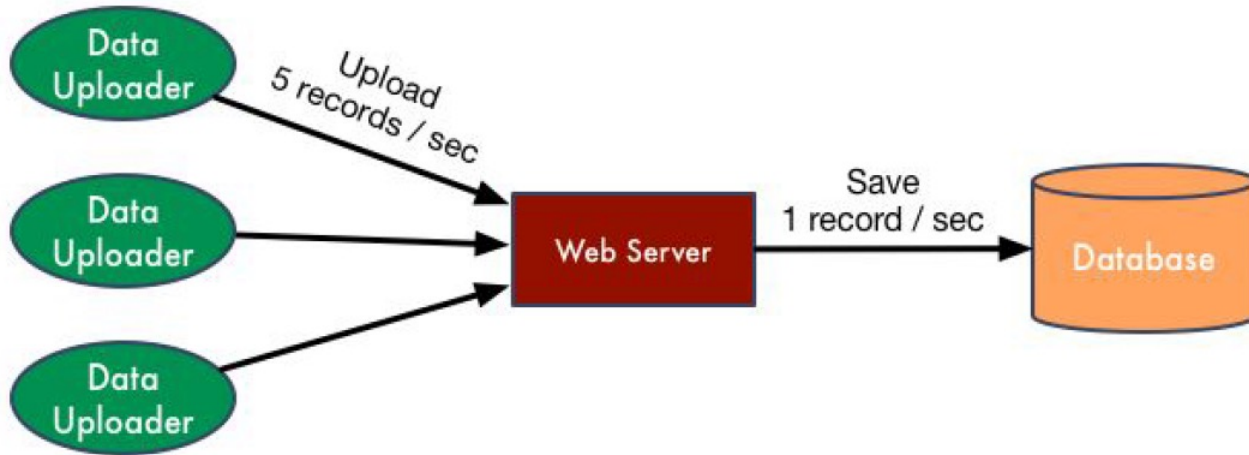
Quels problèmes à résoudre ? "Message Queues" »

Solution :



Quels problèmes à résoudre ? "Message Queues" »

Problème de lenteur du consommateur :

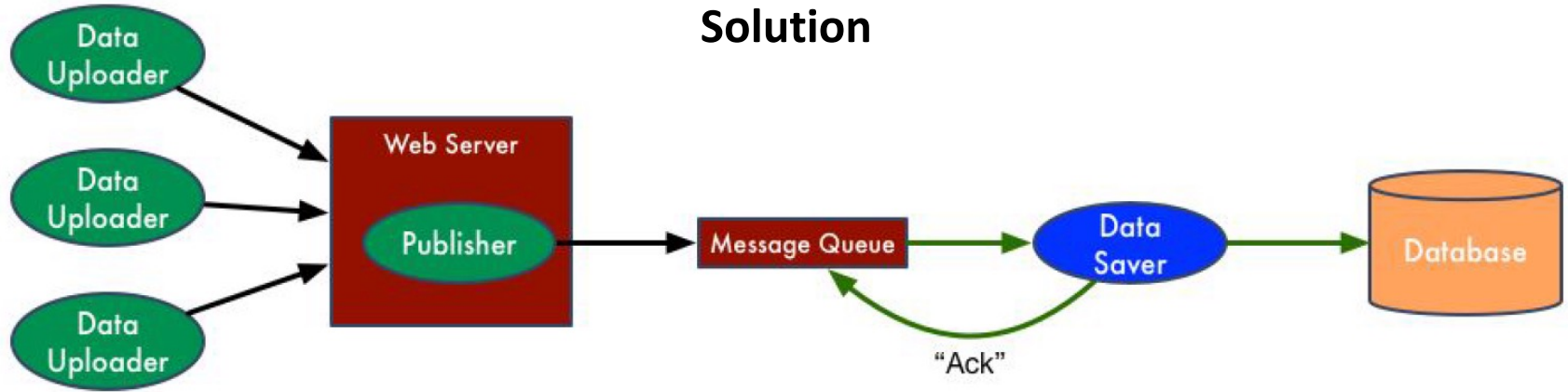




Quels problèmes à résoudre ?

« Message Queues »

Solution

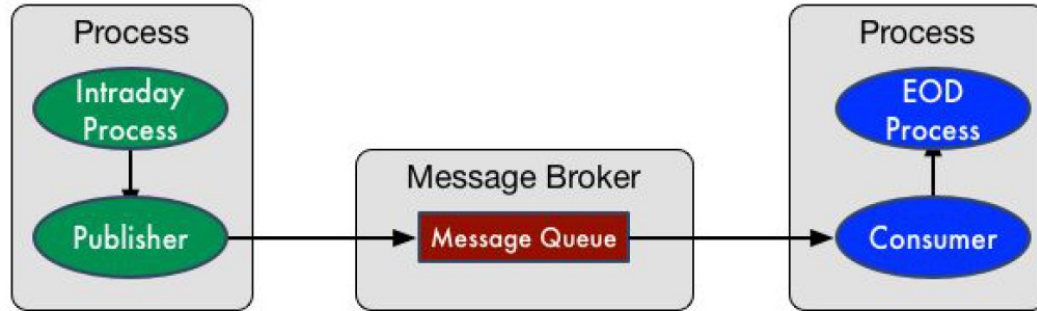


Message Queue agit comme un "buffer" pour un producteur Rapide/Consommateur lent.

Quels problèmes à résoudre ?

« Message Queues »

Traitement en Batch



- Le processus intraday s'exécute plusieurs fois au cours de la journée.
- Le processus EOD (End of Day) s'exécute une fois à une heure prédéfinie dans la journée.



Quels problèmes à résoudre ? "Message Queues" »

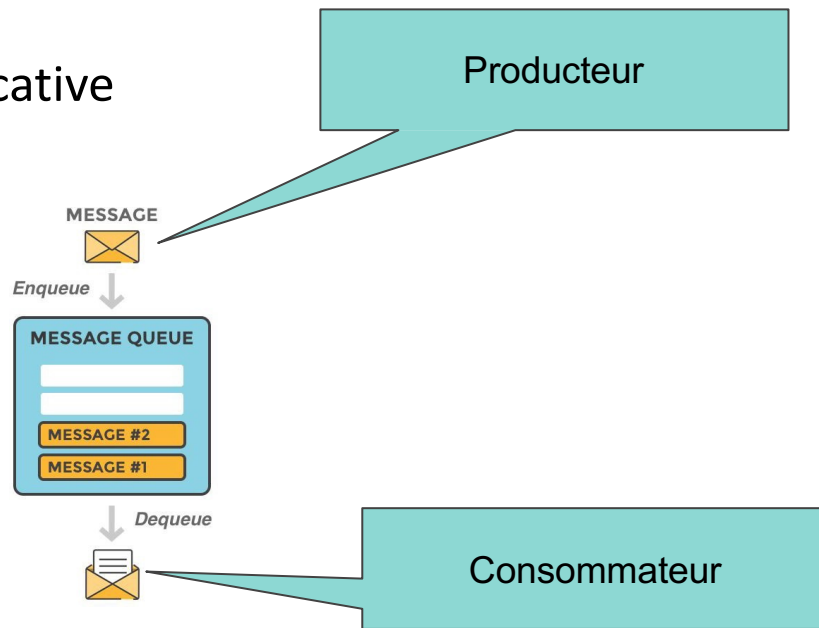
Exemple d'un site d'e-commerce :

- Un site e-commerce prend les commandes utilisateur et les stocke dans une file d'attente de messages.
- « **Message Queue** » agit comme un espace de stockage "en attente" fiable.
- Le processus EOD rassemblera toutes les commandes utilisateur en attente et les envoie en masse au centre de traitement des commandes.



Présentation MOM #1

Définition d'une messagerie applicative





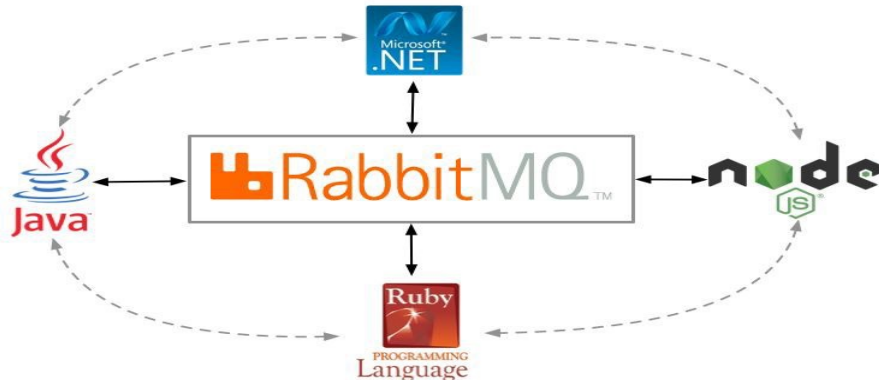
Présentation MOM #2

Le broker la base des MOM



RabbitMQ

- RabbitMQ a été développé en 2006 par Rabbit Technologies Ltd.
- RabbitMQ est un broker de messages se basant sur le standard AMQP afin d'échanger avec différents clients.
- Open source codé en langage de programmation Erlang





Oui utilise RabbitMQ ?

1. Instagram / 10,000+ concurrent connections
2. VMWare - Utilise RabbitMQ dans la virtualisation des produits
3. RedHat's Cloud Services - Utilise RabbitMQ pour coordonner les opérations internes.
4. JPMorgan, NSF, NASA, RedHat, OpenStack, AT&T, et autres...



AMQP : Advanced Message Queuing Protocol

- Le protocole AMQP sur lequel est basé RabbitMQ a pour but d'offrir un système d'échange totalement interopérable entre les différents acteurs.
- Contrairement à JMS, AMQP n'est pas une API, mais est un protocole d'échange tel que HTTP ou SMTP.
- AMQP tire son origine du besoin de normer un système d'échanges de messages totalement asynchrone qui de plus s'abstrait totalement de l'implémentation du broker ou du client.
- Il est tout à fait possible **d'utiliser un broker AMQP en Erlang**, sur lequel **un client java** va envoyer des messages qui seront consommés par un **second client en PHP ou Python**.



Erlang



Erlang: C'est le langage d'implémentation et l'environnement d'exécution de RabbitMQ.

Erlang est un langage de programmation, supportant plusieurs paradigmes :

- concurrent
- Temps réel
- distribué.

Erlang a été créé par Ericsson.

Initialement propriétaire, il est publié sous licence Open Source depuis 1998. Erlang est très utilisé dans les produits Telecom.



Format Message





Format Message

Message Properties

routing_key

ContentEncoding

ContentType

CorrelationId

DeliveryMode

Expiration

MessageId

ReplyTo

Timestamp

...Others..

Message Headers

Map<String, Object> headers;

Values can be simple data types:

String

Long

Integer

Boolean

Double

Store user specific data outside
of the message body



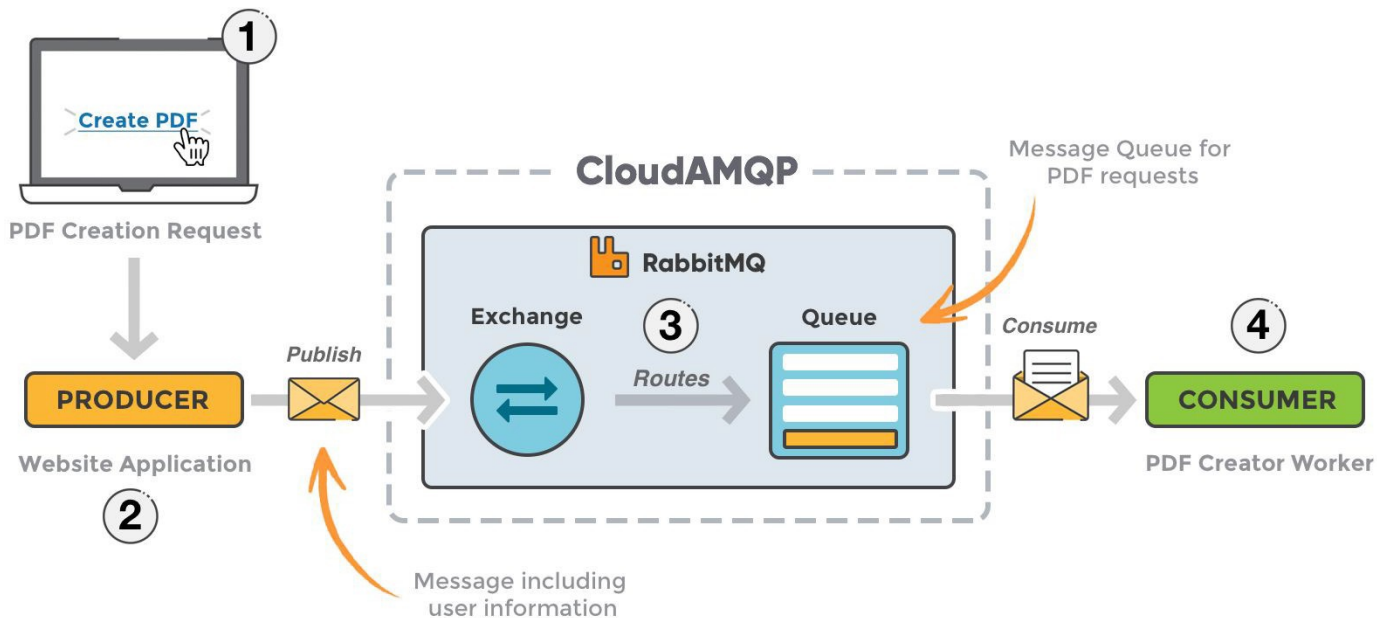
Format Message

Message Body

`byte[] body;`

- Représente string ou binary data
- Utiliser plutôt des messages courts. En effet, les messages transitent via le réseau et les grands messages peuvent poser un problème de congestion.

Présentation MOM #3

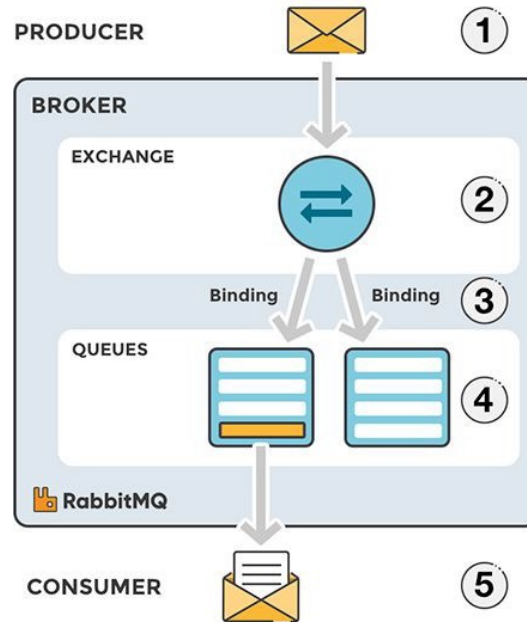




Présentation MOM #4

1. L'utilisateur envoie une demande de création de PDF à l'application Web.
2. L'application Web (le producteur) envoie un message à RabbitMQ.
3. Un **Exchange** accepte le message le dirige vers la **Queue** de messages.
4. Le créateur de PDF (le consommateur) reçoit le message et démarre le traitement du PDF.

Présentation MOM #5





Présentation MOM #6

1. Le **Producteur** publie un message vers un **Exchange**. il est nécessaire de spécifier le type d'Exchange.
2. L' Exchange reçoit le message et à en charge de l'acheminement vers la Queue. Le routage est réalisé en fonction des différents attributs du message, tels que la clé de routage, des propriétés,
3. Un Binding doit être réalisé entre l'Exchange et la Queue.
4. Les messages restent dans la file d'attente jusqu'à ce qu'ils soient consommés.
5. Le consommateur traite le(s) message(s).



Présentation MOM #7

- Le modèle AMQP comporte quatre types d'échange, qui conduisent à quatre types de routage différents. Ils sont les suivants :
 - **Direct.**
 - **Fanout.**
 - **Topic.**
 - **Headers.**



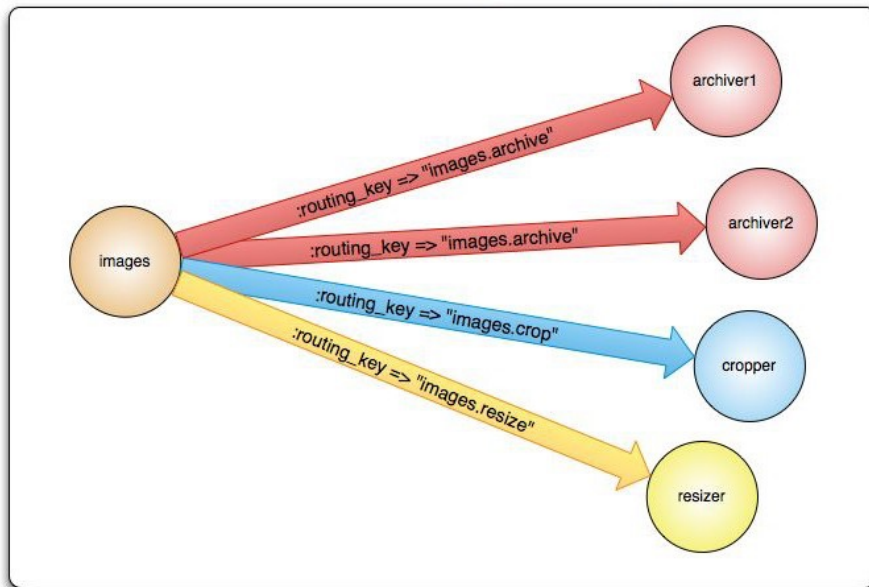
Binding /Routing Key

- **Un système de binding/routing key est implémenté par RabbitMQ :**
 - Chaque binding pourra définir sa propre binding key qui sera une chaîne de caractère identifiant un mapping précis.
 - Les messages pourront de leur côté définir une routing key qui servira à définir au travers de quel binding le message devra passer.
 - Une routing key est simplement un identifiant écrit sous la forme de noms séparés par des points.



Présentation MOM #8

Direct exchange routing



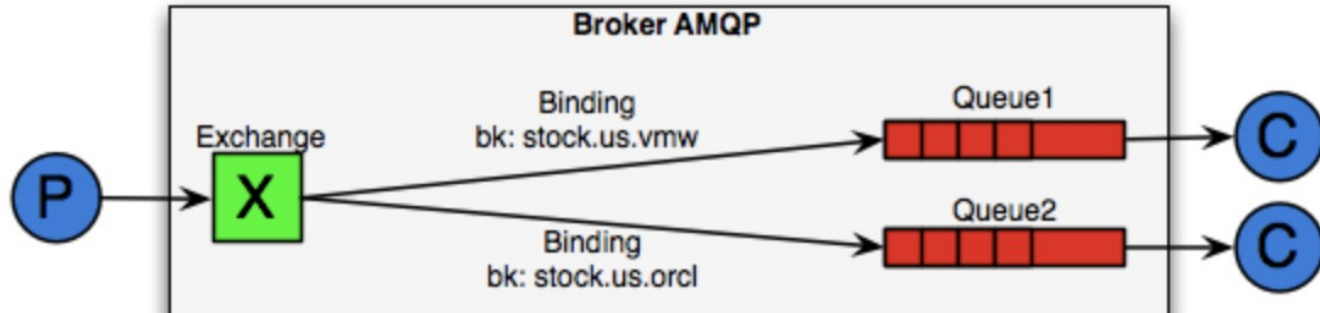
Présentation MOM #9

Direct :

La routing key sera comparée aux binding keys de chaque binding disponible;

si l'un correspond alors le message sera transféré uniquement sur la queue représentée dans ce binding.

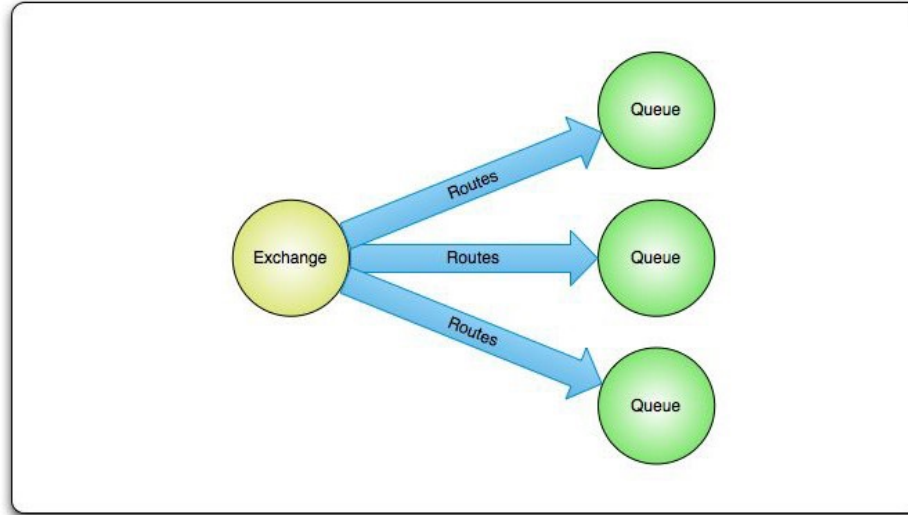
Dans le cas où la routing key ne correspondrait à aucune des possibilités, alors le message serait simplement ignoré.





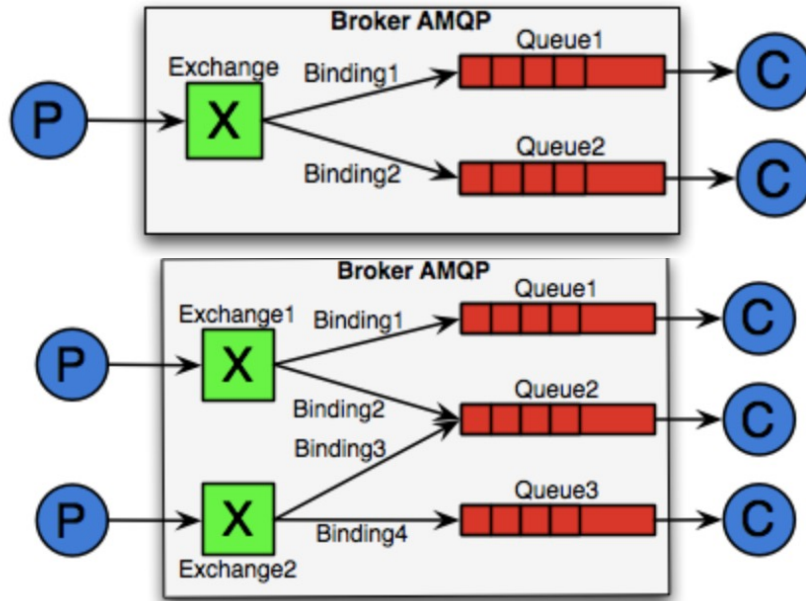
Présentation MOM #10

Fanout exchange routing



Présentation MOM #11

Fanout: un **Exchange** fanout achemine les messages vers toutes les files d'attente qui lui sont liées.





Présentation MOM #12

Topic (Les wildcards) :

L'Exchange **Topic** effectue une correspondance générique entre la routing key et le binding key en utilisant des wildcards dans les noms des clefs.

> Le premier type de wildcard : *

Permet de représenter un seul et unique élément dans la routing-key.

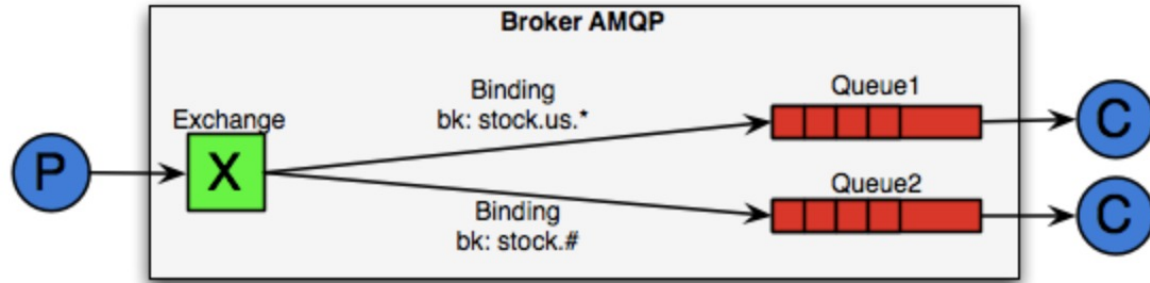
Par exemple une binding key *stock.us.** pourrait représenter toutes les routing-key commençant par *stock.us.* et se terminant par un unique élément.

Exemple : *stock.us.vmw* et *stock.us.orcl* seraient valides, *stock.us.test.assert* ne le serait pas.

Présentation MOM #13

Le deuxième type de wildcard :

- Utilisable pour représenter une suite de plusieurs éléments.
Par exemple la binding key stock.# peut représenter stock.us.vmw, stock.us.test.Assert ou stock.fr.ead.



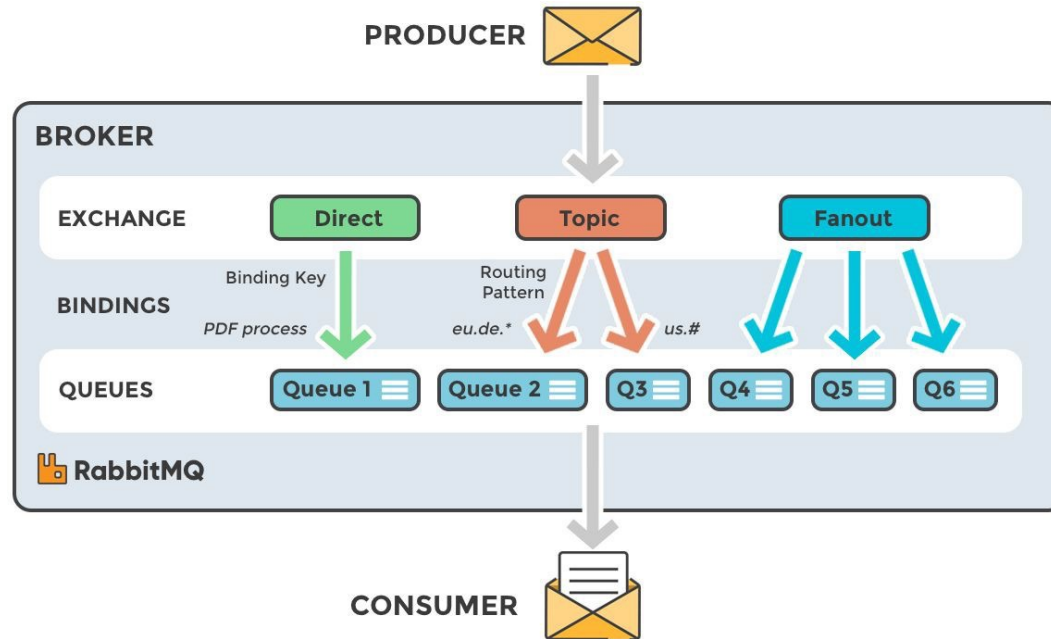
- Les différents wildcards peuvent figurer n'importe où dans la binding key,
- Ce système d'échange est appelé le mode topic, il permet de mettre en place un échange en 1:N basé sur des règles définies dans les binding keys.



Présentation MOM #14

Headers: l'Exchanges Headers utilisent les attributs d'en-tête (header) du message pour effectuer le routage.

Présentation MOM #15





Présentation MOM #16

Les concepts à retenir sont:

- Producer.
- Consumer.
- Queue.
- Message.
- Connection.
- Channel.
- Exchange.
- Binding.
- Routing key.
- AMQP.
- Users.
- Vhost, virtual host.



Présentation MOM #17

Producer : application qui envoie les messages.

Consumer : application qui reçoit les messages.

Queue : tampon qui stocke les messages.

Message : informations envoyées par le producteur à un consommateur via RabbitMQ.

Connection : une Connection est une connexion TCP entre votre application (client) et le courtier RabbitMQ.

Channel : Un canal est une connexion virtuelle à l'intérieur d'une connexion. Lorsque vous publiez ou utilisez des messages d'une file d'attente, tout se fait sur un canal.



Présentation MOM #18

Exchange: reçoit les messages des producteurs et les place dans les files d'attente en fonction de règles définies par le type d'échange. Pour recevoir des messages, une file d'attente doit être liée à au moins un échange.

Binding: une liaison est un lien entre une file d'attente et un échange.

Routing key : la clé de routage est une clé utilisée par l'échange pour décider de l'acheminement du message vers les files d'attente. La clé de routage est identique à une adresse.



Présentation MOM #19

Users : Il est possible de se connecter à RabbitMQ avec un nom d'utilisateur et un mot de passe. Des autorisations peuvent être attribuées à chaque utilisateur, telles que les droits de lecture, d'écriture et de configuration dans l'instance. La configuration des utilisateurs (autorisation) doivent être spécifiés sur des hôtes virtuels spécifiques.

Vhost, virtual host : un hôte virtuel permet de séparer les applications utilisant la même instance RabbitMQ. Différents utilisateurs peuvent disposer de privilèges d'accès différents pour vhost et différentes files d'attente. Des échanges peuvent être créés. Ils n'existent donc que dans un seul vhost.

Il y a un virtual host par défaut. Dans une configuration standard, le virtual host par défaut est “/”



Installation





Installation #1

L'installation est possible par:

- Utilisation des gestionnaires de paquets.
 - lien: <https://www.rabbitmq.com/download.html>
- Via Docker.
 - lien: https://hub.docker.com/_/rabbitmq/



Installation #2

Exemple d'installation Linux distribution type debian

1. `sudo apt-get install -fy erlang-nox python-pip git-core python-setuptools git-core`
2. `wget https://www.rabbitmq.com/releases/rabbitmq-server/vX.X.X/rabbitmq-server_X.X.X-X_all.deb`
3. `sudo dpkg -i rabbitmq-server_X.X.X-X_all.deb`

X.X.X-X est la version de RabbitMQ.



Structure File System RabbitMQ



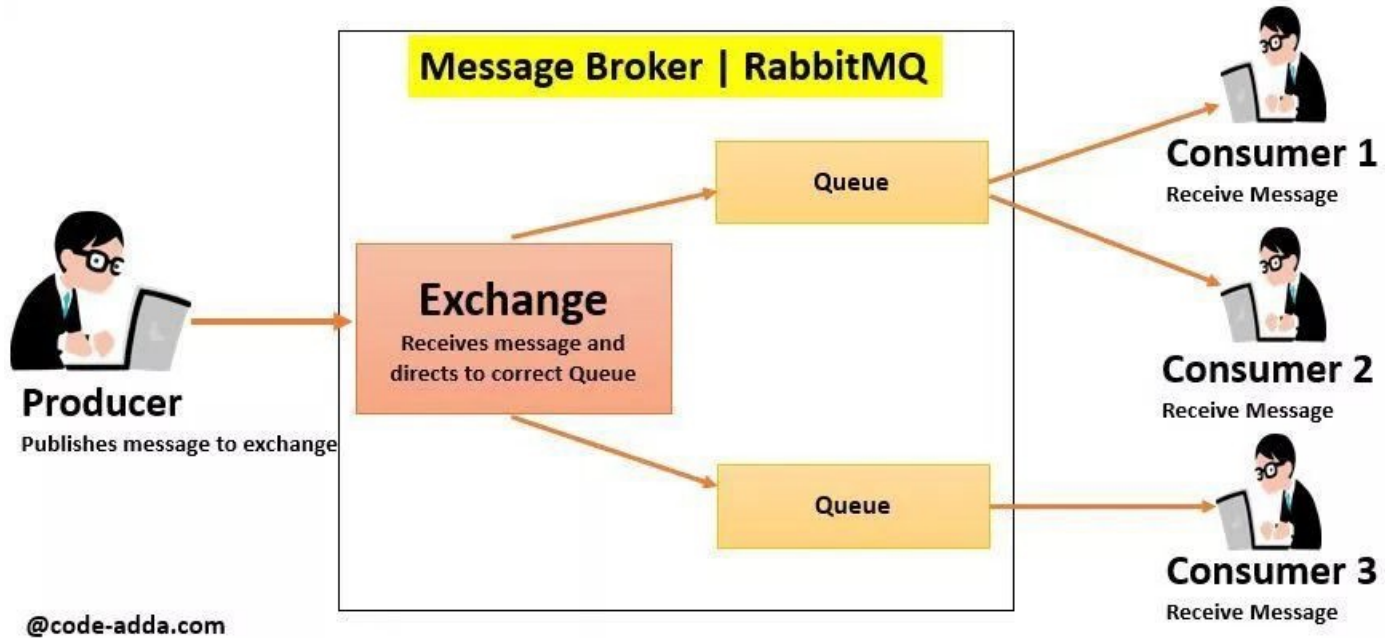


Structure File System RabbitMQ #1

Répertoire	Description
/etc/rabbitmq/rabbitmq.config	Config file
/var/lib/rabbitmq/mnesia/rabbit@mqmaster	Database directory (nom du node)
var/log/rabbitmq/rabbit@mqmaster.log	Log file

MNESIA est une base de données développée en ERLANG.

Exemple code-client





Exemple code-client #1

- Samples officiels de RabbitMQ
 - lien: <http://www.rabbitmq.com/getstarted.html>

4-Configuration.



Sommaire Chapitre

- Configuration.
 - fichier
- rabbitmqctl.
- rabbitmqadmin.
- rabbitmq-plugins.



Configuration #1

La configuration est possible via:

- Fichier de configuration.
- Variables d'environnements.
- Outil rabbitmqctl.
- Outil rabbitmqadmin.
- Outil rabbitmq-plugins.



Configuration #2

Fichier de configuration

Les éléments paramétrables sont (par exemple) :

- Écouteurs TCP et autres paramètres liés au réseau.
- Alarmes sur des contraintes liées aux ressources.
- Autorisation.
- Paramètres pour le stockage des messages.
- ...

lien <https://www.rabbitmq.com/configure.html#configuration-files>



Configuration #4

- Les ports principaux sont :

Numéro de port	Protocole
● 5672	● AMQP
● 25672	● CLUSTERING
● 15672	● WEB Management



Configuration #5

Variables d'environnement.

Les éléments paramétrables sont (par exemple):

- Le nom du node.
- Les emplacements des fichiers et des répertoires.
- Les flags d'exécution depuis le shell dans le fichier `rabbitmq-env.conf`.
- ...



Configuration #6

rabbitmqctl est l'outil qui gère entre autre:

- Les hôtes virtuels.
- Les utilisateurs et les autorisations.

lien <http://www.rabbitmq.com/rabbitmqctl.8.html>



Configuration #7

rabbitmqadmin est un autre outil d'administration.

Il est nécessaire d'activer le plugin rabbitmq_management

lien <https://www.rabbitmq.com/cli.html>



Configuration #8

rabbitmq-plugins est l'outil qui gère les actions sur les plugins.

- Activation.
- Désactivation.
- Liste.
- ...

lien <https://www.rabbitmq.com/rabbitmq-plugins.8.html>

3-Sécurité / Policy.



Sommaire Chapitre

- Introduction
- User.
- Virtual Host.
- Policy



Introduction





Intoduction

La sécurité concerne:

- Gestion des utilisateurs
 - La connexion à une instance.
 - Droit (privilèges) des utilisateurs.
- Le chiffrement de la communication.



User





User #1

Un utilisateur est un élément de rabbitMQ avec les éléments suivants:

- Un nom (ID).
- un label (rôle).

Le nom est unique pour l'instance complète.

Les label par défaut sont :





User #2

Par défaut un utilisateur ne peut pas accéder à l'instance.
Il doit être attaché à au moins un virtual host.



IMPORTANT

Le label “administrator” ne permet pas à un utilisateur d'accéder à tous les virtual host.



Virtual Host



Virtual Host #1

Un utilisateur doit être associé à un ou plusieurs Virtual Host.
Il est nécessaire de spécifier par Virtual Host les droits de chaque utilisateur.

Un utilisateur doit être au moins autorisé sur un Virtual Host.



X-arguments

X-Arguments sont des propriétés supplémentaires ajoutées à des exchanges, des queues ou des messages dans RabbitMQ.

Ils permettent de configurer des fonctionnalités avancées et des comportements spécifiques.

Utilisations courantes des X-Arguments

Déclarations de Queue :

x-message-ttl : Détermine le temps de vie d'un message en millisecondes.

x-dead-letter-exchange : Spécifie l'échange où envoyer les messages expirés ou rejetés.

x-max-length : Limite le nombre de messages qu'une queue peut contenir.

x-max-length-bytes : Limite la taille totale des messages dans une queue.

x-queue-mode : Peut être défini sur "lazy" pour stocker les messages sur disque plutôt qu'en mémoire.



Policy





Policy #1

- **Policies** sont des règles appliquées aux échanges et aux queues pour modifier leur comportement sans avoir à les recréer.
- Elles permettent de configurer des paramètres tels que la durabilité, la gestion des messages, et bien plus.
- **Pourquoi utiliser des Policies ?**
 - Centralisation : Gère les configurations depuis un seul endroit.
 - Flexibilité : Applique des changements à de multiples ressources sans interruption de service.
 - Automatisation : Facilite la gestion automatique des ressources.
- **Types de Parameters dans les Policies**
 - x-message-ttl : Temps de vie des messages.
 - x-max-length : Nombre maximum de messages.
 - x-max-length-bytes : Taille maximale totale des messages.
 - x-dead-letter-exchange : Exchange de Dead-Lettering.



Policy #2

Exemple de Policy

```
rabbitmqctl set_policy my_policy "^my_queue" \  
'{"x-message-ttl":60000, "x-dead-letter-exchange":"dlx_exchange"}'
```

Nom de la Policy : **my_policy**

Pattern : `^my_queue` (s'applique à toutes les queues commençant par "my_queue")

Arguments :

x-message-ttl : Messages expirent après 60 secondes.

x-dead-letter-exchange : Messages expirés ou rejetés sont envoyés à `dlx_exchange`.

Points Clés

Application Globale : Les policies peuvent être appliquées à plusieurs queues/exchanges via des expressions régulières.

Facilité d'Utilisation : Simplifie la gestion des configurations complexes.

Gestion Dynamique : Permet des ajustements en temps réel sans redémarrage des services.

5-Trace et debug.



Journalisation

Chemin des logs :

`/var/log/rabbitmq`

Deux manières pour redéfinir le nom et l'emplacement du fichier de log :

- Fichier de configuration
- Variable d'environnement RABBITMQ_LOGS : Si la valeur de cette variable est - , les logs vont être redirigés vers la sortie standard.

L'écriture des logs :

- Fichiers logs
- Sortie standard

Pour journaliser sur un fichier :

- `log.file = true`
- `log.console = true` journaliser sur la sortie standard



Journalisation

Niveau de log

- **log.file.level** : niveau de log à partir duquel les messages vont être stockés dans le fichier log. Un niveau inférieur sera ignoré.

Les niveaux de log existants :

- debug
- info
- warning
- error
- critical
- None

Changer le niveau de log en ligne de commande :

- `rabbitmqctl -n rabbit@target-host set_log_level debug`



Journalisation

Les catégories de log :

- **connection**
- **channel**
- **queue**
- **Default**
- **.....**

Pour définir le niveau de log pour chaque catégorie :

- `log.<category>.level = level_log`




Journalisation

Rotation des logs : 3 politiques

- **log.file.rotation.date**
- **log.file.rotation.size**
- **log.file.rotation.count**

Rotation par date :



```
# rotate every night at midnight
log.file.rotation.date = $D0

# keep up to 5 archived log files in addition to the current one
log.file.rotation.count = 5
```

```
# rotate every day at 23:00 (11:00 p.m.)
log.file.rotation.date = $D23
```

```
# rotate every hour at HH:00
log.file.rotation.date = $H00
```

```
# rotate every day at 12:30 (00:30 p.m.)
log.file.rotation.date = $D12H30
```

```
# rotate every week on Sunday at 00:00
log.file.rotation.date = $W0D0H0
```

```
# rotate every week on Friday at 16:00 (4:00 p.m.)
log.file.rotation.date = $W5D16
```

```
# rotate every night at midnight
log.file.rotation.date = $D0
```



Journalisation

Rotation par taille ou par nombre de messages :

```
# rotate when the file reaches 10 MiB  
log.file.rotation.size = 10485760  
  
# keep up to 5 archived log files in addition to the current one  
log.file.rotation.count = 5
```



Journalisation

Journalisation sur un serveur Syslog :

- Il faut définir les configurations suivantes :

log.syslog = true

log.syslog = true

log.syslog.transport = tcp (ou UDP)

log.syslog.protocol = rfc5424

log.syslog.ip = x.y.z.w

log.syslog.port = 1514



Journalisation

Journalisation des Events :

Les connexions TCP qui envoient au moins 1 byte de données seront journalisées, les connexions qui n'envoient pas de données seront ignorées. **Exemple** : health checks des nodes de load blancer.

2018-11-22 10:44:33.654 [info] <0.620.0> accepting AMQP connection
<0.620.0> (**127.0.0.1:52771 -> 127.0.0.1:5672**)

<0.620.0> : Id du processus du connexion d'Erlang

2018-06-17 06:28:40.868 [warning] <0.646.0> closing AMQP connection
<0.646.0> (127.0.0.1:58667 -> 127.0.0.1:5672, vhost: '/', user: 'guest'):
client unexpectedly closed TCP connection



Journalisation

Observer les Events :

Utiliser le plugin `rabbitmq_event_exchange`:

`rabbitmq-plugins enable rabbitmq_event_exchange`

Il faut déclarer un échange de type Topic appelé : `amq.rabbitmq.event` , Tous les events seront publiés sur cet échange avec des routing keys comme `exchange.created` et.... Vont être envoyé à cet échange.

Les Events :

Queue, Exchange and Binding events:

`queue.deleted,queue.created,exchange.created,exchange.deleted,binding.created,binding.deleted`

Connection et Channel events:

`connection.created, connection.closed, channel.created,channel.closed`

Autres,..



Journalisation

Les logs de catégorie upgrade seront stockés dans un fichier à part :

[rabbitmq@xxx upgrade.log](#)

Possible de les remettre sur le log par défaut :

`log.upgrade.file = false`



Troubleshooting et débogage

Si un problème surgit dans un node RabbitMQ, il faut commencer à vérifier les éléments ci-dessous avec des outils dédiés :

- Les logs de RabbitMQ,
- La liste des utilisateurs qui tentent de se connecter sur un vHost particulier,
- Connectivité réseau, paramètres de pare-feu, résolution d'hôte DNS



Troubleshooting et débogage

Inspection des fichiers logs :

RabbitMQ enregistre :

- Les échecs brusques de connexion TCP,
- Les timeouts,
- Les incompatibilités de version de protocole,
- Autres.



Troubleshooting et débogage

Exemple 1 : Problème des credentials

```
=ERROR REPORT==== 12-Jul-2013::16:49:03 ===  
closing AMQP connection <0.31567.1> (127.0.0.1:50458 -> 127.0.0.1:5672):  
{handshake_error,starting,0,  
    {amqp_error,access_refused,  
        "PLAIN login refused: user 'pipeline_agent' - invalid credentials",  
        'connection.start_ok'}}}
```



Troubleshooting et débogage

Exemple 2 : Versions incompatibles

```
=ERROR REPORT==== 17-May-2011::17:26:28 ===  
exception on TCP connection <0.4201.62> from 10.8.0.30:57990  
{bad_header,<<65,77,81,80,0,0,9,1>>}
```



Troubleshooting et débogage

Channel-level exceptions:

- Les exceptions liées aux channels est la source d'une partie de problèmes remontés par RabbitMQ.
- Ces exceptions sont une indication que les paramètres fournis par les applications sur (exchange, Queues,...), Quelques exemples :
 - Exchange is re-declared with attributes different from the original declaration. For example, a non-durable exchange is being re-declared as durable.
 - Queue is re-declared with attributes different from the original declaration. For example, an autodeletable queue is being re-declared as non-autodeletable.
 - Queue is bound to an exchange that does not exist.



Troubleshooting et débogage

Exceptions Réseau :

- **Connexion d'un client sur un node Rabbit MQ acceptée:**

```
telnet localhost 5672
Connected to localhost.
Escape character is '^]'.
adjasd
AMQP      Connection closed by foreign host.
```

- **Connexion d'un client sur un node Rabbit MQ refusée :**

```
telnet: connect to address [host or ip]: Connection refused
telnet: Unable to connect to remote host
```

- **Cela laisse penser à l'un des problèmes :**
 - Configuration du Firewall pour le port 5672
 - DNS setup (si hostname est utilisé)



Troubleshooting et débogage

Problème de démarrage de RabbitMQ avec l'erreur suivante :

```
ERROR: failed to load application os_mon: {"no such file or  
directory","os_mon.app"}
```

Ceci est dû au fait que le package **erlang-os-mon** n'est pas installé



Troubleshooting et débogage

Utilisation des outils de Monitoring :

L'outil rabbitmqctl :

Exemple : `sudo rabbitmqctl list_queues name messages messages_ready state
consumer_utilisation`

HTTP API

Overview de la console d'administration :

- Monitors queue length and message rates
- Monitors Erlang processes
- Monitors memory use
- Monitor connections and exchanges
- Monitor users and their permissions



Troubleshooting et débogage

Autres outils open source sont disponible sous forme de plugins que nous pouvons intégrer avec RabbitMQ :

Nagios

Munin

Zabbix



Troubleshooting et débogage

Nagios :

- **Monitoring your entire infrastructure**
- **Responding to the issues for the limits and problems**
- **Coordinating the technical team responses**

Permet d'avoir une vue 360 sur le fonctionnement de RabbitMQ

Fourni en tant que plugin pour Rabbit : Nagios-RabbitMQ

A télécharger depuis GitHub :

```
git clone https://github.com/jamesc/nagios-plugins-rabbitmq.git
cd nagios-plugins-rabbitmq/scripts
cp * /usr/lib/nagios/plugins/
```



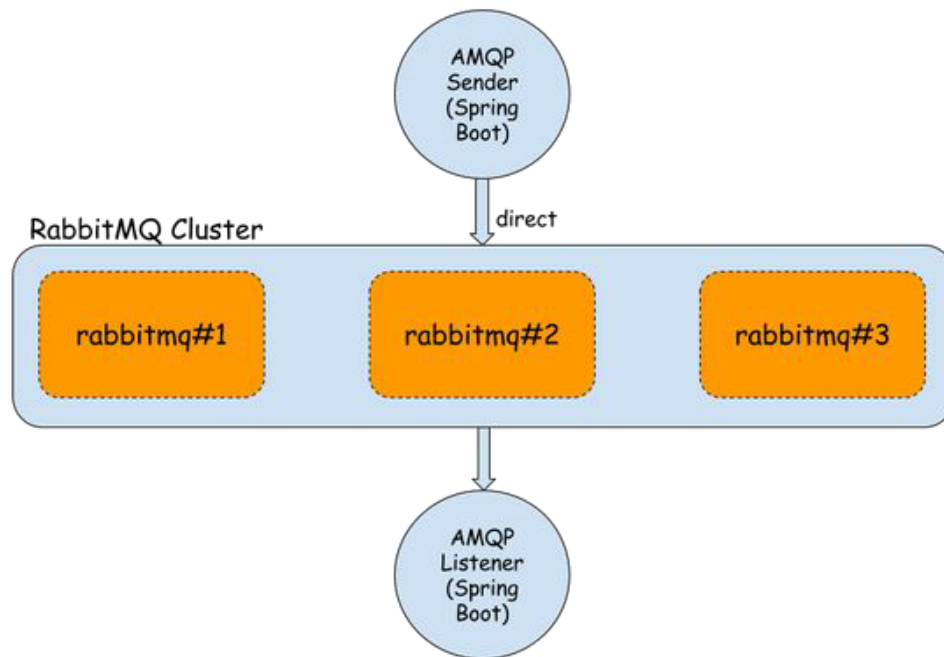
Troubleshooting et débogage

Demande un prérequis (Perl Runtime), à télécharger :
`sudo apt-get install libnagios-plugin-perl libjson-perl`

4-RabbitMQ et le clustering.



Clustering





clustering #1

Le clustering permet :

- Scalabilité : Ajouter plus de nœuds pour gérer plus de charge.
- Tolérance aux Pannes : Si un nœud échoue, les autres continuent de fonctionner.
- Haute Disponibilité : Réplication des données entre nœuds.
- **Architecture :**
 - **Nœuds RAM** : Stockent les données en mémoire pour des performances rapides.
 - **Nœuds DISK** : Stockent les données sur disque pour une persistance.
 - **Types de Réplication :**
 - **Mirroring** : Réplication des queues entre les nœuds pour la haute disponibilité



clustering

Configuration:

- Installer RabbitMQ sur chaque nœud.
- Configurer les nœuds pour se joindre au cluster
- `rabbitmqctl stop_app`
- `rabbitmqctl reset`
- `rabbitmqctl join_cluster rabbit@<autre-noeud>`
- `rabbitmqctl start_app`



clustering

association esclave #1:

- **Les node doivent être actif**
 - `/usr/sbin/rabbitmq-server -detached`
- **arrêt de l'application**
 - `rabbitmqctl stop_app`
- **Copie du cookie du maître vers le nouveau node.**
- **association**
 - `rabbitmqctl join_cluster nom du maître@nom du maître`
- **démarrage de l'application.**
 - `rabbitmqctl start_app`



clustering

Gestion des Nœuds

Voir les nœuds du cluster :

```
rabbitmqctl cluster_status
```

Promouvoir un nœud en nœud DISK :

```
rabbitmqctl change_cluster_node_type disc
```



Mise en place des Queues Mirroring :

Queues Mirroring

Configurer les politiques pour le mirroring :

```
rabbitmqctl set_policy ha-all "^" '{"ha-mode":"all"}'
```

Paramètres de mirroring :

ha-mode: Mode de haute disponibilité (all, exactly, nodes).

6-Administration.



Sommaire

1. Démarrage et arrêt des “nodes”.
2. Gestion des privilèges.
3. Les statistiques et analyse des logs.
4. Gestion des alertes (alarmes).
5. Gestion des limits.
6. Gestions des Queues.
7. Dead Letter Q.



Démarrage et arrêt des “nodes”.

Commande	Description
rabbitmqctl start	Démarrage
rabbitmqctl stop	Arrêt



Gestion des privilèges #1.

Commande	Description
<code>rabbitmqctl add_user <user id> <password></code>	Création utilisateur
<code>rabbitmqctl set_user_tags <user id> administrator</code>	Mise en place Tag pour l'utilisateur
<code>rabbitmqctl set_permissions -p / <user id> ".*" ".*" ".*"</code>	Mise en place des permissions. ".*" (Configure) ".*" (Write) ".*" (Read) -p / (virtual host où appliquer les permissions du user). Il est nécessaire de répéter <i>set_permissions</i> par " virtual host ".

Gestion des privilèges #2.

AMQP 0-9-1 Operation		configure	write	read
exchange.declare	(passive=false)	exchange		
exchange.declare	(passive=true)			
exchange.declare	(with AE)	exchange	exchange (AE)	exchange
exchange.delete		exchange		
queue.declare	(passive=false)	queue		
queue.declare	(passive=true)			
queue.declare	(with DLX)	queue	exchange (DLX)	queue
queue.delete		queue		
exchange.bind			exchange (destination)	exchange (source)
exchange.unbind			exchange (destination)	exchange (source)
queue.bind			queue	exchange
queue.unbind			queue	exchange
basic.publish			exchange	
basic.get				queue
basic.consume				queue
queue.purge				queue



Les statistiques et analyse des logs.

Exemples d'entrées dans le log:

```
=INFO REPORT==== 8-Nov-2016::15:03:26 === accepting AMQP connection <0.27270.101> (192.168.33.8:36503 ->
192.168.33.101:5672)
=INFO REPORT==== 8-Nov-2016::15:03:26 === closing AMQP connection <0.27270.101> (192.168.33.8:36503 ->
192.168.33.101:5672)
=WARNING REPORT==== 8-Nov-2016::16:54:56 === closing AMQP connection <0.27388.101> (192.168.33.1:64438 ->
192.168.33.101:5672): client unexpectedly closed TCP connection
=WARNING REPORT==== 10-Nov-2016::14:11:03 === HTTP access denied: user 'guest' - User can only log in via localhost
=ERROR REPORT==== 10-Nov-2016::14:11:03 === webmachine error: path="/api/vhosts" "Unauthorized"
=INFO REPORT==== 10-Nov-2016::16:04:49 === Adding vhost 'foo'
=WARNING REPORT==== 10-Nov-2016::16:06:07 === HTTP access denied: user 'admin' - User not authorised to access
virtual host
=ERROR REPORT==== 10-Nov-2016::16:06:07 === webmachine error: path="/api/exchanges/foo" "Unauthorized"
=INFO REPORT==== 5-Nov-2016::19:53:13 === Mirrored queue 'hello' in vhost '/': Synchronising: all slaves already synced
=INFO REPORT==== 5-Nov-2016::19:53:13 === Mirrored queue 'hello' in vhost '/': Synchronising: 0 messages to synchronise
=INFO REPORT==== 5-Nov-2016::19:53:13 === Mirrored queue 'hello' in vhost '/': Synchronising: batch size: 4096
```



Gestion des alertes (alarmes).

RabbitMQ cessera de lire les données ou d'accepter les connexions dans les cas suivants:

- Trop de mémoire utilisé (configurable).
- Pas assez de disque (configurable).

Attention dans le cas d'un environnement cluster, un problème disque ou mémoire bloquera tous les nodes du cluster.

Il est donc possible de mettre en places des alarmes pour éviter d'atteindre les limites et donc de bloquer RabbitMQ.

Les alarmes possibles sont:

- Mémoire.
- Disque.



Gestion des alertes (Memory #1).

- Il est possible de gérer des niveaux de mémoire via les options suivantes:
 - **vm memory high watermark.**
 - C'est le seuil de mémoire
 - La valeur spécifiée peut être:
 - relative.
 - `vm_memory_high_watermark.relative = 0.6`
 - absolute.
 - `vm_memory_high_watermark.absolute = 2GB`
 - La valeur par défaut est
 - `vm_memory_high_watermark.relative = 0.4`



Gestion des alertes (Memory #2).

- **vm_memory_high_watermark_paging_ratio :**
 - Seuil de mémoire ou une pagination est réalisée pour libérer de la mémoire.
 - Les valeurs sont possibles relative, absolute.

7-Best pratice.



Bests practice (Q) #1

- **Les bonne pratiques pour la mise en oeuvre des Queues.**
- **Avoir des Queues les plus petites possible.**
 - Une Queues trop volumineuse à un impacte sur la mémoire.
 - La pagination (sur disque) des Queues dégrade les performances.
 - Le redémarrage est aussi impacté par des Queues (durable) trop volumineuse car il doit régénérer les indexs.
 - Il est donc recommandé de définir une taille et un TTL par Q.
- Limiter le nombre de Queues.
- Laisser RabbitMQ définir le nom des Queues Temporaires.
- Créer des Queues de Type autodelete (quand c'est possible).
 - La libération de la Queue par le dernier client détruira la Queue.