JEUDI DE PARTAGE #2

Rendre nos applications plus performantes

26-12-2019

SOMMAIRE

SOMMAIRE

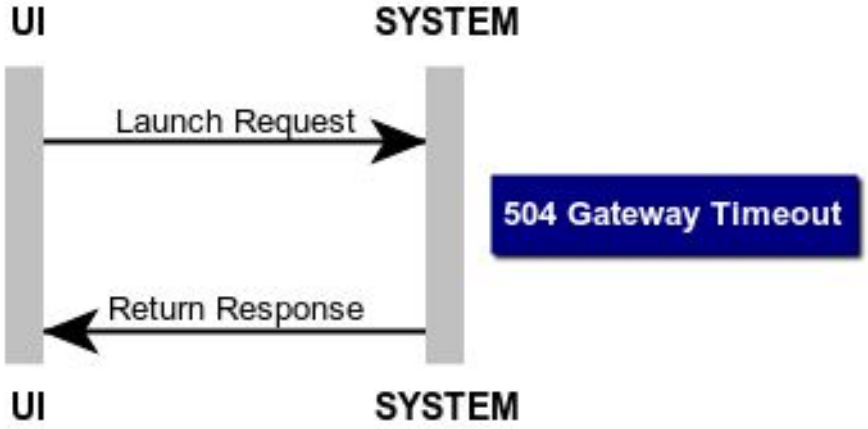
Problématiques

- Comment traiter les big data sans avoir des timeout ?
- Comment avoir des rapports instantanés (real-time) ?
- Comment accroître la performance de nos applications en appliquant le système de multithreading?

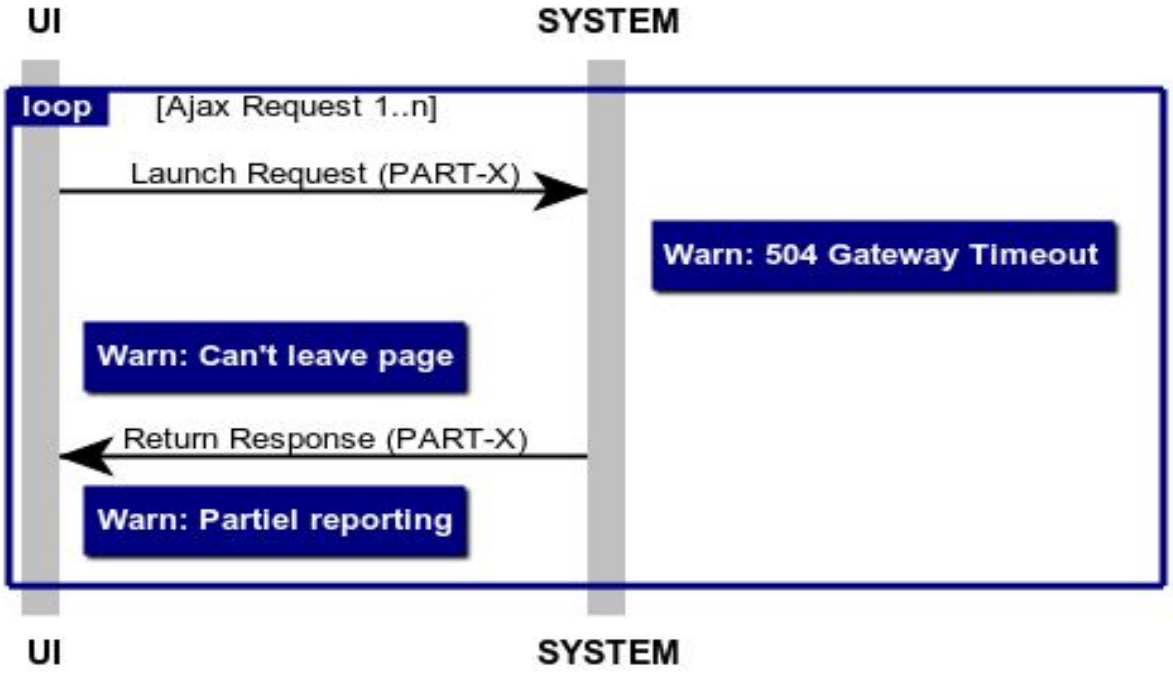
SOMMAIRE

Modèles classiques

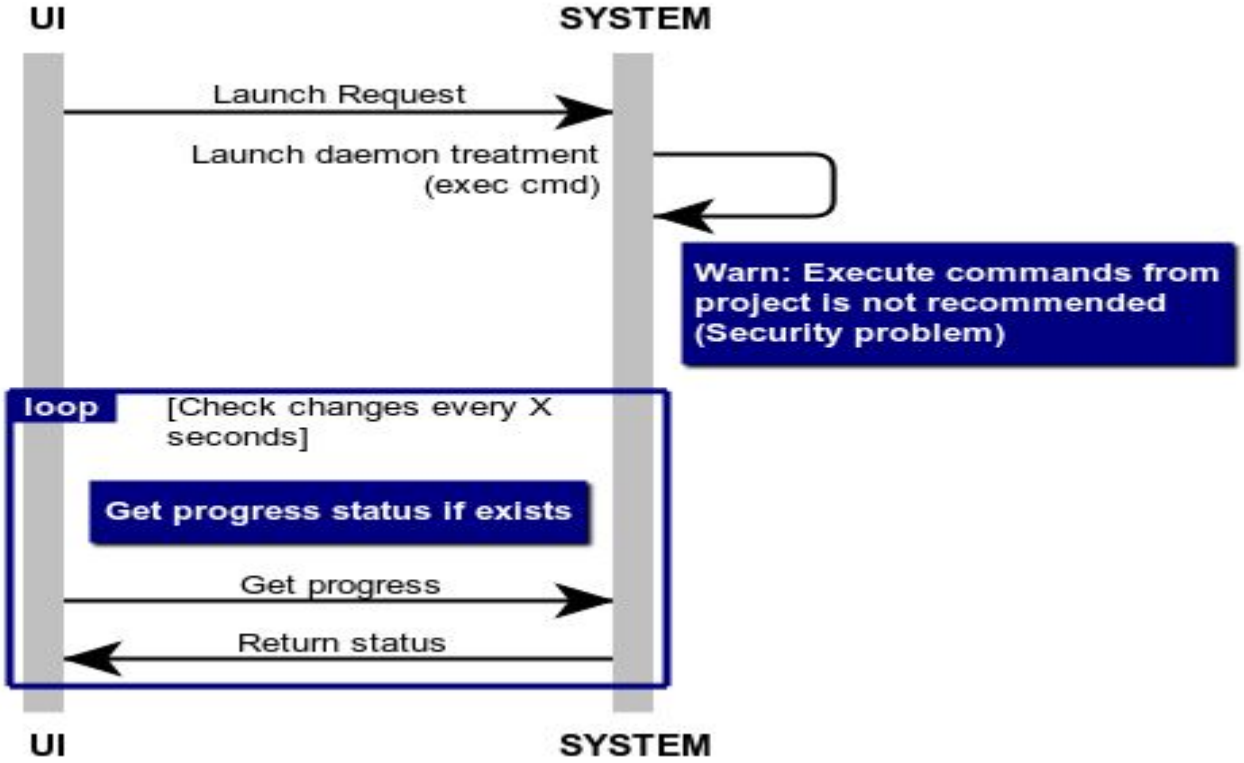
Modèle 1 : Une seule requête



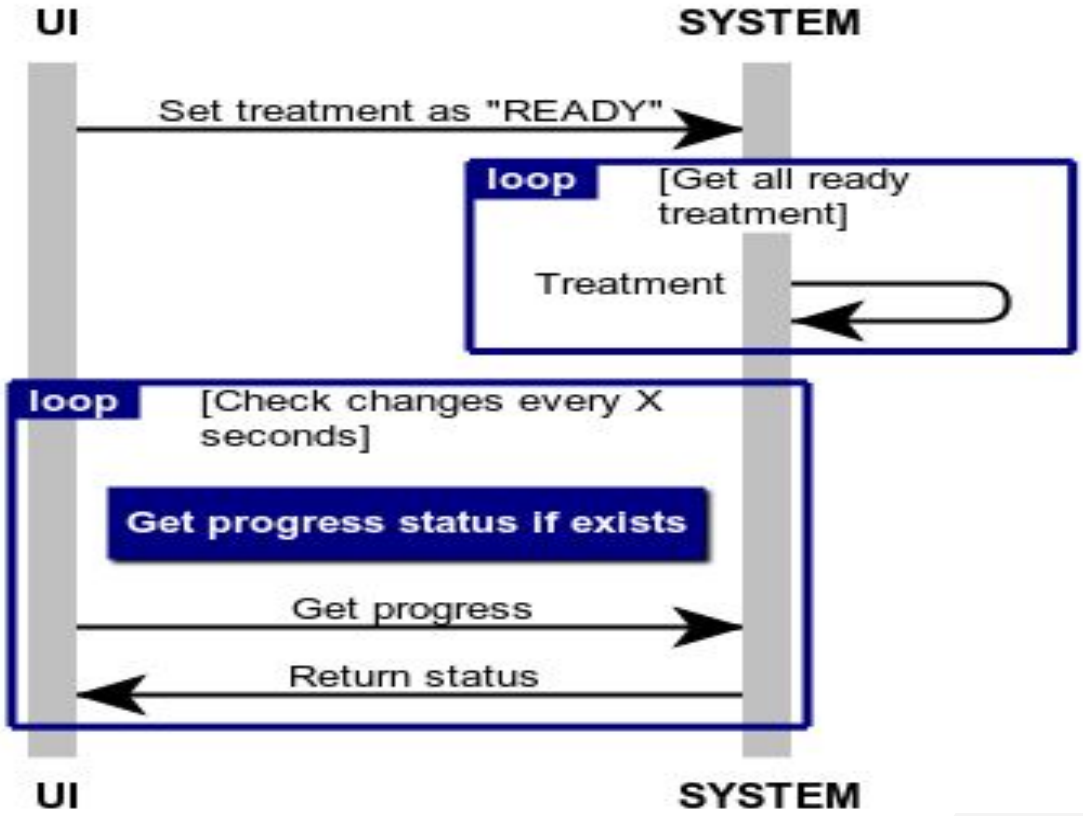
Modèle 2 : Plusieurs requêtes AJAX



Modèle 3 : Traitement démon



Modèle 4 : Traitement cronné



SOMMAIRE

Présentation de la solution | RabbitMq

RabbitMq:

- Implémentation du protocole AMQP
- Opensource
- Développé en Erlang
- Système de clustering pour la haute disponibilité et la scalabilité
- Dispose d'une architecture « pluggable » et fournit une extension pour d'autres protocoles tels que LDAP, HTTP, STOMP et MQTT
- Priorisation des messages (Quality Of Service (QOS))
- Mutualisation des serveurs (env dev/rec/preprod/prod)

Présentation de la solution | RabbitMq

Protocole AMQP:

- Protocole binaire qui vise à standardiser la communication middleware
- Advanced Message Queuing Protocol :
    - Conçu en 2006 par JP Morgan
    - Standardisé : Norme internationale ISO / IEC 19464: 2014
    - Sécurisé, Flexible, Fiable, Multiplexé

Présentation de la solution | RabbitMq

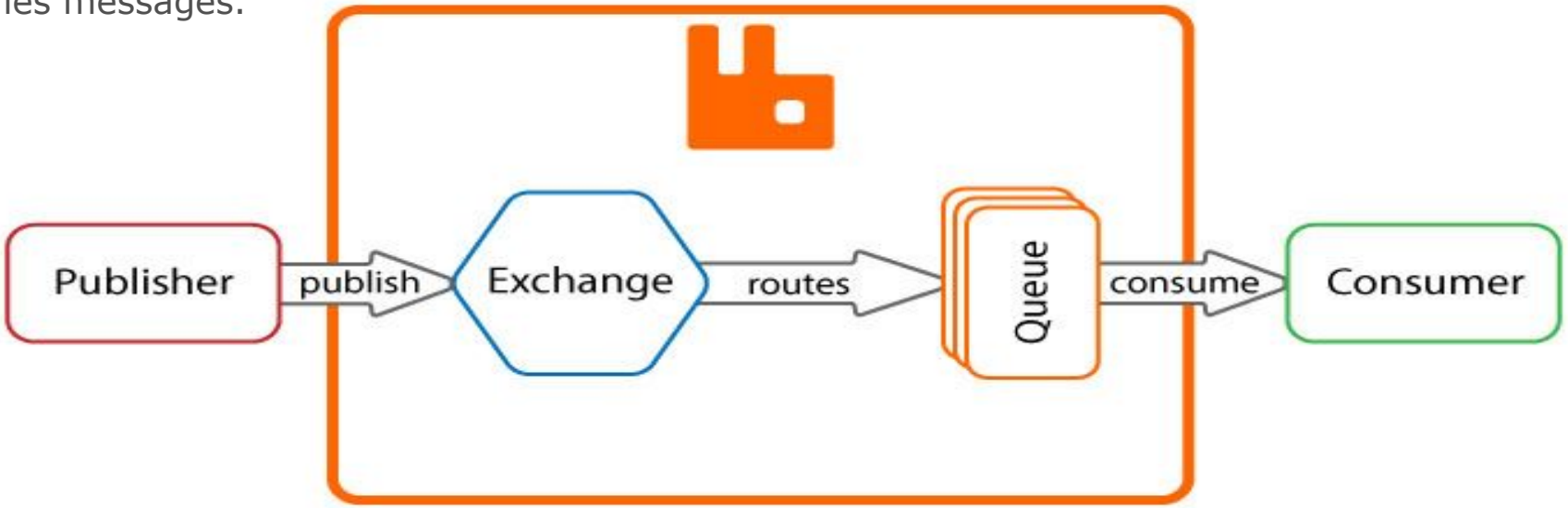
Protocole AMQP:

Il définit :

- **Exchanges:** est le broker endpoints qui se charge de la réception du message et le route vers la queue appropriée.
- **Queues** : stockent les messages destinés à un subscriber en attente qu'ils soient récupérés
- **Bindings** : règles qui lient les exchanges et les queues.

Présentation de la solution | RabbitMq

Le publisher va envoyer un message dans un exchange qui va, en fonction du binding, router le message vers la ou les queues. Ensuite un consumer va consommer les messages.



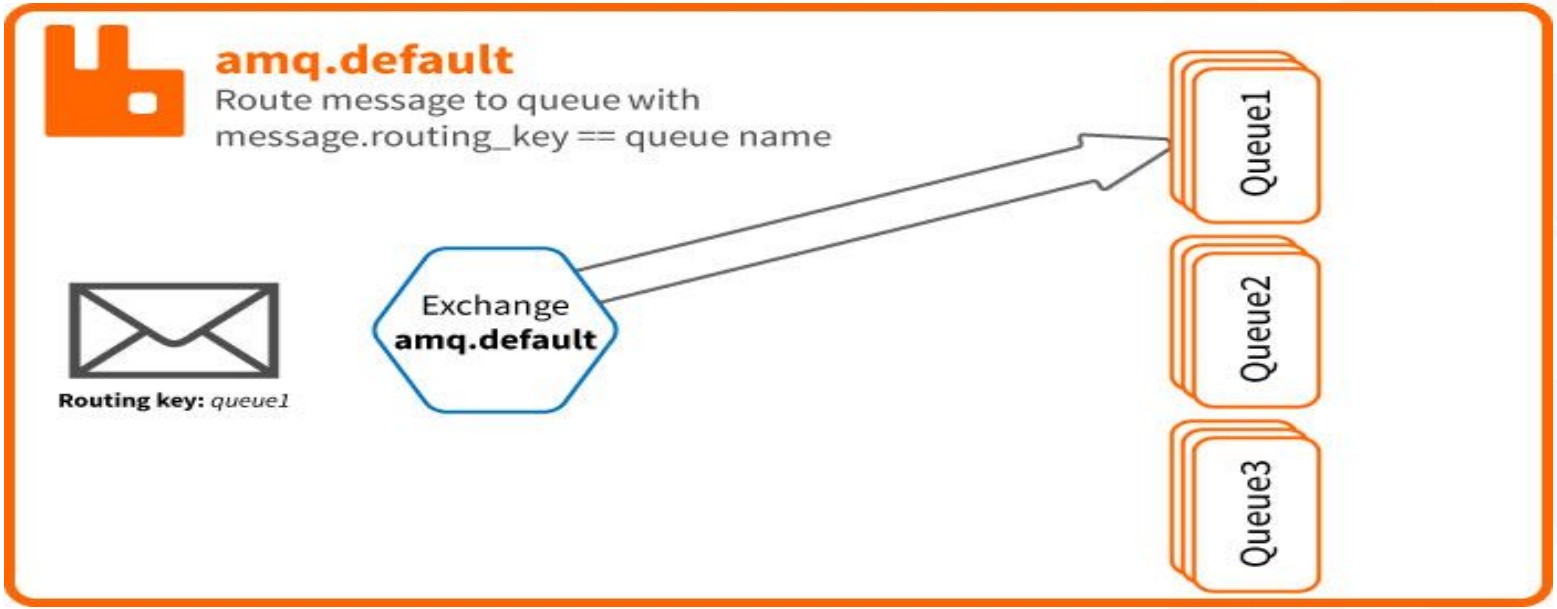
Les Exchanges:

Il existe différents types de routages définis par le type d'exchange:

- amq.default
- Type FANOUT
- Type DIRECT
- Type TOPIC
- Type HEADERS

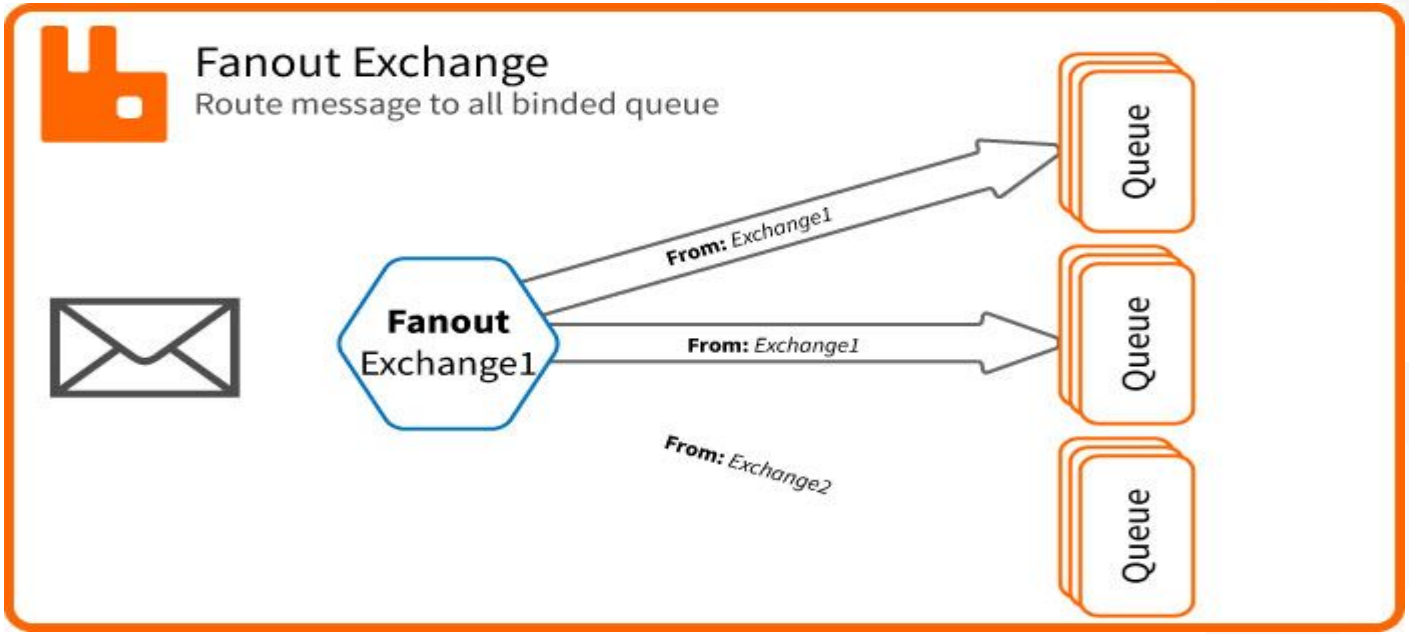
Exchange amq.default:

Cet exchange est auto bindé avec toutes les queues avec une routing key égale au nom de la queue.



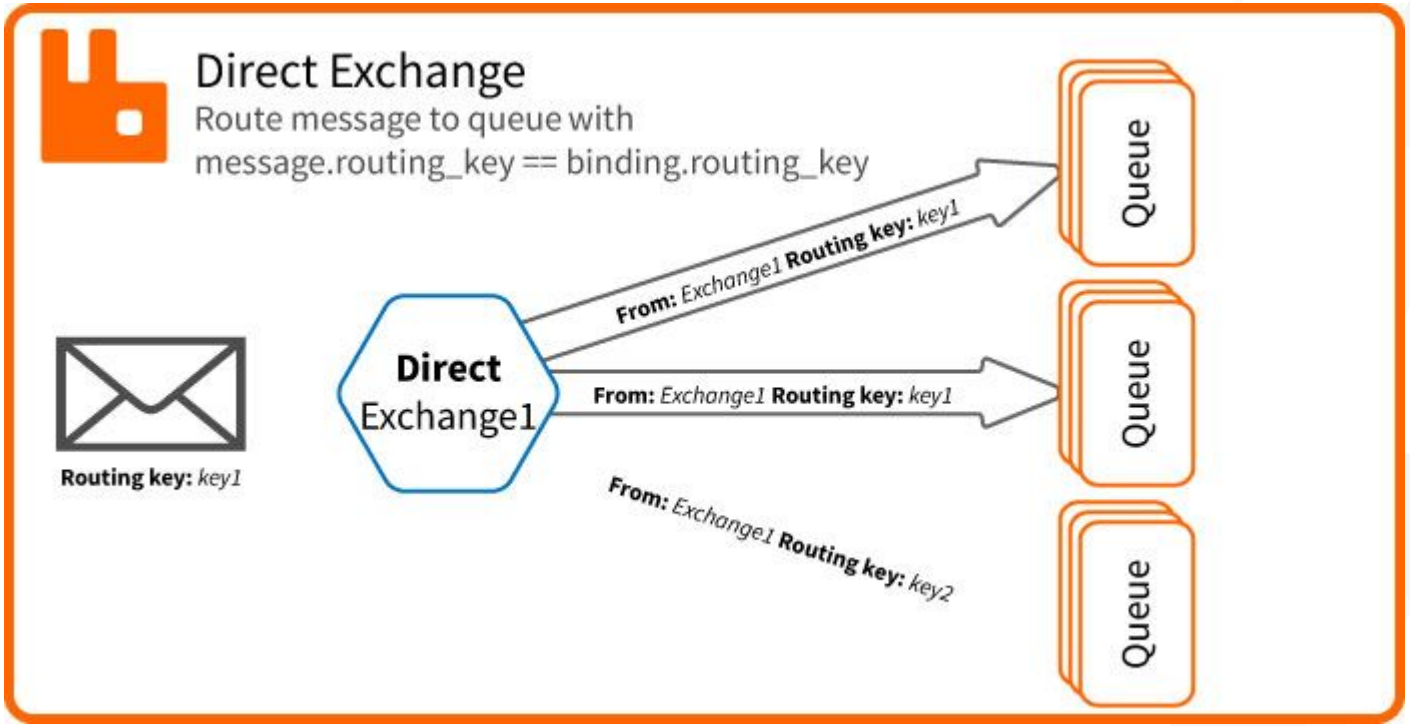
Type FANOUT:

L'exchange fanout est le plus simple. Il délivre le message à toutes les queues bindées.



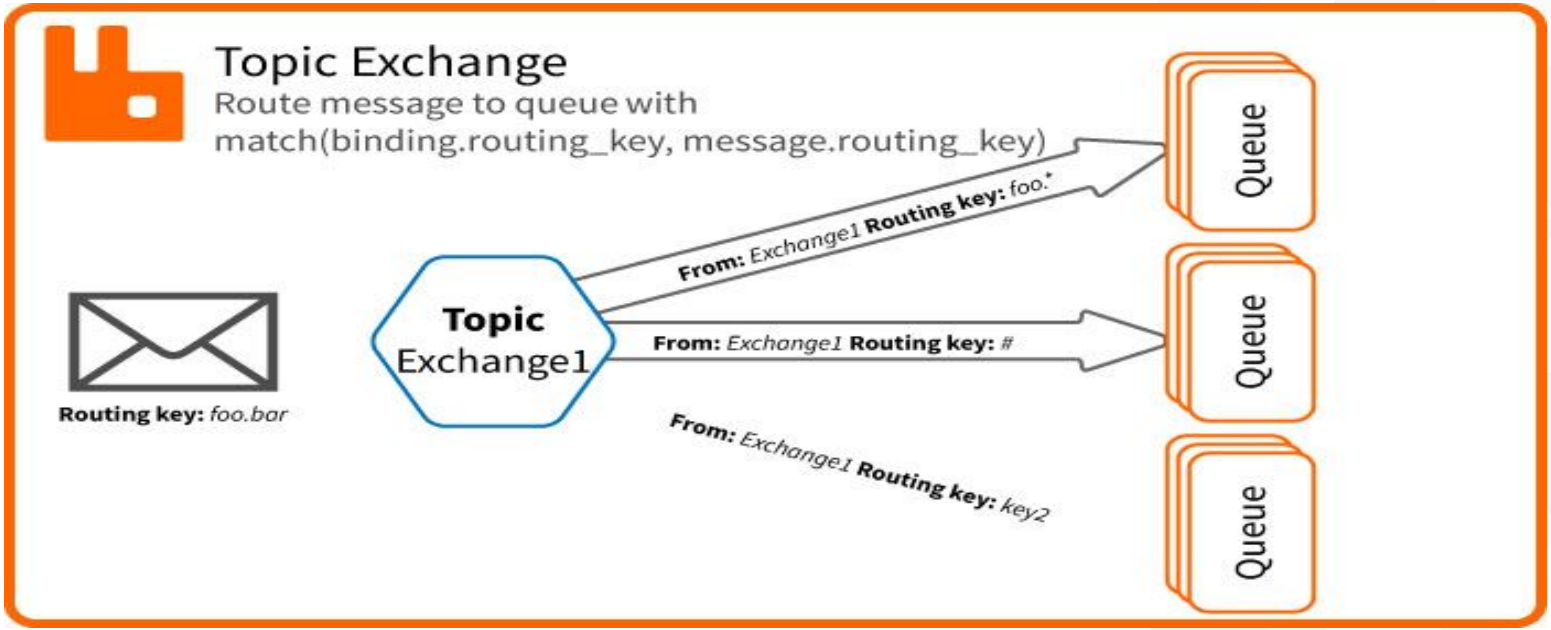
Type DIRECT:

L'exchange direct n'autorise que le binding utilisant strictement la routing key.



Type TOPIC:

L'exchange topic délivre le message si routing_key du message matche le pattern défini dans le binding.



Une routing key est composée de plusieurs segments séparés par des **points** (.). Il y a également 2 caractères utilisés dans le matching.

***** n'importe quelle valeur de segment

n'importe quelle valeur de segment une ou plusieurs fois

Par exemple pour la routing key **foo.bar.baz**

foo.*.baz **match**

foo.# **match**

..baz **match**

#.bar.baz **match**

foo.* **non trouvé**

foo.*.* **match**

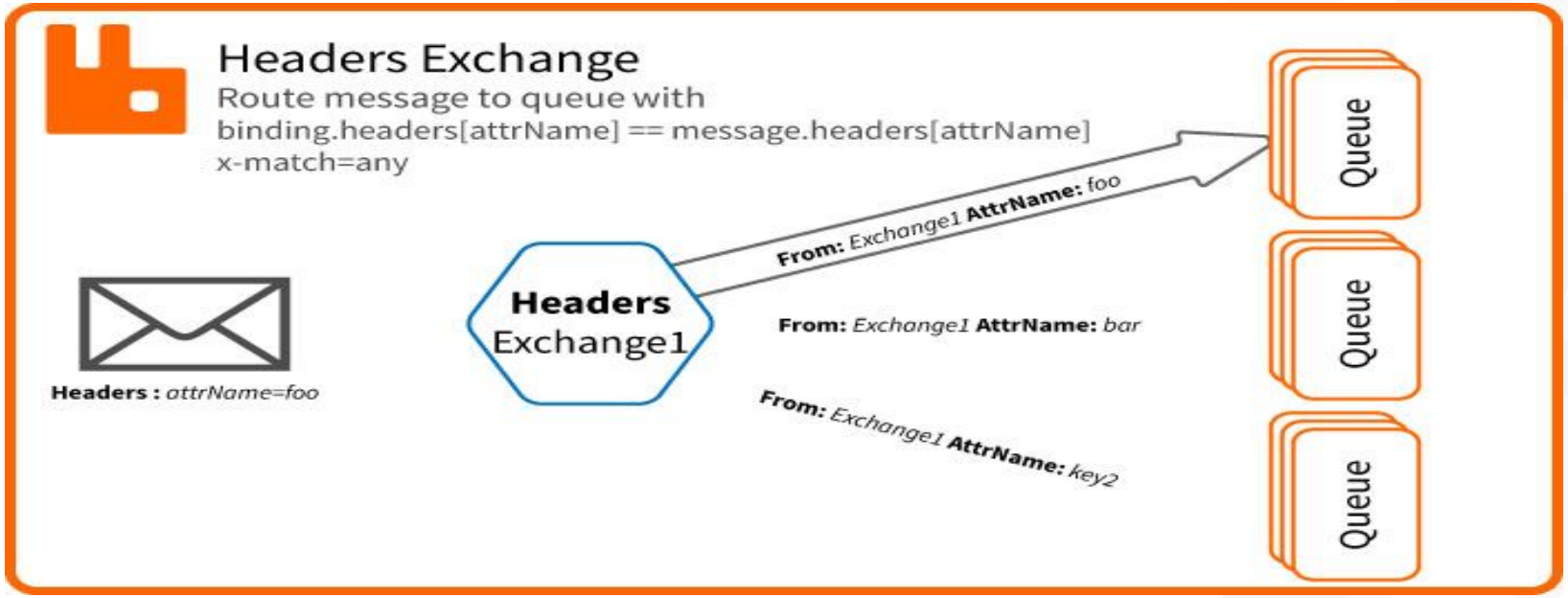
foo.#.baz **match**

#.baz **match**

match

Type HEADERS:

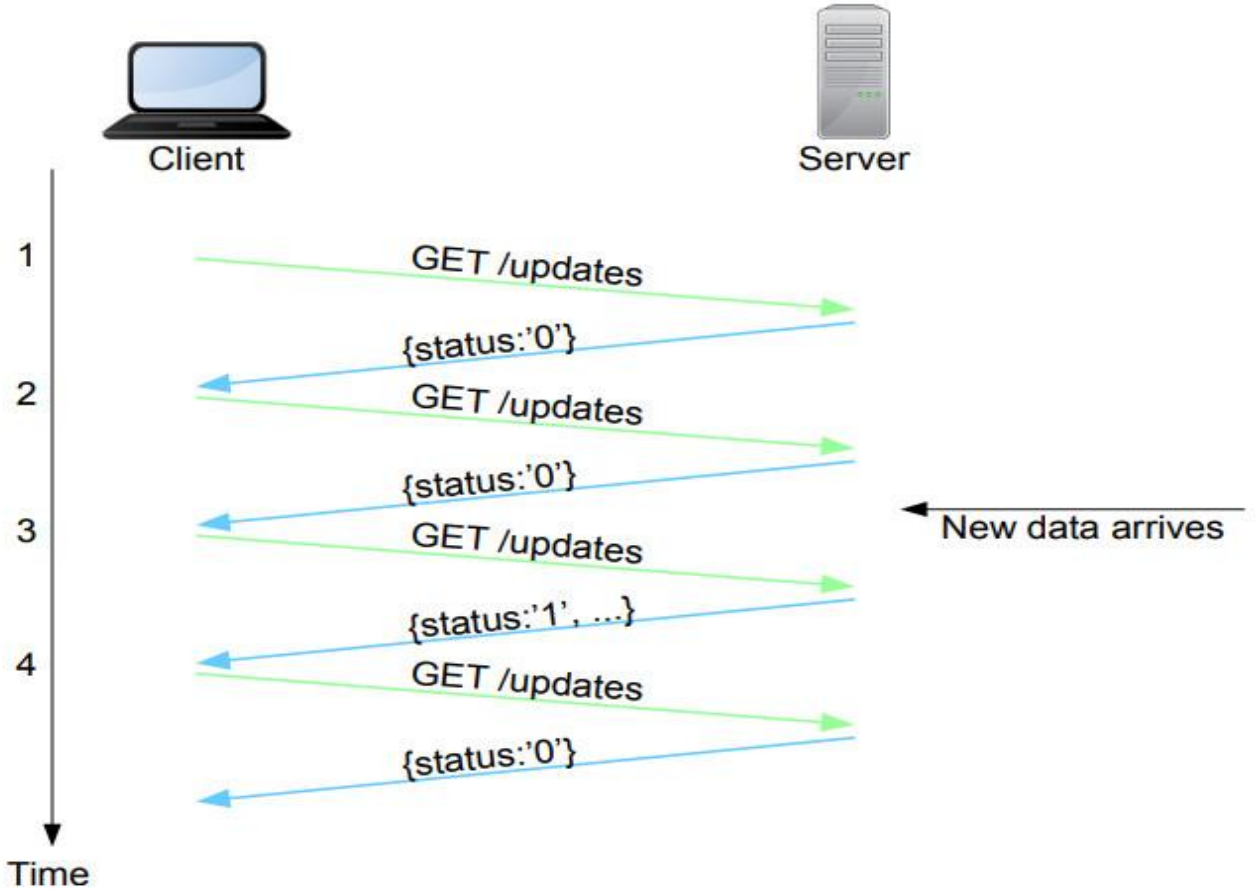
L'exchange headers délivre le message si les headers du binding matchent les headers du message.



Présentation de la solution | WebSocket

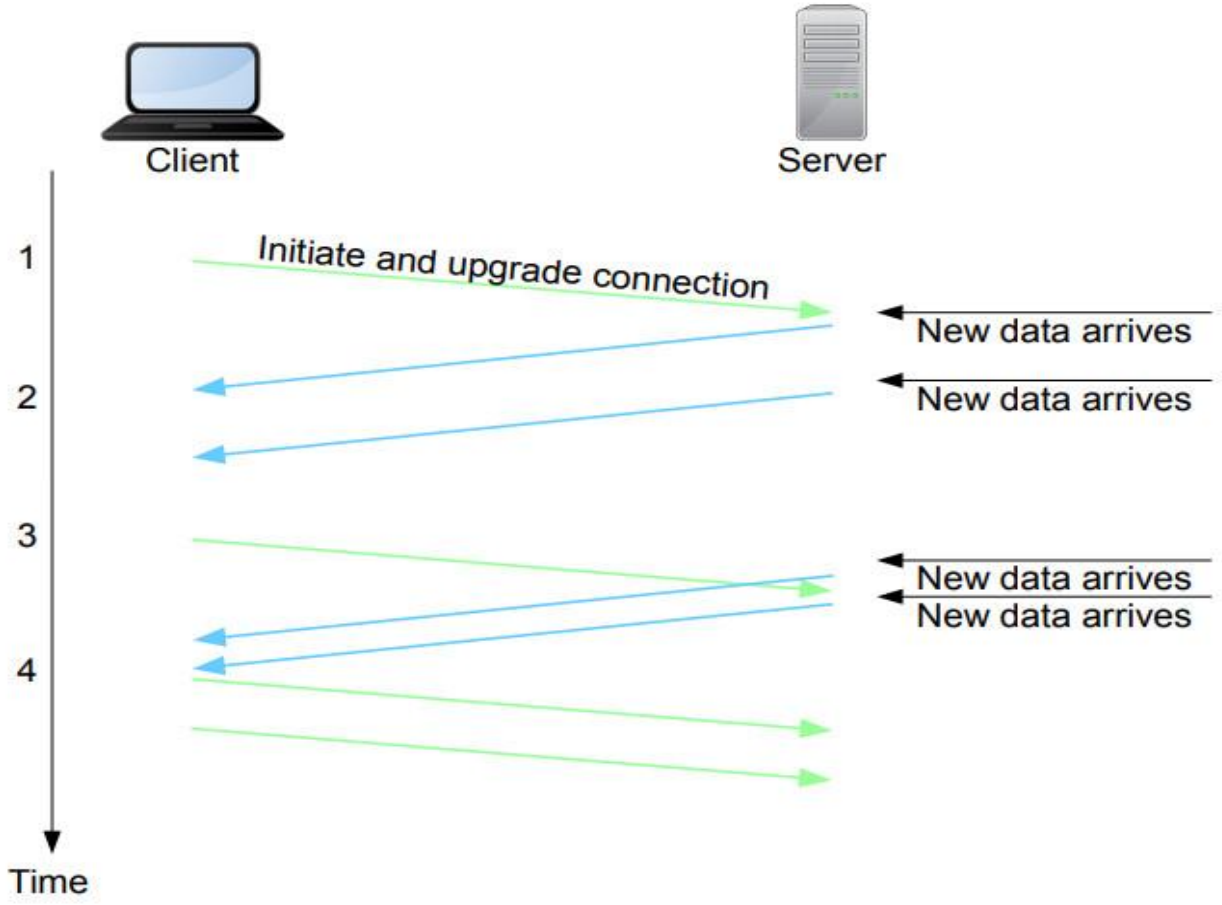
Présentation de la solution | WebSocket

De la méthode classique



Présentation de la solution | WebSocket

Vers WebSocket



Présentation de la solution | WebSocket

Le **protocole WebSocket** vise à développer un canal de communication **full-duplex** (bidirectionnel) sur un socket TCP pour les navigateurs et les serveurs web.

Il est utilisé pour :

- Synchronisation des données entre utilisateurs
- Mises à jour en temps réel sur les processus en cours d'exécution
- Applications financières
- Messagerie / chat
- etc ...

Présentation de la solution | WebSocket

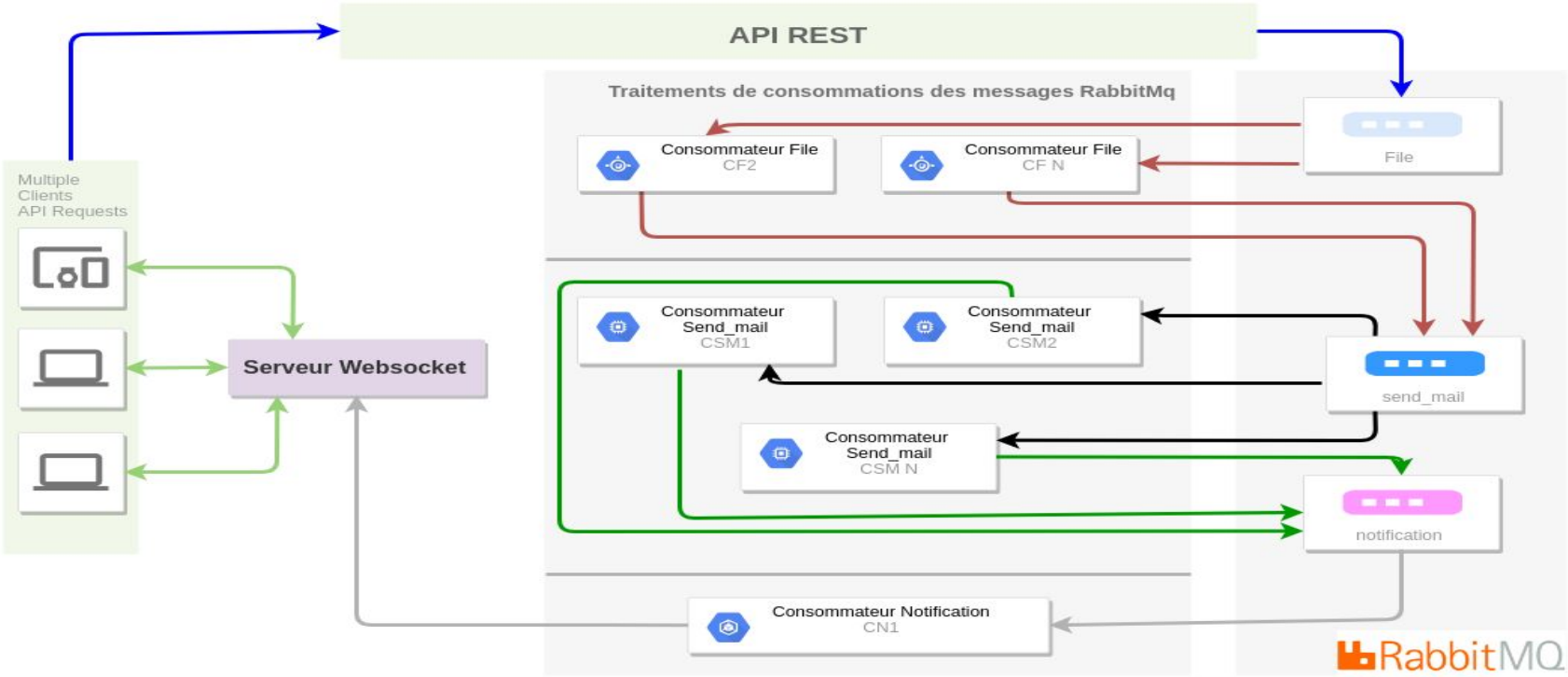
Le **protocole WebSocket** vise à développer un canal de communication **full-duplex** (bidirectionnel) sur un socket TCP pour les navigateurs et les serveurs web.

Il est utilisé pour :

- Synchronisation des données entre utilisateurs
- Mises à jour en temps réel sur les processus en cours d'exécution
- Applications financières
- Messagerie / chat
- etc ...

PRÉSENTATION DE LA SOLUTION | ARCHITECTURE SYSTÈME

Architecture d'envoi des emails en masse en multithreading



- Ouverture des flux entre les navigateurs et le serveurs websocket
- Lancer le traitement et placer les messages dans la file d'attente "file"
- Consommer les messages depuis la files d'attente "file" et parcourir les fichier et remplir la file d'attente "send_mail".
- Consommer les messages depuis la file d'attente "send_mail" et envoyer les mails personnalisés
- Pour chaque lot de messages, alimenter la file d'attente "notification" par la progression
- Consommer les messages depuis la file d'attente "notification" et les renvoyer vers le serveur websocket

SOMMAIRE

Références

http://www.rabbitmq.com/documentation.html

https://blog.eleven-labs.com/fr/rabbitmq-partie-1-les-bases/

https://fr.wikipedia.org/wiki/WebSocket#Le_protocole_WebSocket

https://fr.slideshare.net/wimg/building-interactivity-with-websockets