Julien SOMASUNDARAM
Sashtiga SASIRAJAH
Robin MALPART
Dev Full Stack - Groupe 3

Rapport de conception du système : Projet Messagerie

Implémentation d'un Système de Messagerie avec RabbitMQ.

Contexte

Pour ce projet, il est attendu des équipes qu'elles livrent un système de messagerie basé sur RabbitMQ. Le langage de programmation est au choix des équipes et l'application devra se lancer depuis un conteneur Docker.

La forme du système de messagerie est libre mais il était demandé de respecter les contrainte suivante :

- Implémentation des Producteurs et des Consommateurs
- Gestion des files d'attentes
- Routage et échanges
- Gestions des erreurs et de la fiabilité
- Sécurité
- Monitoring et Maintenance

Ce rapport présentera d'abord le projet grâce à une description globale, il décrira ensuite les choix fait à la conception du projet pour répondre aux contraintes imposées et il présentera enfin une fonctionnalité supplémentaire que nous avons implémentée pour aller plus loin.

Description générale du projet effectué

Pour implémenter notre système de messagerie, nous avons défini deux principaux acteurs : le Producteur et le Consommateur. Le Producteur est responsable de l'envoi des messages, tandis que le Consommateur réceptionne les messages envoyés par un ou plusieurs Producteurs.

Un Producteur transmet ses messages à un ou plusieurs Consommateurs via un exchange, qui distribue les messages dans une ou plusieurs queues en fonction des règles de routage.

Ainsi, un Consommateur abonné à une queue peut visualiser et traiter les messages reçus (via RabbitMQ).

En ce qui concerne les exchanges, nous avons opté pour le type topic. Nous avons écarté l'exchange amq.default car il envoie les messages à toutes les queues, et l'exchange Fanout car il ne permet pas de gérer des conversations privées. Les exchanges Direct et Headers nécessiteraient des clés de routage identiques pour l'exchange et les queues, ce qui n'était pas optimal pour notre projet. Le choix de l'exchange topic nous permet de bénéficier de la flexibilité des masques de routage (#) pour matcher avec toutes nos queues et de faciliter la gestion des logs.

Nous avons mis en place différentes queues dédiées à chaque type de conversation. Il existe une queue pour la conversation « all », destinée à tous les utilisateurs, et des queues spécifiques pour chaque conversation privée, assurant une communication exclusive entre deux utilisateurs.

Pour la gestion des erreurs, nous utilisons auto_ack=true lors de la réception des messages. Cela permet d'envoyer automatiquement un accusé de réception au serveur RabbitMQ, qui peut alors supprimer le message de la queue.

Pour assurer la sécurité de notre système de messagerie, nous avons implémenté une authentification. Seuls les utilisateurs authentifiés via RabbitMQ peuvent se connecter, envoyer et recevoir des messages.

Pour le suivi et la surveillance, nous avons intégré un système de logs. Cela nous permet de visualiser et d'analyser tous les échanges entre les utilisateurs, garantissant ainsi une traçabilité complète.

Le choix du python nous semblait naturel, car c’est le langage avec lequel nous partageons le plus de connaissances..

Description des choix réalisé pour les features demandées

Gestion des files d’attentes

Choix faits :

Déclaration de file d'attente anonyme :

```
result = channel.queue_declare(queue="")
queue_name = result.method.queue
```

Avantages :

- Simplicité : La déclaration d'une file d'attente anonyme est simple et ne nécessite pas de nom spécifique.
- Flexibilité : Les files d'attente anonymes sont temporaires et sont supprimées automatiquement lorsque le consommateur se déconnecte, ce qui est utile pour des sessions de messagerie temporaires.

Inconvénients :

- Non persistance : Les files d'attente anonymes ne sont pas durables, donc les messages sont perdus si RabbitMQ redémarre ou si le l’utilisateur se déconnecte.
- Difficile à suivre : L'usage de files d'attente anonymes peut rendre le suivi et le débogage plus difficiles car il n'y a pas de nom de file d'attente constant.

Routage et Échanges

Choix faits :

Utilisation de l'échange de type topic

```
channel.exchange_declare(exchange="direct", exchange_type="topic")
channel.queue_bind(exchange="direct", queue=queue_name, routing_key=discussion)
```

Avantages :

- Flexibilité de routage : Les échanges de type topic permettent un routage flexible basé sur des clés de routage qui peuvent contenir des jokers (* et #), ce qui est utile pour les discussions entre utilisateurs.
- Scalabilité : Permet d'ajouter facilement de nouvelles clés de routage pour de nouvelles discussions sans reconfigurer l'échange.

Inconvénients :

- Complexité : La gestion des clés de routage peut devenir complexe si les règles de routage deviennent nombreuses ou sophistiquées, dans le mesure où nous voudrions étendre notre projet.

Gestion des Erreurs et de la Fiabilité

Choix faits :

Utilisation de la persistance des messages :

```
channel.basic_publish(exchange="direct", routing_key=self.current_discussion, body=message,
 properties=pika.BasicProperties(delivery_mode=pika.DeliveryMode.Persistent))
```

Avantages :

- Durabilité : Les messages persistants sont sauvegardés sur le disque et ne sont pas perdus en cas de redémarrage de RabbitMQ.
- Fiabilité accrue : Améliore la fiabilité du système en garantissant que les messages ne sont pas perdus.

Inconvénients :

- Performance : La persistance des messages peut affecter les performances car l'écriture sur le disque est plus lente que la mémoire.
- Complexité : Ajoute une couche supplémentaire de complexité en termes de gestion des ressources et de configuration.

Sécurité

Choix faits :

Utilisation des informations d'identification pour la connexion :

```
credentials = pika.PlainCredentials(utilisateur, mot_de_passe)
connection_params = pika.ConnectionParameters("localhost", credentials=credentials
```

Avantages :

- Contrôle d'accès : Utiliser des informations d'identification permet de restreindre l'accès au système de messagerie.
- Traçabilité : Permet de suivre quel utilisateur a envoyé ou reçu quels messages.

Inconvénients :

- Gestion des informations d'identification : Nécessite une gestion sécurisée des informations d'identification pour éviter les fuites.

Monitoring et Maintenance

Choix faits :

Utilisation de l'API HTTP de RabbitMQ pour récupérer les utilisateurs et les connexions :

```
api_url = "http://localhost:15672/api/users/"
response = requests.get(api_url, auth=(self.input_utilisateur.get(), self.input_mot_de_passe.get()))
```

Avantages :

- Visibilité : Utiliser l'API de RabbitMQ pour le monitoring permet d'obtenir des informations détaillées sur les utilisateurs et les connexions.
- Facilité d'intégration : L'API HTTP est facile à intégrer avec les outils de gestion et de monitoring existants.

Inconvénients :

- Performance : Faire des appels API peut ajouter une charge supplémentaire sur le serveur RabbitMQ.
- Sécurité : L'utilisation de l'API HTTP nécessite des informations d'identification, et ces appels doivent être sécurisés pour éviter des fuites de données

Fonctionnalité supplémentaire

Pour aller un peu plus loin dans notre projet et pour expérimenter un avec les outils de rabbitMQ, nous avons également implémenté un système de messagerie privé, ou un utilisateur à la possibilité d'envoyer un message à un autre sans qu'il ne puisse être lu par d'autres.

De la même manière que nous avons décrit les fonctionnalité technique précédente, voici une présentation de la manière dont nous avons implémenté cette feature, et les avantages et inconvénient de s'y prendre de cette manière :

Système de Messagerie Privée entre Utilisateurs

Choix faits :

Utilisation de clés de routage pour les discussions privées

```
def getDiscussion(self, user):
 la_users = sorted([user, self.input_utilisateur.get()], key=str.lower)
 new_discussion = f"{la_users[0]}_{la_users[1]}"
 if(user == "all"):
 new_discussion = "all"
 return new_discussion
```

Création de boutons pour les utilisateurs connectés :

```
self.users[user] = customtkinter.CTkButton(self.root, text=user,
 width=100, height=50,
 fg_color="green",
 command=lambda user=user: self.switch_discussion(user))
self.users[user].pack()
```

Avantages :

- Séparation des discussions : En utilisant des clés de routage basées sur les noms des utilisateurs, les discussions privées sont clairement séparées.
- Facilité d'utilisation : L'interface graphique permet de basculer facilement entre les discussions privées et générales.

Inconvénients :

- Complexité de gestion des clés de routage : La gestion des discussions privées nécessite une bonne gestion des clés de routage pour éviter les collisions et assurer que les messages sont correctement routés.
- Sécurité et confidentialité : Les messages ne sont pas chiffrés, ce qui pourrait poser des problèmes de confidentialité.