Brahim BEN EL HADJ
Chuaibe TAMIMOUL ANESSARY

Rapport de Conception du Système : Projet Messagerie

Implémentation d’un Système de Messagerie avec RabbitMQ

Contexte

Ce projet vise à développer un système de messagerie basé sur RabbitMQ. Les équipes ont le choix du langage de programmation, et l’application doit être déployée à partir d'un conteneur Docker. Les contraintes à respecter incluent :

- Implémentation des Producteurs et des Consommateurs
- Gestion des files d’attentes
- Routage et échanges
- Gestion des erreurs et de la fiabilité
- Sécurité
- Monitoring et maintenance

Ce rapport décrit le projet de manière globale, les choix de conception effectués pour répondre aux contraintes, ainsi qu'une fonctionnalité supplémentaire implémentée.

Description Générale du Projet

Pour notre système de messagerie, nous avons défini deux acteurs principaux : le Producteur et le Consommateur. Le Producteur envoie des messages, et le Consommateur les reçoit. Les messages passent par un échange (exchange) qui les distribue dans des files d'attente (queues) selon des règles de routage. Nous avons choisi un exchange de type topic pour sa flexibilité et sa capacité à gérer les conversations privées.

Nous avons mis en place diverses queues pour chaque type de conversation. Une queue "all" est destinée à tous les utilisateurs, tandis que des queues spécifiques assurent des communications privées entre deux utilisateurs. Pour la gestion des erreurs, nous utilisons auto_ack=true pour accuser réception automatiquement et permettre à RabbitMQ de supprimer le message de la queue. En termes de sécurité, seuls les utilisateurs authentifiés peuvent se connecter, envoyer et recevoir des messages. Un système de logs est intégré pour le suivi et l'analyse des échanges.

Description des Choix Réalisés pour les Fonctionnalités Demandées

Gestion des Files d'Attentes

- **Choix :** Déclaration de file d'attente anonyme.

python
Copier le code
```
result = channel.queue_declare(queue="")
queue_name = result.method.queue
```

- **Avantages :**
  - Simplicité : Déclaration facile sans besoin de nom spécifique.
  - Flexibilité : Les queues anonymes sont temporaires et supprimées automatiquement lorsque le consommateur se déconnecte.
- **Inconvénients :**
  - Non persistance : Les messages sont perdus si RabbitMQ redémarre.
  - Difficile à suivre : Pas de nom de queue constant, ce qui complique le suivi et le débogage.

Routage et Échanges

- **Choix :** Utilisation de l'échange de type topic.

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

- **Avantages :**
  - Flexibilité de routage avec des clés de routage contenant des jokers (* et #).
  - Scalabilité : Ajout facile de nouvelles clés de routage.
- **Inconvénients :**
  - Complexité accrue avec l’augmentation des règles de routage.

Gestion des Erreurs et de la Fiabilité

- **Choix :** Utilisation de la persistance des messages.

python
Copier le code
```
channel.basic_publish(exchange="direct",
routing_key=self.current_discussion, body=message,
properties=pika.BasicProperties(delivery_mode=pika.DeliveryMode.Pers
istent))
```

- **Avantages :**
  - Durabilité : Les messages sont sauvegardés sur le disque et ne sont pas perdus en cas de redémarrage.
  - Fiabilité accrue : Assure que les messages ne sont pas perdus.
- **Inconvénients :**
  - Performance : L'écriture sur le disque est plus lente que la mémoire.
  - Complexité supplémentaire pour la gestion des ressources et de la configuration.

Sécurité

- **Choix :** Utilisation des informations d'identification pour la connexion.

python
Copier le code
```
credentials = pika.PlainCredentials(utilisateur, mot_de_passe)
connection_params = pika.ConnectionParameters("localhost",
credentials=credentials)
```

- **Avantages :**
  - Contrôle d'accès : Restreint l'accès au système de messagerie.
  - Traçabilité : Suivi des utilisateurs ayant envoyé ou reçu des messages.
- **Inconvénients :**
  - Gestion sécurisée nécessaire des informations d'identification.

Monitoring et Maintenance

- **Choix :** Utilisation de l'API HTTP de RabbitMQ pour récupérer les utilisateurs et les connexions.

python
Copier le code

```
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 : Informations détaillées sur les utilisateurs et les connexions.
  - Facilité d'intégration avec des outils de gestion existants.
- **Inconvénients :**
  - Performance : Charge supplémentaire sur le serveur RabbitMQ.
  - Sécurité : Nécessite des informations d'identification sécurisées.

Fonctionnalité Supplémentaire : Messagerie Privée

Pour améliorer notre projet, nous avons implémenté un système de messagerie privée permettant à un utilisateur d'envoyer un message à un autre sans que ce message puisse être lu par d'autres.

Système de Messagerie Privée entre Utilisateurs

- **Choix :** Utilisation de clés de routage pour les discussions privées.

python
Copier le code

```
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 :**

python
Copier le code

```
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 claire des discussions privées.
    - Interface utilisateur facilitant le basculement entre discussions privées et générales.
- **Inconvénients :**
    - Gestion complexe des clés de routage pour éviter les collisions.
    - Absence de chiffrement des messages, posant des problèmes de confidentialité.