TP noté : Application de chat

Equipe :
NOHET Floriane
SAYEZ Jacques
DEFER Tanguy

Lien du projet :
https://github.com/tanguyDefer/tp_kafka

3 Implémentation

Exercice 1

Nous avons pris connaissance et analysé le fichier chat_client,py,

Exercice 2

Décrire la méthode subscribe de KafkaConsumer:
Elle permet de s'abonner à un ou plusieurs topics qui seront consommés, elle prend en paramètre une liste de topics.

Décrire la méthode subscription de KafkaConsumer:
Permet d'obtenir l'abonnement actuel. Renverra les mêmes topics utilisés dans l'appel le plus récent à subscribe(), ou un ensemble vide si aucun appel de ce type n'a été effectué.

3. Implémentez la commande join. + Exercice 2,1

La méthode **cmd_join** est appelée dans **main_loop()** :

```
elif cmd == "join":
    if check_channel_format(args):
        if cmd_join(nick_name, consumer, producer, args):
            curchan = args
```

Ici, si la commande (donnée récupérée à partir de ce que l'utilisateur tape dans la console) est **join**, premièrement nous faisons une vérification du format sur **args** (*String → nom du canal à rejoindre*) avec la méthode **check_channel_format()** , si la vérification est **OK** alors on passe à la méthode **cmd_join()**, si **cmd_join()** est **OK** alors le canal courant (*canal actif sur lequel l'utilisateur peut envoyer et recevoir des messages*) est mis à jour avec le **nom de canal.**

```python
def check_channel_format(args):      You, il y a 3 jours • Add color lvl for logger
    """check if args format is correct
    Args:
        args (_type_): #general

    Returns:
        True or False
    """
    if re.match(r'^#[a-zA-Z0-9_-]+$', args):
        return True
    log.warning("Incorrect channel format for '%s', channel format must be like '#general'", args)
    return False
```

La méthode **check_channel_format** vérifie le format du nom de canal grâce à la **regex '^#[a-zA-Z0-9_-]+$'** car il doit commencer par un dièse et ne contenir que des lettres, chiffres ou tirets), Si **OK** on retourne **True**, sinon un message de **log warning** informe de l'erreur et retourne **False** afin de pouvoir refaire la commande.

```
WARNING | Incorrect channel format for 't', channel format must be like '#general'
```

Méthode cmd_join() :

```python
def cmd_join(nick_name,consumer, producer, args):
    try:
        consumer.subscribe(check_channel_format(args))
        if args not in SUB_CHANNELS:
            SUB_CHANNELS.append(args)
        message_to_channel = "{} has joined chat channel : {} ".format(nick_name, args[1:])
        log.info(message_to_channel)
        info_message_to_channel(producer, args, message_to_channel)
        log.info("List of %s's channels : %s",nick_name, SUB_CHANNELS)
        return True
    except Exception as err:
        log.error("Subscribe to : %s failed", err)
        return False
```

Elle prend en paramètre **nick_name** *(string pseudo de l'utilisateur*), **consumer**(*objet KafkaConsumer()*), **producer** (*objet KafkaProducer()*), **args**(*string nom du canal*),

Le but de cette méthode est de rejoindre un canal avec la commande **/join #nomDuCanal** et de le définir comme **canal actif**,

En premier on fait un **consumer,subscribe** avec le bon formatage grâce à la méthode **format_channel_name()** :

```python
def format_channel_name(args):
    """function to transform #general to chat_channel_general"""
    return "chat_channel_" + args[1:]
```

SUB_CHANNELS est une constante qui permet de stocker les canaux que l'utilisateur à rejoins,
&
Si le canal entré en paramètre n'est pas dans **SUB_CHANNELS** alors on l'ajoute avec la fonction ,**append()**, ensuite on construit un message qui annoncera que l'utilisateur a rejoint le canal par un message ce dernier.

```
[##general]> < chat_channel_general: b'Tanguy has joined chat channel : general '
```

La méthode **info_message_to_channel()** est appelée :

```python
def info_message_to_channel(producer, args, message_to_channel):
    formated_channel = format_channel_name(args)
    producer.send(formated_channel, str.encode(message_to_channel))
```

Dans un premier temps on formate le nom du canal, ensuite **producer,send** permet *d'envoyer des messages asynchrones sur un topics*,

Pour finir un **log,info** nous renseigne les canaux auxquels l'utilisateur est abonnés,
Quand l'utilisateur rejoint un canal celui-ci est *activé* et l'utilisateur est directement dessus, il peut le voir avec [##general]>
Voici ce qu'affiche le terminal :

```
> /join #general
[2022-06-15 18:19:08] INFO | Tanguy has joined chat channel : general
[2022-06-15 18:19:08] INFO | List of Tanguy's channels : ['#general']
[##general]>
```

4. Implémentez la commande part + exercice 2,2

La méthode **cmd_part** est appelée dans **main_loop()** :

```python
elif cmd == "part":
    if cmd_part(nick_name,consumer, producer, args) or None:
        if SUB_CHANNELS == []:
            curchan = None
        else:
            curchan = SUB_CHANNELS[-1]
```

Ici, si la commande (donnée récupérée à partir de ce que l'utilisateur tape dans la console) est **part**, si la méthode **cmd_part()** est **OK** ou **None** → si **SUB_CHANNELS** est vide cela veut dire que l'utilisateur n'est abonné à aucun autre canal et donc il n'y a pas de canal **actif**, sinon on prend le dernier canal dans la liste et on le passe comme **actif**,

Méthode cmd_part() :

```python
def cmd_part(nick_name, consumer, producer, args):
    """function to quit a channel and join one of user's channels

    Args:
        consumer (kaka consumer)
        producer (kaka producer)
        args (str): channel to quit
    Returns:
        _type_: subscribe channel or False
    """
    if args in SUB_CHANNELS:
        if len(SUB_CHANNELS) == 0:
            log.warning("No channel subscribe")
        else:
            if len(SUB_CHANNELS) == 1:
                log.warning("You will not subscribe to any channels ... redirect to channels list")
                message_to_channel = "{} has left chat channel : {} ".format(nick_name, args[1:])
                log.info(message_to_channel)
                info_message_to_channel(producer, args, message_to_channel)
                consumer.unsubscribe()
                SUB_CHANNELS.remove(args)
                return True
            consumer.unsubscribe()
            log.info("%s has left chat channel : %s",nick_name, args[1:])
            message_to_channel = "{} has left chat channel : {} ".format(nick_name, args[1:])
            info_message_to_channel(producer, args, message_to_channel)
            consumer.unsubscribe()
            SUB_CHANNELS.remove(args)
            return True      You, il y a 4 heures • Update /part cmd …
    else:
        log.warning("%s is not in your channels", args)
        return False
```

Elle prend en paramètre **nick_name** *(string pseudo de l'utilisateur*), **consumer**(*objet KafkaConsumer()*), **producer** (*objet KafkaProducer()*), **args**(*string nom du canal*),

Le but de cette méthode est de quitter un canal avec la commande **/part #nomDuCanal** et d'envoyer un message ''USER has left chat chanel nomDuCanal'' sur ce canal,

En premier vérifie que le canal entré (**args**) se trouve dans la liste des canaux souscrits (**SUB_CHANNELS**), sinon **log.warning("No channel subscribe")** sera affiché dans le terminal.

```
[##general]> /part #noChan
[2022-06-15 19:41:37] WARNING | #noChan is not in your channels
```

Ensuite, s'il y a un seul canal souscrit dans **SUB_CHANNELS** alors un log,warning prévient qu'après avoir quittée le canal l'utilisateur ne sera abonné à aucun canal

```
[##general]> /part #general
[2022-06-15 19:43:25] WARNING | You will not subscribe to any channels ... redirect to channels list
```

un message (qui sera envoyé sur le canal quitté) est construit et sera envoyé avec la méthode **info_message_to_channel()** décrite plus haut,

```
[##general]> < chat_channel_general: b'Floriane has left chat channel : general '
```

Aussi un **log,info** affiche le même message

```
[2022-06-16 10:37:29] INFO | Floriane has left chat channel : general
```

consumer.unsubscribe() permet de retirer le **topic** du **consumer.**
Enfin, **SUB_CHANNELS.remove(args)** permet de retirer le canal de la liste des canaux souscrits de l'utilisateur.

Si l'utilisateur à plus d'un canal souscrit, la procédure est la même mais sans le log,warning qui prévient qu'après avoir quitté le canal l'utilisateur ne sera abonné à aucun canal.

Pour finir, si nous sommes dans aucun des cas cités précédemment, cela signifie que le canal entré ne se trouve pas dans la liste des canaux actifs, un **log,warning** affichera ceci :

```
/part #noChan
[2022-06-15 19:58:34] WARNING | #noChan is not in your channels
[##general]>
```

5. Implémentez la commande msg:

La méthode **cmd_msg** est appelée dans **main_loop**() :

```
if cmd == "msg":
    cmd_msg(producer, curchan, args, nick_name)
```

Ici, si la commande (donnée récupérée à partir de ce que l'utilisateur tape dans la console) est **msg**.

Méthode **cmd_msg() :**

```
def cmd_msg(producer, curchan, message, nick_name):
    if curchan:
        log.info("Sending message to %s ...", curchan)
        formated_curchan = format_channel_name(curchan)
        formated_message = str.encode(nick_name + ": " + message)
        try:
            # Kafka attend un message format bytes, il faut donc convertir avec str.encode()
            producer.send(formated_curchan, formated_message)
            producer.send(BOT_CHANNEL, key=str.encode(nick_name), value=formated_message)
            producer.flush()
            log.info("Message sent by %s on channel %s", nick_name, curchan)
        except Exception as err:
            log.warning("Impossible to send message ... %s", err)
    else:
        log.warning("No active channel")
```

Elle prend en paramètre **nick_name** *(string pseudo de l'utilisateur*), **producer** (*objet KafkaProducer()*), **curchan**(*string nom du canal actif*), **message** (string message à

envoyer), elle permet à l'utilisateur d'envoyer des messages sur le canal actif, elle envoie aussi une copie de chaque messages sur un topic destiné à la modération.

Premièrement, on vérifie si un canal est actif, sinon un **log,warning** affichera ceci :

```
> /msg hello
[2022-06-15 20:09:52] WARNING | No active channel
>
```

Ensuite, un **log,info** informe que l'envoie du message est en cours :

```
[##general]> hello
[2022-06-15 20:12:24] INFO | Sending message to #general ...
```

Le nom du canal est formaté grâce à la méthode **format_channel_name()** vue dans l'exercice 2. Le message est formaté au bon format et on ajoute le pseudo de l'utilisateur.

Le message est envoyé sur le canal actif et sur le canal modération **BOT_CHANNEL**

un **log,info** informe que le message est bien envoyé avec le canal et l'utilisateur comme informations :

```
[##general]> /msg Hello
[2022-06-16 10:39:14] INFO | Sending message to #general ...
[2022-06-16 10:39:14] INFO | Message sent by Jacques on channel #general
[##general]> < chat_channel_general: b'Jacques: Hello'
```

Si le message ne s'est pas envoyé alors l'utilisateur est informé par un **log,warning**

6. Rajoutez une commande active.

La méthode active est appelée dans main_loop() :

```
elif cmd == "active":
    if is_active(args):
        curchan = args
```

Elle permet d'appeler la méthode **is_active()**, si elle retourne **True** alors le canal entré en paramètre est défini comme actif,

Méthode is_active():

```python
def is_active(args):      You, il y a 6 heures • add bot_channel.py
    """Permet d'activer un canal et recevoir les messages sur ce canal

    Args:
        args (str): channel name
    """
    if args in SUB_CHANNELS:
        log.info("Channel : {} is now active".format(args))
        return True
    else:
        log.warning("Channel {} is not in your channel list : {}".format(args, SUB_CHANNELS))
        return False
```

Elle prend en paramètre **args**(string nom du canal).
Si le canal entré se trouve dans la liste des canaux souscrits alors on retourne **True** et ce canal sera actif, un **log,info** informe de ceci :

```
[##general]> /active #general
[2022-06-15 20:52:55] INFO | Channel : #general is now active
[##general]>
```

Sinon un **log,warning** informe de l'erreur et on retourne **false:**

```
[##general]> /active #NoChan
[2022-06-15 20:54:06] WARNING | Channel #NoChan is not in your channel list : ['#general']
[##general]>
```

Exercice 2 Messages de connexion

1, Envoyez un message "nick has joined" sur le canal qui vient d'être rejoint
cf : voir méthode **cmd_join()**

2, Similairement, envoyez un message "nick has left" sur le canal qui vient d'être quitté.
Cf : voir méthode **cmd_part()**

3. Lorsqu'un utilisateur se déconnecte envoyez un message de départ sur tous les canaux qui étaient joint

La méthode **cmd_quit** est appelée dans **main_loop()** :

```python
elif cmd == "quit":
    cmd_quit(producer, nick_name)
    break
```

```python
def cmd_quit(producer, nick_name):      You, il y a 19 heures • add bot_cha
    message_to_channel = "{} has disconnected".format(nick_name)
    for channel in SUB_CHANNELS:
        info_message_to_channel(producer, channel, message_to_channel)
        log.info("Disconnected")
```

Elle prent en paramètres **producer** et **nick_name**
Premièrement on construit le message qui sera envoyé à tout les canaux souscrit par l'utilisateur,

Ensuite une boucle for permet de récupérer les canaux un par un et d'appeler la méthode **info_message_to_channel()** décrite avant,

Voici la vue de l'utilisateur :

```
[##general]> /quit
[2022-06-16 09:53:21] INFO | Disconnected
```

Voici la vue des autres utilisateurs :

```
< chat_channel_general: b'Tanguy has disconnected'
```

Exercice 3 Modération : flood

1. Modifiez votre client de chat pour qu'il envoie tous les messages sur un topic dédié
en plus du topic dédié au canal. Votre bot consommera les messages de ce topic

cf : voir méthode **cmd_msg(),**

Création d'un fichier **bot_channel,py** qui aura pour but de souscrire au canal **''chat_chanel_bot"** et de répertorier les utilisateurs qui flood, dans l'état actuel il n'est pas possible (ou du moins trop compliqué à notre niveau de découverte) de traiter les données avec **SparkSession,** donc notre programme souscrit bien au canal **chat_channel_bot** et retranscrit les message dans un canal **chat_channel_ban,** il n'y a pas d'opération de count,

Ce programme est à exécuter avec la commande spark-submit → **bin/spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.1 /folder/to/file/location/bot_channel.py**

```python
def main():

  spark = SparkSession \
      .builder \
      .appName("chat_client") \
      .getOrCreate()


  df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "chat_channel_bot") \
    .load()

  words = df.select(
   explode(split(df.value, " ")).alias("value")
)

  result = words.writeStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("topic", "chat_chanel_ban").option("checkpointLocation",
  result.awaitTermination()
      You, il y a 19 heures • add bot_channel.py
if __name__ == "__main__":
    try:
        main()
    except Exception as error_exception:
        print("ERROR: %s" % error_exception)
```

Le fichier **README,MD** explique les différentes commandes du projet

Porocédure pour déployer l'application dans un environnement de dev

- Pour lancer zookeeper, se mettre dans le dossier kafka et faire la commande:

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

- Pour lancer kafka server, se mettre dans le dossier kafka et faire la commande:

```
bin/kafka-server-start.sh config/server.properties
```

- Pour ouvrir un consumer kafka, se mettre dans le dossier kafka et faire la commande:

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic chat_chanel_consumer
```

- Pour lancer notre programme, ouvrir un terminal dans le projet et faire la commande:

```
python 3 /CHEMIN/DU/FICHIER/chat_client.py USER
```

- Dans notre programme pour rejoindre un channel:

```
/join #CHANNEL_NAME
```

- Dans notre programme pour quitter un channel:

```
/part #CHANNEL_NAME
```

- Dans notre programme pour quitter le programme:

```
/quit #CHANNEL_NAME
```

- Pour lancer le producer kafka, se mettre dans le dossier kafka et faire la commande:

```
bin/kafka-console-producer.sh --topic chat_channel_general --bootstrap-server localhost:9092
```

- Pour lancer le programme bit_channel.py, ouvrir un nouveau terminal, se mettre dans le dossier spark et faire la commande:

```
bin/spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.1 /folder/to/file/location/bot_channel.py
```