



# **Rendu TP partie III :**

# **Intergiciel et programmation**

# **par composants**

Réalisé par: MEFTAH Naoufal - SEKKOUMI Samir - EL MOUDEN El Mehdi

## Configuration PostgreSQL et la base de données DBeaver

On a commencé tout d'abord par créer la base de donnée nommée "etudiants", et une table "etudiants" avec les champs suivants : identifiant, nom et prénom via ce code sql :

**CREATE TABLE public.etudiants (**

**identifiant varchar NULL,**

**nom varchar NULL,**

**prenom varchar NULL**

**);**

Ensuite, on lance le script **go.cmd (.sh)** qui nous a été déjà fourni, destiné à supprimer les fichiers et les répertoires existants de Kafka, puis démarre ZooKeeper, le courtier Kafka, un producteur (genkp) et un consommateur (kc-etudiants) dans un environnement local.

enfin, on utilise l'URL suivant : <http://localhost:7000/swagger-ui.html>, pour accéder au service Swagger.

**kp-\_-controller** KP\_ \_Controller

**GET** /insa/isalive [PROD] : Interroger le service pour connaître son état, retourner la réponse à la question sur l'univers et tout.

**POST** /insa/kppublish Send Message to Kafka Topic http://localhost:port/insa/kppublish

Parameters Cancel

Name	Description
<b>message</b> <small>required</small> (body)	message  Example Value   Model <pre>{   "identifiant": "753258",   "nom": "SEM",   "prenom": "MSN" }</pre>

Cette page de documentation nous permet de tester et d'explorer les API exposées par le programme "genKP" de manière conviviale. Elle peut remplacer l'utilisation de l'outil Postman, du moins pour cette situation simple. Cependant, Postman reste également une option intéressante pour tester les API Web Services de manière plus générale.

Curl

```
curl -X POST "http://localhost:7000/insa/kppublish" -H "accept: */*" -H "Content-Type: application/json" -d "{\"identifiant\":\"753258\", \"nom\":\"SEM\", \"prenom\":\"MSN\"}"
```

Request URL

http://localhost:7000/insa/kppublish

Server response

Code

201

Details

Response body

```
{"reponse": "Inserted"}
```

Download

Response headers

```
connection: keep-alive
content-length: 22
content-type: text/plain; charset=UTF-8
date: Thu, 08 Jun 2023 15:53:37 GMT
keep-alive: timeout=60
```

Responses

Le programme "**genKP**" reçoit les objets **JSON** via un web service, les envoie à Kafka et les publie dans le topic "ÉTUDIANTS". Le programme "kc-étudiants" consomme les messages de ce topic, extrait les données **JSON** et les intègre dans la base de données PostgreSQL. Cette architecture permet de transférer les données de manière asynchrone et de les stocker de manière persistante dans une base de données.

Navigateur de bases de données X Projets

etudiants - localhost:5432

etudiants

Propriétés Données ER Diagram


etudiants Databases etudiants Schemas public

Entrez une expression SQL pour filtrer les résultats (utilisez Ctrl+Espace)

	ABC identifiant	ABC nom	ABC prenom	
1	123456	TONDEUR	HERVE	
2	1000	MEFTAH	NAOUFAL	
3	101010	MEF	Naou	
4	78965	mehdi	samir	
5	753258	SEM	MSN	

Visionneuse de valeurs

78965



D'après la capture d'écran ci-dessus, on remarque bien que le programme 'kc-étudiants' a correctement consommé les messages du topic " ÉTUDIANTS " provenant de **KAFKA**, extrait les données **JSON** et les a insérées dans notre base de données.

On peut même vérifier notre base de données PostgreSQL pour confirmer que les données ont été insérées avec succès. En exécutant des requêtes **SQL** comme **SELECT** pour récupérer les enregistrements correspondants à partir de la table dans laquelle on a stocké les données .

## Configuration du Zookeeper :

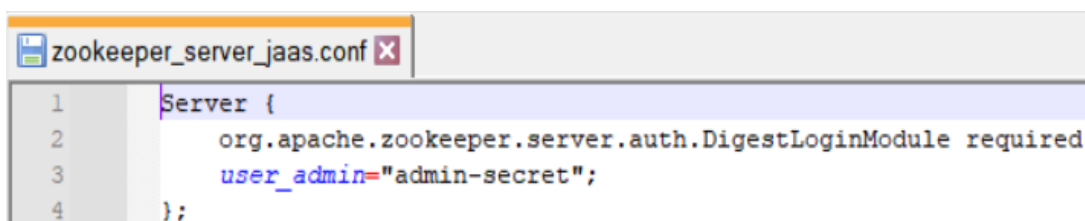
Pour sécuriser ZooKeeper avec SASL/PLAIN, nous avons effectué les étapes suivantes :

### 1. Modification du fichier `zookeeper.properties` :

```
authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider
jaasLoginRenew=3600000
jaasLoginRefresh=3600000
jaasRealm=ZooKeeper
jaasServerName=zookeeper
requireClientAuthScheme=sasl
```

- Ajout de la ligne **“authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider”** pour activer l'authentification SASL.
- Configuration des paramètres `jaasLoginRenew` et `jaasLoginRefresh` à 3600000 (1 heure) pour spécifier la durée de validité des informations d'identification.
- Définition de `jaasRealm` à ZooKeeper pour spécifier le nom du royaume JAAS utilisé pour l'authentification.
- Définition de `jaasServerName` à zookeeper pour spécifier le nom du serveur JAAS utilisé pour l'authentification.
- Spécification de `requireClientAuthScheme` à sasl pour exiger que les clients utilisent SASL pour s'authentifier.

### 2. Création du fichier JAAS (`zookeeper_server_jaas.conf`) :



```
1 Server {
2     org.apache.zookeeper.server.auth.DigestLoginModule required
3     user_admin="admin-secret";
4 }
```

Nous avons utilisé le module d'authentification `DigestLoginModule` pour définir un utilisateur admin avec le mot de passe admin-secret. Le module d'authentification `DigestLoginModule` est utilisé pour stocker les informations d'identification sous forme de hachage digest sécurisé. Dans ce cas, le mot de passe de l'utilisateur est

préalablement converti en une chaîne de hachage digest et stocké dans la configuration JAAS.

La configuration de l'utilisateur admin avec le mot de passe admin-secret dans le fichier JAAS (zookeeper\_server\_jaas.conf) permet à cet utilisateur de s'authentifier auprès de ZooKeeper en utilisant les informations d'identification fournies. Lorsqu'un client tente de se connecter à ZooKeeper, il devra fournir les mêmes informations d'identification (nom d'utilisateur et mot de passe) pour être autorisé à accéder à ZooKeeper.

### 3. Ajout de la ligne suivante dans le fichier **zookeeper-server-start.bat** :

**"set ZOOKEEPER\_OPTS=-Djava.security.auth.login.config=%~dp0../..../config/zookeeper\_server\_jaas.conf"** définit les options d'exécution (**ZOOKEEPER\_OPTS**) pour le démarrage de ZooKeeper. Lorsque ZooKeeper est démarré, il charge les propriétés spécifiées dans les options d'exécution. En définissant **java.security.auth.login.config** sur le chemin du fichier **JAAS**, nous indiquons à ZooKeeper de charger cette configuration JAAS lors de l'initialisation de l'authentification **SASL/PLAIN**.

Ces étapes assurent que ZooKeeper utilise l'authentification SASL/PLAIN avec le fichier JAAS spécifié pour valider les informations d'identification des clients qui tentent de se connecter.

## Configuration du Broker :

Pour sécuriser le broker Kafka avec SASL/PLAIN, nous avons effectué les étapes suivantes :

### 1. Modification du fichier **server.properties** :

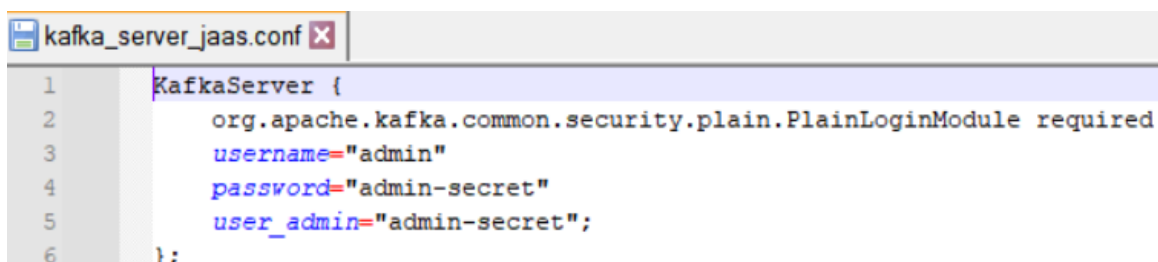
```
listeners=SASL_PLAINTEXT://localhost:9092

# Listener name, hostname and port the broker will advertise to clients.
# If not set, it uses the value for "listeners".
advertised.listeners=SASL_PLAINTEXT://localhost:9092
security.inter.broker.protocol=SASL_PLAINTEXT
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN
```

- Ajout de la ligne **"listeners=SASL\_PLAINTEXT://localhost:9092"** pour spécifier le protocole d'écoute SASL/PLAINTEXT sur l'adresse localhost:9092.

- Définition de **advertised.listeners** à **SASL\_PLAINTEXT://localhost:9092** pour spécifier l'adresse annoncée aux clients.
- Configuration de **security.inter.broker.protocol** à **SASL\_PLAINTEXT** pour spécifier le protocole de sécurité inter-brokers.
- Définition de **sasl.mechanism.inter.broker.protocol** à **PLAIN** pour spécifier le mécanisme d'authentification SASL utilisé pour les connexions inter-brokers.
- Spécification de **sasl.enabled.mechanisms** à **PLAIN** pour activer le mécanisme d'authentification SASL PLAIN.

## 2. Création du fichier JAAS (kafka\_server\_jaas.conf) :



```

1  KafkaServer {
2      org.apache.kafka.common.security.plain.PlainLoginModule required
3      username="admin"
4      password="admin-secret"
5      user_admin="admin-secret";
6  };

```

Dans le fichier **kafka\_server\_jaas.conf**, nous avons configuré le module d'authentification **PlainLoginModule** pour le broker Kafka. Nous avons défini un utilisateur appelé **admin** avec le mot de passe **admin-secret**. Le module d'authentification **PlainLoginModule** est utilisé pour l'authentification simple à l'aide d'un nom d'utilisateur et d'un mot de passe. Dans ce cas, nous avons spécifié les informations d'identification de l'utilisateur admin directement dans le fichier **JAAS**.

La configuration de l'utilisateur admin avec le nom d'utilisateur admin et le mot de passe admin-secret permet à cet utilisateur de s'authentifier auprès du broker Kafka en utilisant ces informations. Lorsqu'un Producer ou un Consumer tente de se connecter au broker Kafka, il devra fournir les mêmes informations d'identification pour être autorisé à accéder aux ressources Kafka.

## 3. Ajout de la ligne suivante dans le fichier kafka-server-start.bat :

«Set **KAFKA\_OPTS=-Djava.security.auth.login.config=%~dp0../config/kafka\_server\_jaas.conf**» définit les options d'exécution (**KAFKA\_OPTS**) pour le démarrage du broker Kafka. Lorsque le broker Kafka est démarré, il charge les propriétés spécifiées dans les options d'exécution. En définissant **java.security.auth.login.config** sur le chemin du fichier JAAS, nous indiquons au broker Kafka de charger cette configuration JAAS lors de l'initialisation de l'authentification SASL/PLAIN.

Ces étapes assurent que le broker Kafka est configuré pour utiliser l'authentification SASL/PLAIN, avec le fichier JAAS spécifié pour valider les informations d'identification des clients qui tentent de se connecter.

## Configuration du Producer et Consumer :

Dans la configuration du **producer** et du **consumer**, nous avons ajouté les lignes suivantes aux fichiers de propriétés **kc-etudiants.properties** et **genkp.properties** :

- **spring.kafka.jaas.enabled=true** : Cette ligne active l'utilisation de la configuration JAAS pour l'authentification du producer et du consumer Kafka.
- **spring.kafka.properties.security.protocol=SASL\_PLAINTEXT** : Cette ligne spécifie le protocole de sécurité utilisé pour la communication avec le broker Kafka, qui est ici défini sur SASL\_PLAINTEXT pour une connexion sécurisée SASL/PLAIN.
- **spring.kafka.properties.sasl.mechanism=PLAIN** : Cette ligne spécifie le mécanisme d'authentification utilisé pour la connexion sécurisée, qui est ici défini sur PLAIN pour l'authentification SASL/PLAIN.
- **spring.kafka.properties.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \ username="admin" \ password="admin-secret";** : Cette ligne spécifie la configuration JAAS utilisée pour l'authentification du producer et du consumer. Elle indique l'utilisation du module d'authentification PlainLoginModule fourni par Apache Kafka, avec le nom d'utilisateur **admin** et le mot de passe **admin-secret**.

Ces configurations permettent au producer et au consumer d'établir une connexion sécurisée avec le broker Kafka en utilisant l'authentification SASL/PLAIN. Les informations d'identification fournies (nom d'utilisateur et mot de passe) doivent correspondre à celles définies dans la configuration JAAS du broker Kafka pour que la connexion soit autorisée.



## Résultats:

On execute le script qui lance zookeeper,kafka broker,consumer et le producer.

On verifie au debut si SASL fonctionne:

Dans le broker on remarque que SASL est activé:

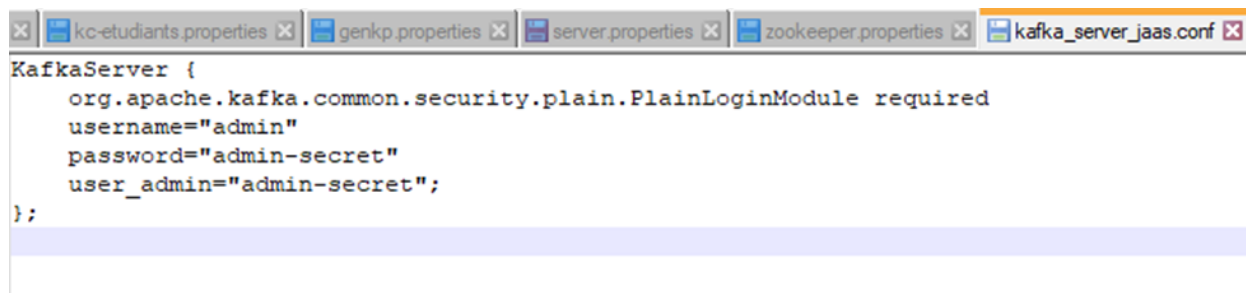
```
BROKER - C:\kafka\kafka_2.12 x + v
sasl.oauthbearer.jwks.endpoint.url = null
sasl.oauthbearer.scope.claim.name = scope
sasl.oauthbearer.sub.claim.name = sub
sasl.oauthbearer.token.endpoint.url = null
sasl.server.callback.handler.class = null
sasl.server.max.receive.size = 524288
security.inter.broker.protocol = SASL_PLAINTEXT
security.providers = null
```

Dans le consumer:

```
KC-ETUDIANTS - run.cmd x + v
sasl.kerberos.ticket.renew.jitter = 0.05
sasl.kerberos.ticket.renew.window.factor = 0.8
sasl.login.callback.handler.class = null
sasl.login.class = null
sasl.login.refresh.buffer.seconds = 300
sasl.login.refresh.min.period.seconds = 60
sasl.login.refresh.window.factor = 0.8
sasl.login.refresh.window.jitter = 0.05
sasl.mechanism = PLAIN
security.protocol = SASL_PLAINTEXT
```

On donne un mot de passe incorrect d'utilisateur au producer et consumer :

```
spring.kafka.properties.sasl.jaas
  username="admin" \
  password="admin-sec";
```



```
KafkaServer {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="admin"
  password="admin-secret"
  user_admin="admin-secret";
};
```

On peut voir que sur Kafka-server-jaas.conf on a le mot de passe de l'utilisateur qu'il faut utiliser pour se connecter au broker Kafka est admin-secret, alors que sur le fichier kc-etudiants.properties du consumer on a écrit un mot de passe incorrect : admin-sec. Cela a empêché l'établissement de la connexion entre le broker et le consumer :

```
[2023-06-07 23:29:17,202] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Failed authentication with /127.0.0.1 (channelId=127.0.0.1:9092-127.0.0.1:58393-36) (Authentication failed: Invalid username or password) (org.apache.kafka.common.network.Selector)
```

```
2023-06-07 23:28:34.204 INFO 6188 --- [ntainer#0-0-C-1] o.apache.kafka.common.network.Selector : [Consumer clientId=consumer-ETUDIANTS_GROUP-1, groupId=ETUDIANTS_GROUP] Failed authentication with localhost/127.0.0.1 (Authentication failed: Invalid username or password)
2023-06-07 23:28:34.209 ERROR 6188 --- [ntainer#0-0-C-1] org.apache.kafka.clients.NetworkClient : [Consumer clientId=consumer-ETUDIANTS_GROUP-1, groupId=ETUDIANTS_GROUP] Connection to node -1 (localhost/127.0.0.1:9092) failed authentication due to: Authentication failed: Invalid username or password
```