

TP Intergiciel et programmation par composants

Partie I :

- 1) La couche **SASL** (Simple Authentication and Security Layer) est une structure fournissant des services d'authentification et de sécurité des données aux protocoles Internet.
- 2) **SASL** est l'acronyme de Simple Authentication and Security Layer. **SASL** est une couche d'abstraction utilisée dans les protocoles de communication pour ajouter une couche d'authentification et de sécurité.

SASL permet à un client et à un serveur de négocier et d'établir un mécanisme d'authentification mutuellement acceptable.
- 3) **SASL** (Simple Authentication and Security Layer) repose sur les principes suivants :
 - a) **Abstraction de l'authentification** : SASL fournit une couche d'abstraction pour la gestion de l'authentification. Les applications n'ont pas à gérer directement les détails de l'authentification.
 - b) **Flexibilité des mécanismes** : SASL supporte différents mécanismes d'authentification, tels que les mots de passe, les certificats numériques, les jetons d'accès, etc., permettant aux parties de choisir celui qui convient le mieux.
 - c) **Négociation** : SASL facilite la négociation entre le client et le serveur pour sélectionner un mécanisme d'authentification mutuellement acceptable.
 - d) **Sécurité des données** : SASL offre des services supplémentaires de sécurité tels que l'intégrité des données, la confidentialité et la protection contre la relecture,

- e) **Indépendance du protocole** : SASL peut être utilisé dans différents protocoles réseau, assurant une couche d'authentification et de sécurité commune et cohérente entre les différents protocoles.

Partie II :

1) SASL/GSSAPI (Kerberos) :

Caractéristiques principales : Utilise le protocole Kerberos pour l'authentification sécurisée basée sur des tickets. Il s'appuie sur une infrastructure d'autorisation centralisée, où les utilisateurs sont authentifiés par un service d'authentification central (AS) et obtiennent un ticket de service pour accéder aux services.

Apport sur la sécurité d'accès à Kafka : Il offre une authentification forte et sécurisée basée sur des tickets Kerberos, garantissant l'identité des utilisateurs et empêchant les accès non autorisés.

2) SASL/PLAIN :

Caractéristiques principales : Utilise une authentification simple basée sur des paires d'identifiants (nom d'utilisateur et mot de passe) en texte clair. Il ne fournit pas de mécanisme de chiffrement ou de protection des données d'authentification.

Apport sur la sécurité d'accès à Kafka : Il offre une méthode d'authentification basique mais non sécurisée, ne convenant généralement pas aux environnements de production. Il est principalement utilisé pour des tests ou des développements locaux.

3) SASL/SCRAM-SHA-256 et SASL/SCRAM-SHA-512 :

Caractéristiques principales : Utilise le mécanisme SCRAM (Salted Challenge Response Authentication Mechanism) pour l'authentification basée sur des mots de passe sécurisés.

SCRAM-SHA-256 utilise une fonction de hachage SHA-256, tandis que SCRAM-SHA-512 utilise SHA-512.

Apport sur la sécurité d'accès à Kafka : Ils offrent une authentification sécurisée basée sur des mots de passe, empêchant l'utilisation de mots de passe en texte clair. Les mots de passe sont stockés sous forme de hachages sécurisés, garantissant une protection supplémentaire des informations d'identification.

4) SASL/OAUTHBEARER :

Caractéristiques principales : Utilise le protocole OAuth pour l'authentification et l'autorisation basées sur des jetons. Il permet l'intégration avec des fournisseurs d'identité externes, tels que Google, Facebook, etc.

Apport sur la sécurité d'accès à Kafka : Il permet l'authentification basée sur des jetons OAuth, offrant une intégration avec des fournisseurs d'identité tiers et une gestion centralisée des autorisations.

En ce qui concerne ZooKeeper, il est recommandé de le sécuriser également. ZooKeeper est utilisé par Kafka pour stocker les métadonnées et les configurations, et il est important de protéger l'accès à ces informations sensibles. On peut sécuriser ZooKeeper en utilisant des mécanismes tels que SASL ou TLS (Transport Layer Security).

ZooKeeper est essentiel pour le fonctionnement de Kafka en tant que registre de métadonnées. Cependant, à partir de la version 2.8 de Kafka, une nouvelle fonctionnalité appelée **Kafka Raft Metadata Mode** est introduite, qui permet à Kafka de fonctionner sans dépendre de ZooKeeper pour le stockage des métadonnées. Cette fonctionnalité utilise un protocole de consensus appelé Raft pour la réplication et la coordination des métadonnées Kafka. Cela signifie que Kafka peut fonctionner sans ZooKeeper dans ce mode.

En ce qui concerne l'utilisation de SASL avec Kafka en mode sans ZooKeeper, Kafka Raft Metadata Mode prend en charge l'authentification SASL en utilisant les mêmes mécanismes SASL mentionnés précédemment.

Partie III :

1. Lancer les dockers containers avec postgres et pgadmin
2. Lancer Zookeeper, le broker, le consumer et le producer.

Mode non sécurisée :

zookeeper.properties

```

1 # the directory where the snapshot is stored.
2 dataDir=/tmp/zookeeper
3 # the port at which the clients will connect
4 clientPort=2181
5 # disable the per-ip limit on the number of connections since this is a non-production config
6 maxClientCnxns=0
7 # Disable the adminserver by default to avoid port conflicts.
8 # Set the port to something non-conflicting if choosing to enable this
9 admin.enableServer=true
10 admin.serverPort=8080

```

server.properties

```

1 ##### Server Basics #####
2 broker.id=0
3 ##### Socket Server Settings #####
4 listeners=PLAINTEXT://localhost:9092
5 num.network.threads=3
6 num.io.threads=8
7 socket.send.buffer.bytes=102400
8 socket.receive.buffer.bytes=102400
9 socket.request.max.bytes=104857600
10 ##### Log Basics #####
11 log.dirs=/tmp/kafka-logs
12 num.partitions=1
13 num.recovery.threads.per.data.dir=1
14 ##### Internal Topic Settings #####
15 offsets.topic.replication.factor=1
16 transaction.state.log.replication.factor=1
17 transaction.state.log.min.isr=1
18 delete.topic.enable=true
19 ##### Log Retention Policy #####
20 log.retention.check.interval.ms=300000
21 ##### Zookeeper #####
22 zookeeper.connect=localhost:2181
23 zookeeper.connection.timeout.ms=18000
24 ##### Group Coordinator Settings #####
25 group.initial.rebalance.delay.ms=0

```

genkp.properties

```
1 server.port=7000
2
3 spring.kafka.producer.bootstrap-servers=localhost:9092
4 spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
5 spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
6
7 #definition du nom du topic pour ce service producer
8 application.topic=ETUDIANTS
9
```

kc-etudiants.properties

```
1 server.port=7100
2
3 spring.kafka.consumer.bootstrap-servers=localhost:9092
4 spring.kafka.consumer.topic-name=ETUDIANTS
5 spring.kafka.consumer.group-id=ETUDIANTS_GROUP
6 spring.kafka.consumer.auto-offset-reset=earliest
7 spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
8 spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
9
10 spring.datasource.url=jdbc:postgresql://localhost:5432/etudiants
11 spring.datasource.username=postgres
12 spring.datasource.password=admin
13 spring.datasource.driverClassName=org.postgresql.Driver
```

GET request

GET ⌵ http://localhost:7000/insa/isalive Send

Status: 200 OK Size: 16 Bytes Time: 8 ms

Response Headers ⁴ Cookies Results Docs {} ≡

1 {
2 "reponse": "42"
3 }

POST request

The screenshot shows a REST client interface with a POST request to `http://localhost:7000/insa/kppublish`. The request body is a JSON object with the following content:

```
{
  "identifiant" : "12345",
  "nom" : "ALMEIDA",
  "prenom" : "ANDRES"
}
```

The response status is **201 Created**, with a size of **22 Bytes** and a time of **938 ms**. The response body is a JSON object:

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

Base de données

The screenshot shows a database query interface with a SQL query:

```
1 select * from etudiants
2
```

The query results are displayed in a table with the following columns: **identifiant** (character varying), **nom** (character varying), and **prenom** (character varying). The results show one row with the following values:

	identifiant character varying	nom character varying	prenom character varying
1	12345	ALMEIDA	ANDRES

Mode SASL/PLAIN :***Mode non sécurisée :******zookeeper.properties***

```

1 # the directory where the snapshot is stored.
2 dataDir=/tmp/zookeeper
3 # the port at which the clients will connect
4 clientPort=2181
5 # disable the per-ip limit on the number of connections since this is a non-production config
6 maxClientCnxns=0
7 # Disable the adminserver by default to avoid port conflicts.
8 # Set the port to something non-conflicting if choosing to enable this
9 admin.enableServer=true
10 admin.serverPort=8080

```

server_sasl_plain.properties

```

1 ##### Server Basics #####
2 broker.id=0
3 ##### Socket Server Settings #####
4 listeners=SASL_PLAINTEXT://localhost:9092
5 advertised.listeners=SASL_PLAINTEXT://localhost:9092
6 num.network.threads=3
7 num.io.threads=8
8 socket.send.buffer.bytes=102400
9 socket.receive.buffer.bytes=102400
10 socket.request.max.bytes=104857600
11 ##### Log Basics #####
12 log.dirs=/tmp/kafka-logs
13 num.partitions=1
14 num.recovery.threads.per.data.dir=1
15 ##### Internal Topic Settings #####
16 offsets.topic.replication.factor=1
17 transaction.state.log.replication.factor=1
18 transaction.state.log.min.isr=1
19 delete.topic.enable=true
20 ##### Log Retention Policy #####
21 log.retention.check.interval.ms=300000
22 ##### Zookeeper #####
23 zookeeper.connect=localhost:2181
24 zookeeper.connection.timeout.ms=18000
25 ##### Group Coordinator Settings #####
26 group.initial.rebalance.delay.ms=0
27 ##### SASL #####
28 security.inter.broker.protocol=SASL_PLAINTEXT
29 sasl.enabled.mechanisms=PLAIN
30 sasl.mechanism.inter.broker.protocol=PLAIN
31 listener.name.sasl_plaintext.plain.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
32     username="admin" \
33     password="admin-secret" \
34     user_admin="admin-secret" \
35     user_consumer="consumer-secret" \
36     user_producer="producer-secret";

```

genkp.properties

```
1 server.port=7000
2
3 spring.kafka.producer.bootstrap-servers=localhost:9092
4 spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
5 spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
6
7 #definition du nom du topic pour ce service producer
8 application.topic=ETUDIANTS
9
10 spring.kafka.jaas.enabled=true
11 spring.kafka.properties.security.protocol=SASL_PLAINTEXT
12 spring.kafka.properties.sasl.mechanism=PLAIN
13 spring.kafka.properties.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
14     username="producer" \
15     password="producer-secret";
```

kc-etudiants.properties

```
1 server.port=7100
2
3 spring.kafka.consumer.bootstrap-servers=localhost:9092
4 spring.kafka.consumer.topic-name=ETUDIANTS
5 spring.kafka.consumer.group-id=ETUDIANTS_GROUP
6 spring.kafka.consumer.auto-offset-reset=earliest
7 spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
8 spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
9
10 spring.datasource.url=jdbc:postgresql://localhost:5432/etudiants
11 spring.datasource.username=postgres
12 spring.datasource.password=admin
13 spring.datasource.driverClassName=org.postgresql.Driver
14
15 spring.kafka.jaas.enabled=true
16 spring.kafka.properties.security.protocol=SASL_PLAINTEXT
17 spring.kafka.properties.sasl.mechanism=PLAIN
18 spring.kafka.properties.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
19     username="consumer" \
20     password="consumer-secret";
```

GET request

GET

http://localhost:7000/insa/isalive

Send

Query

Headers ²

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

Status: 200 OK

Size: 16 Bytes

Time: 179 ms

Response

▼

1 {

2 "reponse": "42"

3 }

POST request

The screenshot shows a REST client interface with a POST request to `http://localhost:7000/insa/kppublish`. The request body is a JSON object with the following fields: `identifiant` (123456), `nom` (TONDEUR), and `prenom` (HERVE). The response status is 201 Created, with a size of 22 Bytes and a time of 2.89 s. The response body is a JSON object with the field `reponse` (Inserted).

```
POST http://localhost:7000/insa/kppublish
```

Query Headers² Auth **Body¹** Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

```
1 {
2   "identifiant": "123456",
3   "nom": "TONDEUR",
4   "prenom": "HERVE"
5 }
```

Status: 201 Created Size: 22 Bytes Time: 2.89 s Response ▾

```
1 {
2   "reponse": "Inserted"
3 }
```

Copy

Base de données

The screenshot shows a database query interface with a SQL query: `select * from etudiants`. The results are displayed in a table with columns: `identifiant` (character varying), `nom` (character varying), and `prenom` (character varying). The results show two rows of data.

	identifiant character varying	nom character varying	prenom character varying
1	12345	ALMEIDA	ANDRES
2	123456	TONDEUR	HERVE