Data Science & Big Data

Master DSBD

INTERNET OF THINGS (IOT) SECURITY

WORKSHOP

directed by: KHADIJA ACHTAICH

by: AMGHAR HIND
DEKDEGUE HAJAR
BRIDILA HIND

PROJECT OVERVIEW

Imagine Vélib' stations as a network of smart devices constantly sending updates. This project focuses on keeping that data safe! We built a system that collect real-time information about bike availability at these stations across different cities. This system treats the stations like part of the "Internet of Things" (IoT) and makes sure the data, especially locations, is secure throughout the process.

API INTEGRATION

The project utilizes the JCDecaux API (https://developer.jcdecaux.com/) to retrieve real-time data on Vélib' stations.
This API provides information about each station, including its address, available bike stands, and status.

OBTAINING AN API KEY

- Create an account on the JCDecaux Developer Portal: https://developer.jcdecaux.com/#/signup
- Once your account is created, you will be provided with an API key in your user profile.
- Verify your API key by retrieving the list of all stations using the following command: curl https://api.jcdecaux.com/vls/v1/stations?apiKey=YOUR_API_KEY
- Replace YOUR_API_KEY with your actual API key.
- The response should be a JSON object containing information about all stations.

API USAGE

The JCDecaux API provides various endpoints for retrieving different types of data. For this project, you will primarily use the stations endpoint to get information about individual stations.
curl https://api.jcdecaux.com/vls/v1/stations?apiKey=YOUR_API_KEY&contract=Dublin
This call will retrieve data for all Vélib' stations in Dublin.

RESPONSE DATA

The API response is a JSON object containing an array of station objects. Each station object has the following properties:

- address: The address of the station
- available_bike_stands: The number of available bike stands at the station
- available_bikes: The number of available bikes at the station
- banking: Indicates whether the station has a payment terminal
- bike_stands: The total number of bike stands at the station
- bonus: Indicates whether the station is part of a bonus program
- contract_name: The name of the Vélib' system (e.g., "Marseille", "Dublin")
- last_update: The timestamp of the last data update
- name: The name of the station
- number: A unique identifier for the station
- position: An object containing the latitude and longitude coordinates of the station
- status: The status of the station (e.g., "OPEN", "CLOSED")

ENVIRONMENT SETUP

To install Java :
java -version
sudo yum install java-1.8.0-openjdk
java -version

Environment Setup

Install Dependencies :

Kafka : Download the latest Kafka binary package: wget https://dlcdn.apache.org/kafka/3.4.0/kafka_2.13-3.4.0.tgz

Extract the downloaded archive: tar -xvf kafka_2.13-3.4.0.tgz

Go to the extracted Kafka directory: cd kafka_2.13-3.4.0

Python Libraries :

- Install the kafka-python library for interacting with Kafka from Python: pip install kafka-python
- Install the cryptography library for encryption: pip install cryptography

Start Kafka and Zookeeper Services

- Start zookeeper: Refer to the Kafka documentation for starting the ZooKeeper : **/bin/zookeeper-server-start.sh ./config/zookeeper.properties**
- Start Kafka: Refer to the Kafka documentation for starting the Kafka broker :**bin/kafka-server-start.sh ./config/server.properties**

ENVIRONMENT SETUP

Environment Setup

Start MongoDB Service

a. Install MongoDB : Follow the official MongoDB installation instructions for Ubuntu:
https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/
The instructions typically involve adding the MongoDB repository key, updating package lists, and installing the mongodb-org package.

b. Start MongoDB Server : After installation, you can start the MongoDB server using:
sudo service mongod start

c. Verify MongoDB : Check the status of the MongoDB server with:
sudo service mongod status
This should indicate if the MongoDB server is running.

ENVIRONMENT SETUP

Environment Setup

```
hind@hind-VM:~/kafka$ ls
bin  config  data  libs  LICENSE  licenses  logs  NOTICE  site-docs
hind@hind-VM:~/kafka$  ./bin/zookeeper-server-start.sh ./config/zookeeper.properties
[2024-05-11 12:44:41,679] INFO Reading configuration from: ./config/zookeeper.properties (org.apache.zookeeper.server.quor
um.QuorumPeerConfig)
[2024-05-11 12:44:41,696] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-11 12:44:41,697] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-11 12:44:41,697] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-11 12:44:41,697] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.
apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-11 12:44:41,705] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-05-11 12:44:41,705] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-05-11 12:44:41,711] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2024-05-11 12:44:41,711] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zoo
keeper.server.quorum.QuorumPeerMain)
[2024-05-11 12:44:41,717] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2024-05-11 12:44:41,720] INFO Reading configuration from: ./config/zookeeper.properties (org.apache.zookeeper.server.quor
um.QuorumPeerConfig)
[2024-05-11 12:44:41,725] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-11 12:44:41,725] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-11 12:44:41,725] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-11 12:44:41,725] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.
apache.zookeeper.server.quorum.QuorumPeerConfig)
[2024-05-11 12:44:41,726] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2024-05-11 12:44:41,793] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsPro
vider@6574b225 (org.apache.zookeeper.server.ServerMetrics)
[2024-05-11 12:44:41,804] INFO ACL digest algorithm is: SHA1 (org.apache.zookeeper.server.auth.DigestAuthenticationProvide
r)
[2024-05-11 12:44:41,804] INFO zookeeper.DigestAuthenticationProvider.enabled = true (org.apache.zookeeper.server.auth.Dig
estAuthenticationProvider)
[2024-05-11 12:44:41,816] INFO zookeeper.snapshot.trust.empty : false (org.apache.zookeeper.server.persistence.FileTxnSnap
```

```
hind@hind-VM:~/kafka$ ./bin/kafka-server-start.sh ./config/server.properties
[2024-05-11 12:45:13,311] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2024-05-11 12:45:14,800] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation
(org.apache.zookeeper.common.X509Util)
[2024-05-11 12:45:15,041] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHandler)
[2024-05-11 12:45:15,047] INFO starting (kafka.server.KafkaServer)
[2024-05-11 12:45:15,048] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2024-05-11 12:45:15,093] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperCl
ient)
[2024-05-11 12:45:15,108] INFO Client environment:zookeeper.version=3.8.2-139d619b58292d7734b4fc83a0f44be4e7b0c986, built on 2023-07-05
19:24 UTC (org.apache.zookeeper.ZooKeeper)
[2024-05-11 12:45:15,108] INFO Client environment:host.name=hind-VM (org.apache.zookeeper.ZooKeeper)
[2024-05-11 12:45:15,108] INFO Client environment:java.version=1.8.0_362 (org.apache.zookeeper.ZooKeeper)
[2024-05-11 12:45:15,108] INFO Client environment:java.vendor=Private Build (org.apache.zookeeper.ZooKeeper)
[2024-05-11 12:45:15,108] INFO Client environment:java.home=/usr/lib/jvm/java-8-openjdk-amd64/jre (org.apache.zookeeper.ZooKeeper)
[2024-05-11 12:45:15,109] INFO Client environment:java.class.path=/home/hind/kafka/bin/../libs/activation-1.1.1.jar:/home/hind/kafka/bin
/../libs/aopalliance-repackaged-2.6.1.jar:/home/hind/kafka/bin/../libs/argparse4j-0.7.0.jar:/home/hind/kafka/bin/../libs/audience-annota
tions-0.12.0.jar:/home/hind/kafka/bin/../libs/caffeine-2.9.3.jar:/home/hind/kafka/bin/../libs/checker-qual-3.19.0.jar:/home/hind/kafka/b
in/../libs/commons-beanutils-1.9.4.jar:/home/hind/kafka/bin/../libs/commons-cli-1.4.jar:/home/hind/kafka/bin/../libs/commons-collections
-3.2.2.jar:/home/hind/kafka/bin/../libs/commons-digester-2.1.jar:/home/hind/kafka/bin/../libs/commons-io-2.11.0.jar:/home/hind/kafka/bin
/../libs/commons-lang3-3.8.1.jar:/home/hind/kafka/bin/../libs/commons-logging-1.2.jar:/home/hind/kafka/bin/../libs/commons-validator-1.7
.jar:/home/hind/kafka/bin/../libs/connect-api-3.6.0.jar:/home/hind/kafka/bin/../libs/connect-basic-auth-extension-3.6.0.jar:/home/hind/k
afka/bin/../libs/connect-json-3.6.0.jar:/home/hind/kafka/bin/../libs/connect-mirror-3.6.0.jar:/home/hind/kafka/bin/../libs/connect-mirro
r-client-3.6.0.jar:/home/hind/kafka/bin/../libs/connect-runtime-3.6.0.jar:/home/hind/kafka/bin/../libs/connect-transforms-3.6.0.jar:/hom
```

DATA PIPELINE IMPLEMENTATION

Producer:

The producer script (producer.py) continuously retrieves data from the JCDecaux API and sends it to the Kafka topic "velib-stations". Each station's data is represented as a JSON object.

```python
import json
import time
import urllib.request

from kafka import KafkaProducer

API_KEY = "eb589860609ef6e0c846beaecbfe0d378fb604f5" # FIXME Set your own API key here
url = "https://api.jcdecaux.com/vls/v1/stations?apiKey={}".format(API_KEY)

producer = KafkaProducer(bootstrap_servers="localhost:9092")

while True:
    response = urllib.request.urlopen(url)
    stations = json.loads(response.read().decode())
    for station in stations:
        producer.send("velib-stations", json.dumps(station).encode())
    print("{} Produced {} station records".format(time.time(), len(stations)))
    time.sleep(1)
```

DATA PIPELINE IMPLEMENTATION

Consumer:

The consumer script (encryption.py) consumes messages from the "velib-stations" topic, encrypts the data, and stores it in MongoDB.

```python
import json
from cryptography.fernet import Fernet
from pymongo import MongoClient
from kafka import KafkaConsumer
import  os
# Generate a key
def generate_key(filename):
    key = Fernet.generate_key()
    with open(filename, 'wb') as key_file:
        key_file.write(key)
    return key

# Load the encryption key
def load_key(filename):
    with open(filename, 'rb') as file:

# Encrypt the sensitive fields
def encrypt_data(data, key):
    f = Fernet(key)
    encrypted_data = {}
    for key, value in data.items():
        if key in ["address", "contract_name"]:
            encrypted_data[key] = f.encrypt(value.encode()).decode()
        else:
            encrypted_data[key] = value
    return encrypted_data
```

DATA PIPELINE IMPLEMENTATION

```python
# Connect to MongoDB
client = MongoClient('localhost', 27017)
db = client['iot']
collection = db['bikes']

# Generate or load the encryption key
encryption_key_file = 'encryption_key.key'
if not os.path.isfile(encryption_key_file):
    key = generate_key(encryption_key_file)
else:
    key = load_key(encryption_key_file)

# Consume messages from Kafka
consumer = KafkaConsumer("velib-stations", bootstrap_servers='localhost:9092', group_id="velib-monitor-stations")
for message in consumer:
    # Decode the message
    station = json.loads(message.value.decode())

    # Encrypt sensitive fields
    encrypted_station = encrypt_data(station, key)

    # Insert the encrypted data into MongoDB
    collection.insert_one(encrypted_station)

    print("Message inserted into MongoDB:", encrypted_station)
```

DATA PIPELINE IMPLEMENTATION

Running the Application :

Start the Kafka producer:

- Run the producer.py script

Start the Kafka consumer:

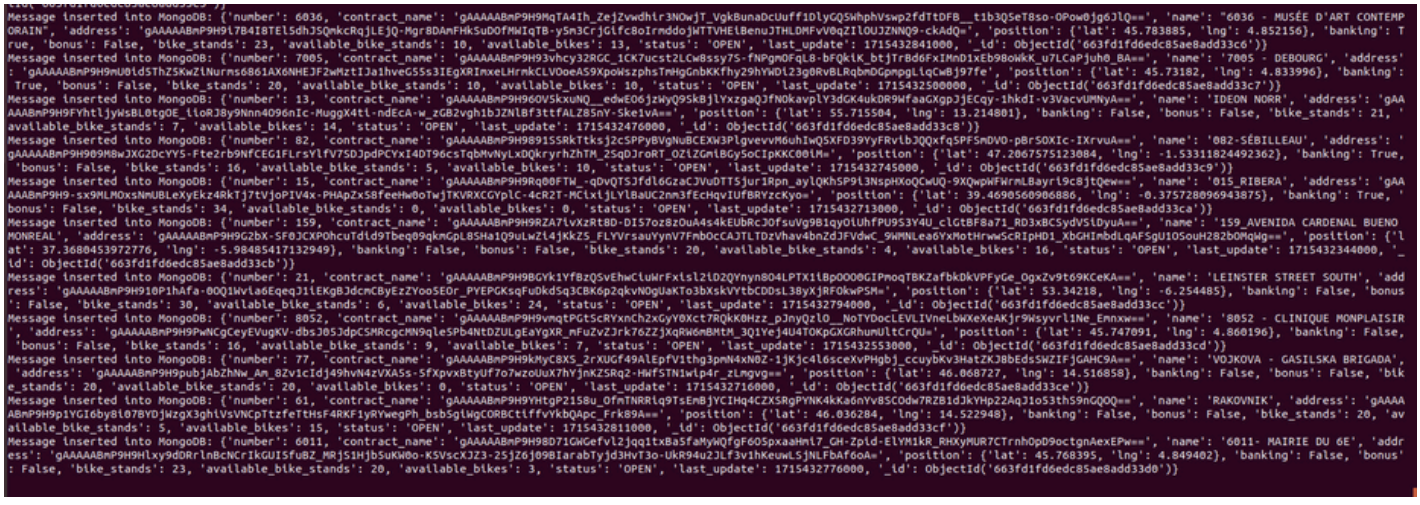
- Run the encryption.py script

```
hind@hind-VM:~/Documents/projectKafka_dataSecurity/kafka_iot_workshop$ cd test
hind@hind-VM:~/Documents/projectKafka_dataSecurity/kafka_iot_workshop/test$ python producer.py
1715458053.2014604 Produced 2386 station records
1715458057.9610445 Produced 2386 station records
1715458062.066862 Produced 2386 station records
1715458068.5264468 Produced 2386 station records
1715458074.5271463 Produced 2386 station records
1715458079.339887 Produced 2386 station records
1715458084.565342 Produced 2386 station records
1715458092.6178777 Produced 2386 station records
1715458098.838114 Produced 2386 station records
1715458108.6980298 Produced 2386 station records
1715458117.7551494 Produced 2386 station records
1715458124.8342514 Produced 2386 station records
1715458130.6536314 Produced 2386 station records
1715458137.6636825 Produced 2386 station records
1715458141.9060369 Produced 2386 station records
1715458145.8879933 Produced 2386 station records
1715458149.8405006 Produced 2386 station records
1715458153.4267428 Produced 2386 station records
1715458156.72784 Produced 2386 station records
1715458160.2970488 Produced 2386 station records
1715458163.4794018 Produced 2386 station records
1715458166.6064203 Produced 2386 station records
1715458169.5870056 Produced 2386 station records
1715458172.8504934 Produced 2386 station records
1715458176.7411826 Produced 2386 station records
1715458182.4587266 Produced 2386 station records
1715458185.8454812 Produced 2386 station records
1715458188.990172 Produced 2386 station records
1715458192.41722 Produced 2386 station records
1715458195.8354719 Produced 2386 station records
1715458199.0422003 Produced 2386 station records
```

encryption.py 3 | encryption_key.key

test > encryption_key.key

```
1  FcsKpVxQAJsKi_f_mIrDBG_5FWDZpz7-6k6y1E3OqZE=
```



ENCRYPTION METHOD

The encryption method used in this project is called symmetric encryption with the specific algorithm being Advanced Encryption Standard (AES).
Here's a breakdown of how it works:

- **Shared Secret Key:** Both the producer and consumer share a secret key, which acts like a password for encryption and decryption. This key is crucial for both processes. In the provided code, the key is generated or loaded from a file (encryption_key.key).
- **Encryption:**

1. The producer identifies specific fields in the station data that are considered sensitive, such as "address" and "contract_name".
2. For these sensitive fields, the producer uses the Fernet library, which implements the AES algorithm, to encrypt the data.
3. The encryption process transforms the data into an unreadable format using the shared secret key.

- **Non-Sensitive Data:** Fields like "available_bike_stands" and "status" are not encrypted because they likely don't contain confidential information.

Security Considerations:

- **Key Management:** The security of this implementation heavily relies on the proper management of the secret key. If the key is leaked or compromised, anyone with access to it can decrypt the sensitive data.

- **Encryption at Rest:** The provided code encrypts data before storing it in MongoDB. However, it's essential to consider additional security measures like encryption at rest offered by MongoDB itself for enhanced protection.