IOT#BC-TRACE

Pablo Santiago Guilarte Hugo de la Camara Saiz

DESCRIPTION AND OBJECTIVE

Objective: define a prototype tool to store traces of IoT data on a public platform like Ethereum.

Description:

- Data are provided from: IoT message generator
- Data are ingested in real time from MQTT broker and analysed with Apache Kafka, guaranteeing scalability and performance.
- Output analytics was saved on an external system: Ethereum Blockchain

TECHNOLOGIES

- Ubuntu 18.04
- Java 8
- MQTT
- Apache Kafka
- Ethereum
- Git and GitHub

LOGICS REQUIRED

Logic 1 - Write a Kafka Stream component that subscribes on the topic MQTT and:

Implements a window function (eg. 5 minutes) to group all the messages that belong to the same time window

Produce a new message on topic IOTRACE with the following JSON format:

{

"tz": "2020-04-06T10:50:39.137+02:00", // ISODATE of the kafka message

"uuids": ["UUID1","UUID2", ...]

LOGICS REQUIRED

Logic 2 - Write a prototype (e.g. in node.js) that subscribes to IOTRACE topic and persist the message on Ethereum

Logic 3 - Configure a Console consumer that dumps the IoT message from MQTT topic to a file. This file will be used to retrieve the original IoT messages from the UUIDs stored in Ethereum

TECHNOLOGIES

Thanks to MQTT we are able to complete the 1st and the 3rd logic

Thanks to Apache Kafka we are able to complete all the logics

Thanks to Ethereum we are able to complete the 2nd logic







First we need to run Kafka:

KAFKA_HOME=/opt/kafka

\$KAFKA_HOME/bin/connect-distributed.sh \$KAFKA_HOME/config/connect-distributed.properties

```
| Section | Content of the Content o
```

Then, go to the simulator and run it too:

docker-compose up -d

docker-compose logs -f

Then connect the connector MQTT of the simulator with Kafka:

curl -s -X POST -H 'Content-Type: application/json' http://localhost:8083/connectors -d @mqtt_connect.json

```
pablo@ubuntu:~$ curl -s -X POST -H 'Content-Type: application/json' http://localhost:8083/con nectors -d @mqtt_connect.json {"name":"mqtt-source", "config":{"connector.class":"io.confluent.connect.mqtt.MqttSourceConnector", "tasks.max":"l", "mqtt.server.uri":"tcp://localhost:1883", "mqtt.topics":"#", "kafka.topic":"mqtt.echo", "value.converter":"org.apache.kafka.connect.converters.ByteArrayConverter", "key.converter":"org.apache.kafka.connect.storage.StringConverter", "key.converter.schemas.enable": "false", "value.converter.schemas.enable": "false", "confluent.topic.bootstrap.servers": "localhost:9092", "confluent.topic.replication.factor":"l", "confluent.license":"", "name": "mqtt-source"
```

Run the app that filters the streamings:

mvn clean package

COMPILE

ANAMEND TO the organized and the property of t

mvn exec:java -Dexec.mainClass=myapps.Streamings

EXECUTE

```
| Description |
```

At this moment MQTT is going to start generating messages:

pablo@ubuntu:-/Project\$ \$KAFKA HOME/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic mqtt.echo > Topic.txt

["t":1676359919304, "tz":"2023-02-11721:28:30.3042", "unid": "9deat917-af93-4757-adds-df83a272a25a", "cuid": "16264547-72c5-0913-adds-290e50614690", "ref": "j:p://edve0504.0000"
["t":1676159912889, "tz":"2023-02-11721:28:12.8892", "unid": "Addbectc-4ade-4abe-ae91-c3461a276488", "cuid": "abcd550-723f-090e-9f0e-27aaf6480002", "ref": "j:p://edve0504.0000"
["t":1676159912889, "tz": "2023-02-11721:28:44.8802", "unid": "07c7a66-25cf-4afa-bi99-ae653a7fa678", "cuid": "noneta10-sba3-4168-9071-c7663491578", "ref": "j:p://edve0504.0000"
["t":1676159912877, "tz": "2023-02-11721:28:44.2472", "unid": "7cf7fcc8-af22-4bte-8f0b-f0556e", "cuid": "100cc827-b646-4f59-b66d-353bbe5b5718", "ref": "j:p://edve0504.0000"
["t":1676159933572, "tz": "2023-02-11721:28:59-7992", "unid": "660c041a-d78a-483b-a672-5923ee89a1d", "cuid": "600c027-b644-4659-b66d-353be5b5718", "ref": "j:p://edve0504.0000"
["t":167615993572, "tz": "2023-02-11721:28:59-5722", "unid": "680c041a-d78a-483b-a672-5923ee89a1d", "cuid": "606c43b-4e1c-4439-8a63-793aefce7aeb", "ref": "j:p://edve0504.0000"
["t":167615993572, "tz": "2023-02-11721:29:99-9712", "unid": "680c041a-d78a-483b-a672-5923ee89a1d", "cuid": "600c7bec-6064-4442-bae5-900bc17e25a", "ref": "j:p://edve0504.0000"
["t":167615995400971, "tz": "2023-02-11721:29:19-9712", "unid": "680c041a-d78a-4550-4571-6500-767160", "cuid": "600c7bec-6064-4442-bae5-900bc17e25a", "ref": "j:p://edve0504.0000"
["t":1676159954009, "tz": "2023-02-11721:29:14.4652", "unid": "680d516-6650-6153-452-60072f500", "cuid": "600c7bec-6064-4442-bae5-900bc17e25a", "ref": "j:p://edve0504.0000"
["t":1676159954009, "tz": "2023-02-11721:29:14.4652", "unid": "680f5162-bae5-60072f500", "cuid": "61760590599, "tz": "2023-02-11721:29:14.4652", "unid": "680f5162-bae5-600590500", "cuid": "61760590599, "tz": "2023-02-11721:29:14.4652", "unid": "680f5162-bae5-6005905", "cuid": "61760590599, "tz": "2023-02-11721:29:14.4652", "unid": "680f5162-bae5-6005905", "cuid": "61760590599, "tz": "2023-02-11721:29:14.4652", "unid": "680f5162-bae5-6005905", "cuid": "61760

In order to execute the smart contract we must use the command:

nvm use v(Version)

```
pablo@ubuntu:~/Project$ nvm use v18.14.0
Now using node v18.14.0 (npm v9.4.2)
pablo@ubuntu:~/Project$
```

Later we go to the carpet where the deploy of the contract and the smart contract are: node deploy.js

```
psichiments (Treign-Missionia) and deploy. 18

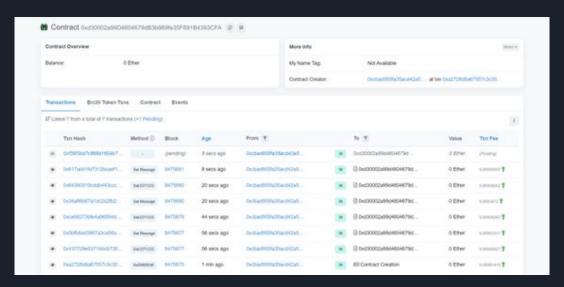
Psichiment Agrees (Salionia) and deploy. 18

Castras (Salionia) and deploy. 19

Castras (Salionia) and deploy
```

In the console the address of the contract is going to be spawned.

With the address we can go to the network Goerli to see its traceability:



ACHIEVED RESULTS

The smart contract in Goerli show us this trace:

```
Deta

Sun Feb 12 22:54:85 UTC 2823|"30bcaf56-afa6-42f2-a5f3-4271f901a465";"3c35481c-05ca-42f8-acaa-97dc525a73d6";"eb05cb41-a493-4610-93ac-dc02c23wbafc";"66cb4

*
```

```
"" 150-042445187, "E" "203 -82 1272154(8).1872" "Wold" "Moudfo die 4272-073-4221901e805" "Cold" "Py788880 (1F1 Merc hole décododaff" "ref" "jgy //edw950.0000" "type" "presente" "T 150-042448833, "E" "NOU -82 1272154(8).1872" "Wold" "NOU -85 4274-4249 NOU -85 4274-
```

FUTURE IMPROVEMENTS

- The contract could be verified with EtherScan, showing the functions in a public way
- The streamings could have other type of data
- A replication factor to work in parallel could be implemented
- An interface that show the messages in real time could be implemented