Blockchain and IoT in Dangerous Goods transportation

Author: Jonathan Lamont

Date: 24-12-2019

Table of Content:

---

1. Overview

Project description

Project which aims to test and measure the behavior of the Hyperledger Fabric blockchain regarding a simple end to end testing scenario including goods transportation and IoT tracing.

We have implemented a simple scenario into different chaincodes / smart-contracts. We have simulated the data providing by IoTs through NodeRED. And we have tested and measured some metrics thanks to bash scripts.

Scenario

Goods have to be carried from a provider to a receiver thanks to a transporter. We assume that goods are still stored in warehouses, except when they are carrying. The provider is in Germany and the receiver is in France. Authorities must be notified for each transport and have to grant the transport, even in case of cross-border transportation. IoT devices can be attached to goods or trucks or warehouses in order to collect data such as humidity rate, GPS geolocation, or temperature.

Details

Needs:

- Docker (docker, docker-compose)
- NodeJS (node, npm)

Uses:

- The **Hyperledger Fabric** blockchain
- The tools Hyperledger **Convector** and **Hurley**

Blockchain network

```
[hurley] - Setup:
        - Channels deployed: 3
            * ch-global
            * ch-de
            * ch-fr
        - Organizations: 7
            * dgprovider:
                - channels:
                    * ch-global
                - users:
                    * admin
```

```
            * User1,
            * User2

    * dgreceiver:
        - channels:
            * ch-global
        - users:
            * admin
            * User1,
            * User2

    * dgtransporter:
        - channels:
            * ch-global
        - users:
            * admin
            * User1,
            * User2

    * localauthorityde:
        - channels:
            * ch-global,
            * ch-de
        - users:
            * admin
            * User1,
            * User2

    * localauthorityfr:
        - channels:
            * ch-global,
            * ch-fr
        - users:
            * admin
            * User1,
            * User2

    * warehousede:
        - channels:
            * ch-global,
            * ch-de
        - users:
            * admin
            * User1,
            * User2

    * warehousefr:
        - channels:
            * ch-global,
            * ch-fr
        - users:
```

```
                    * admin
                    * User1,
                    * User2
```

Blockchain applications / SmartContracts

- **authority** → Local Authority smart contract
- **crossborder** → CrossBorder checkpoint smart contract
- **dg** → DangerousGoods smart contract
- **iot** → IoT smart contract
- **process** → Process smart contract
- **transport** → Transport smart contract
- **warehouse** → Warehouse smart contract

REST API server ports mapping

- **[dgprovider, User1]**
  - *ch-global*
    - **dg** → 8011
    - **iot** → 8012
    - **process** → 8013
- **[dgreceiver, User1]**
  - *ch-global*
    - **dg** → 8021
    - **iot** → 8022
    - **process** → 8023
    - **warehouse** → 8024
- **[dgtransporter, User1]**
  - *ch-global*
    - **dg** → 8031
    - **iot** → 8032
    - **transport** → 8033
    - **process** → 8034
    - **crossborder** → 8035
- **[localauthorityde, User1]**
  - *ch-de*
    - **authority** → 8041
  - *ch-global*
    - **crossborder** → 8042
    - **process** → 8043
- **[localauthorityfr, User1]**
  - *ch-fr*
    - **authority** → 8051
  - *ch-global*
    - **crossborder** → 8052
    - **process** → 8053
- **[warehousede, User1]**
  - *ch-de*
    - **warehouse** → 8061
- **[warehousefr, User1]**
  - *ch-fr*

- **warehouse** → 8071

2. How to use it

2.1. Installations

Prerequisites dependencies

```
# apt update && apt upgrade && apt install nodejs npm docker
docker-compose
```

Configuring Docker

You should add your username/account into the docker group in the file *'/etc/group'*, to avoid using root permissions to connect to the unix socket with Docker.

Configuring NPM

```
$ cd ~
$ mkdir .npm-global
$ echo "prefix=~/.npm-global" > .npmrc

$ nano .profile
→ To add at the end of the file:
# NPM:
export PATH="~/.npm-global/bin/:$PATH"

$ source .profile
```

Installing dependencies

```
$ cd ~
$ npm install -g @worldsibu/convector-cli
$ npm install -g @worldsibu/hurley
$ npm install -g npx
$ npm install -g generator-express-no-stress-typescript
$ npm install -g @worldsibu/conv-rest-api
$ cd bc_and_iot/
$ npm install
```

2.2. Blockchain environment (Hyperledger Fabric)

Starting the environment

```
$ cd bc_and_iot/
$ npm run env:restart
```

Refer to *'/home/$USER/hyperledger-fabric-network/'* folder to get network settings such as machines IP and port, or credentials of the default identities on the network.

Cleaning the environment

```
$ cd bc_and_iot/
$ npm run env:clean
```

2.3. Smart Contracts

Building sources

```
$ cd bc_and_iot/
$ npm run lerna:build
```

Installing a SmartContract on Fabric

```
$ npm run cc:start <smartcontractname>
```

Upgrading a SmartContract on Fabric

```
$ npm run cc:upgrade <smartcontractname> <newversion>
```

2.4. Exposing REST API [optional]

Source: Tutorial

To do once (generating sources):

```
$ cd bc_and_iot/
$ conv-rest-api generate api -c <chaincode name> -f <chaincode
config file>
```

Example:

```
$ conv-rest-api generate api -c dg -f dg.config.json
```

Then, be sure to start the network and instantiate the chaincodes before starting the REST API server:

```
$ cd bc_and_iot/

$ npx lerna bootstrap
$ npx lerna run start --scope server --stream

The two previous commands can be summed up by :
$ npm run api:start
```

To change the identity used to setup the API server, you should use environment variable such as CHAINCODE, CHANNEL, IDENTITYID, IDENTITY, ORG, PORT before starting it. Refer to the files **'env.ts'** and **'pm2.config.json'** in the 'packages/server/' directory for modification.

Note: We raise several API on several ports in order to simulate the different users, chaincodes and channels. See section 1 for the API ports mapping.

2.5. Usage

Submitting transactions

Directly with Hurley

```
$ hurl invoke <smartcontract> <smartcontract_txname> <json
args...> -C <channel> -o <organization> -u <user>
```

Example:

```
$ export DG_1='{"id": "dg_1", "name": "DG 001", "labelling":
"flammable"}'
$ hurl invoke dg dg_create "$DG_1" -C ch-global -o dgprovider -u
User1
```

Through the REST API

Two possibilities: using a browser as Firefox (with a plugin or not), or using a command line tool like Curl.

```
# apt install curl

$ curl -i -X GET http://<ip>:<port>/<smartcontract>/<txname>

$ curl -i -X POST -H 'Content-Type: application/json' -d
'<arguments>' http://<ip>:<port>/<smartcontract>/<txname>
```

Examples:

```
$ curl -i -X POST -H 'Content-Type: application/json' -'{"dg":
{"id": "dg_1", "name": "DG 001", "labelling": "flammable"}}'
http://IP:PORT/dg/create

$ curl -i -X GET http://IP:PORT/dg/get_all
```

2.6. NodeRED

Start the tool (inside Docker)

```
$ docker run -it -d -p 2727:1880 --name mynodered nodered/node-
red:latest
```

Access it with a Browser

https://localhost:2727/

Imports the flows

File (not provided): **'NodeRED--all-flows.json'**.

*Note: This file do the same over NodeRED than the scenario define into the bash script **'test-end2end.sh'**. It just adds some other collected metrics, and the IoT data, which are not mandatorily needed to play the scenario.*

Plug IoT

We use an USB sensor to collect humidity and temperature each 5 seconds and provide the data to NodeRED. The GPS geolocation is simulated by an hardcoded scenario in the NodeRED flow.

---

3. Testing: end-to-end scenario

The script which implements the scenario is **'test-end2end.sh'**.

It :

- prints machine status ;
- defines the network by starting docker containers ;
- installs the chaincodes ;
- initialize the chaincodes ;
- collects the blockchain blocks before running the scenario ;
- may run the scenario thanks to Hurley ;
- may collect the blockchain blocks after running the scenario ;
- prints machine status.

It takes around 50 minutes to finished, and uses 10 Gb RAM at initializing which stabilizes itself around 6Gb by the end.

Run it

```
$ export FILEPATH="end2end--$(date -Idate).log"
$ ./test-end2end.sh | tee $FILEPATH
```

Refer to that script in order to get deeper understanding and examples of usage.

Misc

Get the blocks count on the HLF blockchain

You must choose to which peer you will ask. This will return an integer number.

```
$ export PEER="peer0.dgprovider.hurley.lab"
$ export PEER="peer0.dgtransporter.hurley.lab"

$ export ORDERER="orderer.hurley.lab"
$ export CHANNEL="ch-global"
$ export DOCKER_EXEC="docker exec -it $PEER"
$ export DOCKER_FETCH_CHANNEL_INFO="$DOCKER_EXEC peer channel
getinfo -c $CHANNEL -o $ORDERER"
```

```
$ $DOCKER_FETCH_CHANNEL_INFO | grep "Blockchain info:" | cut -d '
' -f 3 | jq .height
```

Crawl the blocks of the HLF blockchain

We assume that the previous command states us that the number of blocks is 48 after the initialization of the scenario. Now we can do the end-to-end scenario. Then, we want to collect results whose the blockchain weight with the script **'crawl-blockchain-weight.sh'**. So, we can provides *N=0* for crawling everything, or provides *N=48* to skip the previous blocks used at initialization in order to exclude them from the results.

```
$ export N=48

$ bash crawl-blockchain-weight.sh peer0.dgprovider.hurley.lab
dgprovider ch-global orderer.hurley.lab $N out--bc-weight--
peer0.dgprovider.csv

$ bash crawl-blockchain-weight.sh peer0.dgtransporter.hurley.lab
dgtransporter ch-global orderer.hurley.lab $N out--bc-weight--
peer0.dgtransporter.csv
```

Get back the NodeRED results from its container

```
$ docker cp mynodered:/usr/src/node-red/out-tx-sizes.csv ~/
bc_and_iot/out-stats/out--nodered--tx-sizes.csv

$ docker cp mynodered:/usr/src/node-red/out-tx-times.csv ~/
bc_and_iot/out-stats/out--nodered--tx-times.csv
```

---

4. Results

Results are in the folder: **'out-stats/**.

4.1. Assumptions and settings

Machine

```
>>> Distrib:
Linux dg_sec_248 4.15.0-72-generic #81-Ubuntu SMP Tue Nov 26
12:20:02 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux

Distributor ID: Ubuntu
Description:    Ubuntu 18.04.3 LTS
Release:        18.04
Codename:       bionic

>>> Disk:
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2        74G   17G   54G  24% /
```

```
>>> RAM:
              total        used        free      shared  buff/
cache   available
Mem:            15G        400M         12G        948K
2.7G         14G
Swap:          2.0G         35M        2.0G
```

Hyperledger Fabric

Orderer *setting from:* ***'configtx.yaml'***

```
Orderer: &OrdererDefaults
  OrdererType: solo

  Addresses:
    - orderer.hurley.lab:7050

  BatchTimeout: 60s

  BatchSize:
    MaxMessageCount: 10
    AbsoluteMaxBytes: 99 MB
    PreferredMaxBytes: 1 MB
```

4.2. Blockchain weight

Refer to: **'out-stats/out--bc-weight.csv'**.

4.3. Transactions processing time

Refer to: **'out-stats/out--tx-times.csv'**.

4.4. Transactions payload size

Refer to: **'out-stats/out--tx-sizes.csv'**.

4.5. Synthesis

Refer to: **'out-stats/results.ods'**.
