Productchain Thesis Handover Document

By Olena Nesterenko (z5060085)

Preface

The codebase is located here:
https://github.com/OlenaN/THESIS

As much of the start up procedure has been automated as possible. The automation scripts have been commented for more clarity. The command *nohup* is used throughout to ensure that scripts do not terminate upon the SSH connection to the AWS server being closed.

Shard Setup

The following setup was created by modifying the already existing Fabric network implementation found here:
https://github.com/mahoney1/fabric-samples

Specifically, the *first-network* code was modified from a 2 peer 2 organization setup to a 2 peer 1 organization setup. The *byfn.sh* script for this original network was kept as the startup script but changed to work for the Productchain network. The *first-network* folder is found here:
https://github.com/mahoney1/fabric-samples/tree/release-1.2/first-network

Setting up the connection between the underlying Fabric network and the Composer business definition was done according to the following tutorial (however, also changed to work with 1 organization instead of 2):
https://hyperledger.github.io/composer/latest/tutorials/deploy-to-fabric-multi-org

AWS

The first step is to initialize 2 EC2 instances on AWS. For this thesis, the following configuration details were used for the instance:

1. **EC2 type:** t2.medium (Fabric will not run successfully with less than 4 GB RAM, but any EC2 instance with more that 4GB RAM will work)
2. **EC2 disk size:** 30 GB (Productchain setup requires a minimum of 8 GB of disk space)
3. **Operating System:** Ubuntu Bionic 18.04 (Arbitrary choice)
4. **Security groups:** Inbound SSH (Port 22), REST server (Port 3000), Composer Playground (Port 8080)

Once the EC2 instance is initialized, connect to it via SSH and install all the relevant prerequisites and clone the repo:

1. *ssh -i "PEM_FILE.pem" ubuntu@EC2_INSTANCE_NAME.compute.amazonaws.com*
2. *curl -O https://hyperledger.github.io/composer/v0.19/prereqs-ubuntu.sh*
3. *chmod u+x prereqs-ubuntu.sh*
4. *./prereqs-ubuntu.sh*
5. *curl -sSL http://bit.ly/2ysbOFE | bash -s 1.2.1 1.2.1 0.4.10*
6. *npm install -g composer-cli@0.20 composer-rest-server@0.20 generator-hyperledger-composer@0.20 composer-playground*
7. *git clone https://github.com/OlenaN/THESIS.git*
8. *Might be required: npm audit fix*

Hyperledger Fabric

Once all the prerequisites are installed, it is necessary to first start up the 2 peer 1 organization Fabric network. All the relevant files for the Fabric configuration are found in the directory: *THESIS/fabric-samples/first-network/*

To initialize the underlying Fabric network:

1. *cd fTHESIS/abric-samples/first-network*
2. *./byfn.sh generate*
3. *./byfn.sh -m up -s couchdb -a {-o kafka}*

The -o option only needs to be specified if using the Kafka orderer (default orderer is Solo)

Hyperledger Composer

Once the Fabric network has been brought up, it is necessary to install the Composer business definition (the chaincode and access control). The Composer business network is defined in the following directory:
THESIS/fabric-samples/first-network/productchain-network

However, the code to connect it with the Fabric network is found in the connection directory, not the *productchain-network* directory:

1. *cd connection*

2. *./startup.sh*

The startup script will first wipe all existing Composer data, then compile the current version of the business network into a .bna file, install it onto the peers, start the network, import the admin network card and ping the network to make sure it is running.

If the network is already running and the chaincode changes, it is not necessary to bring down the whole network and recompile. Use the *update.sh* script in the same directory after updating the version number in the *package.json* file inside the *productchain-network* directory.

To bring up the Composer REST server and Playground headlessly, use the *nohup* command.

1. *nohup composer−rest-server −c ADMIN_CARD −n never −u true −w true > logs/rest.log*

2. *nohup composer−playground > logs/play.log*

To bring down the network and shard successfully:

1. *./byfn.sh −m down*

2. *./delete_all_docker_containers*

Kill headless nohup processes with:

1. *ps -ef | grep composer-playground*

2. *Kill PLAYGROUND_PROCESS_PID*

3. *ps -ef | grep composer-rest-server*

4. *Kill REST_SERVER_PROCESS_PID*

Top Level App Setup

The top level app is built using flask.

AWS

For the top level app, the first step is also to initialize an EC2 instance on AWS. For this thesis, the following configuration details were used for the top level instance:

5. **EC2 type:** t2.micro (Top level app does not have any major requirements for RAM)
6. **EC2 disk size:** 30 GB (no major requirements for disk space - perhaps minimum 2GB)
7. **Operating System:** Ubuntu Bionic 18.04 (Arbitrary choice)
8. **Security groups:** Inbound SSH (Port 22), main server (Port 5000)

Once the EC2 instance is initialized, connect to it via SSH and install all the relevant prerequisites and clone the repo:

9. *ssh -i "PEM_FILE.pem" ubuntu@EC2_INSTANCE_NAME.compute.amazonaws.com*
10. *curl -O https://hyperledger.github.io/composer/v0.19/prereqs-ubuntu.sh*
11. *chmod u+x prereqs-ubuntu.sh*
12. *./prereqs-ubuntu.sh*
13. *curl -sSL http://bit.ly/2ysbOFE | bash -s 1.2.1 1.2.1 0.4.10*
14. *npm install -g composer-cli@0.20 composer-rest-server@0.20 generator-hyperledger-composer@0.20 composer-playground*
15. *git clone https://github.com/OlenaN/THESIS.git*
16. *Might be required: npm audit fix*

Code changes

Currently the URLs of the two shard AWS servers are hardcoded in the top level code in the file: *THESIS/top-level/utils.py*
These must be changed to the new server URLs before continuing.

Running the top level app

The top level app will immediately send a notification (telling each shard what type of shard it is -- Rural or Urban) to the two shard servers upon the homepage of the top level app being loaded.

To start up the top level app headlessly using *nohup*:

1. *cd THESIS/top-level*

2. *nohup python2 main.py > logs/main.log*

Kill headless nohup processes with:

5. *ps -ef | grep main.py*

6. *Kill MAIN_PROCESS_PID*

Structure of the app

1. **Static:** folder containing pictures and css
2. **Templates:** the html of each page
3. **Config.py:** handles setup and config
4. **Forms.py:** defines the structure of each pages form
5. **Main.py:** defined behaviour of each page
6. **Utils.py:** Behaviour of each form. Most of the shard interfacing code is here.

Sensor Setup

This tutorial was followed:
http://www.circuitbasics.com/raspberry-pi-ds18b20-temperature-sensor-tutorial/

All code exists in one file:
THESIS/Sensor/readtemp.py

To connect to the Pi you will need as screen first to initialize a wifi connection to your phones hotspot, but once you have that you can connect easily without a screen using:

1. Connect to phone hotspot via computer
2. Run *Ifconfig* on command line to find own 192 IP
3. nmap -sP THE_192_IP.0/24
4. *(ie. nmap -sP 192.XXX.XX.0/24)*
5. Nmap should return the IP address of the Pi connected to the same network
6. Ssh pi@{raspberry Pi IP address}
7. Password: raspberry (default password)