UTM

UNIVERSITI TEKNOLOGI MALAYSIA

Managing IOT Data on Hyperledger Blockchain

Name: Faezeh Shahidi, shahidi@graduate.utm.my

Submitted to: Prof. Dr. Salwani

Date: 11/12/2020

1-Necessary Hardware:

- Two raspberry Pi devices (3 or above with Wi-Fi connection)
- Two mini SD cards (16 GB each)
- One Mouse (with USB)
- One Keyboard (with USB)
- One 3.5 Lcd display
- 5 pcs 1 Pin Male to Female Jumper Cable 20 cm Long
- One Breadboard
- One DHT11 sensor
- One or two ethernet cable
- Wi-Fi modem that enables us to connect to the internet through VNC by IP Address
- A laptop with at least a core i5 CPU, 16 GB RAM, and at a hard drive with at least 50 GB empty space

2-Necessary software:

a) IOT:

- Debian OS for raspberry pi
- Mu-editor for running the python coding
- VNC viewer for controlling the raspberry pi through laptop

b) Blockchain:

- Windows / mac
- If the OS is windows, then we need virtual machine installation.
- VMware workstation or oracle VM virtual box (oracle is free but sometimes it has problems especially in its new version, that's what VMware is recommended)
- Ubuntu 16.4
- VSCode for running the blockchain Hyper ledger fabric network

3-Pre-requisites

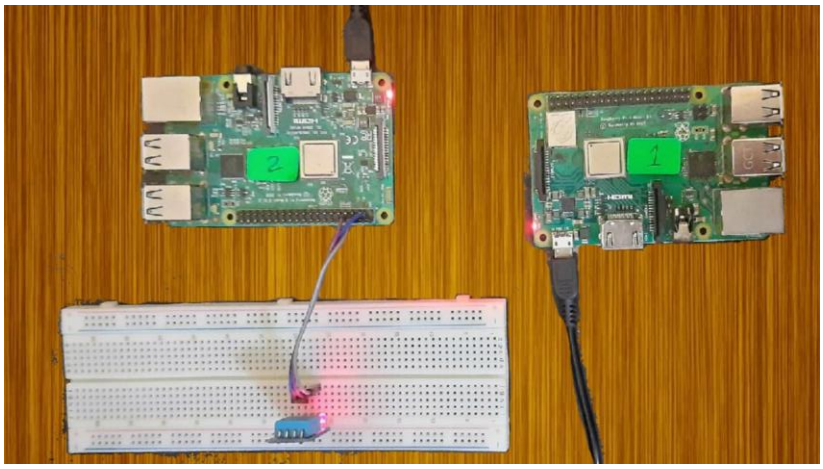
- Node
- Npm
- Docker
- Docker composes
- Hyper-ledger composer command line
- Python
- Go Lang

NB: To run the Hyperledger fabric project needs a lot of packages that have to be installed through terminal throughout the workshop.

IOT

In this part there are two stations, including station 1 and 2. Station 2, will receive the data from the Arduino DHT11 Temperature and Humidity Sensor.
As it is shown in the following picture, the connection from sensor to the raspberry pi is made by a Male to Female Jumper Cable as follows:

- Vcc → 1
- data → 8 (which is gpio =14)
- ground → 6



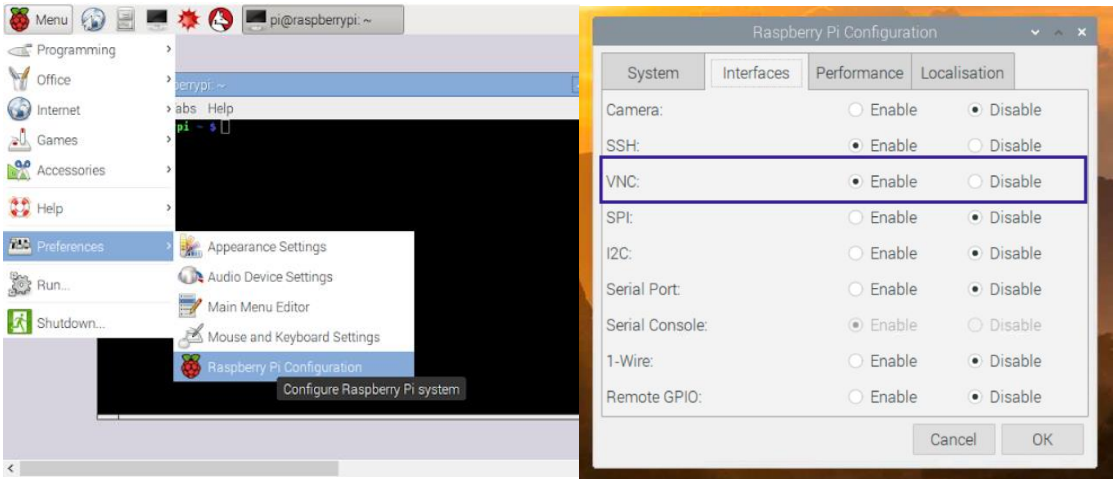
The OS that is installed in each raspberry pi is Debian. It is necessary to be connected to the ethernet cable while installing the Debian OS.
In the following, the environment settings will be explained for station 1 and 2.

Station1:

IP address: 192.168.0.107
At first, after installation of the Debian, we have the raspberry pi with Debian OS that is connected to the internet by Ethernet cable. Once connected to the Wi-Fi, by **ifconfig** command we can have an access to the IP address of the device through Wi-Fi. We can connect to the raspberry pi device through VNC viewer through the computer by the IP address that is written in the Wi-Fi part while running ifconfig.

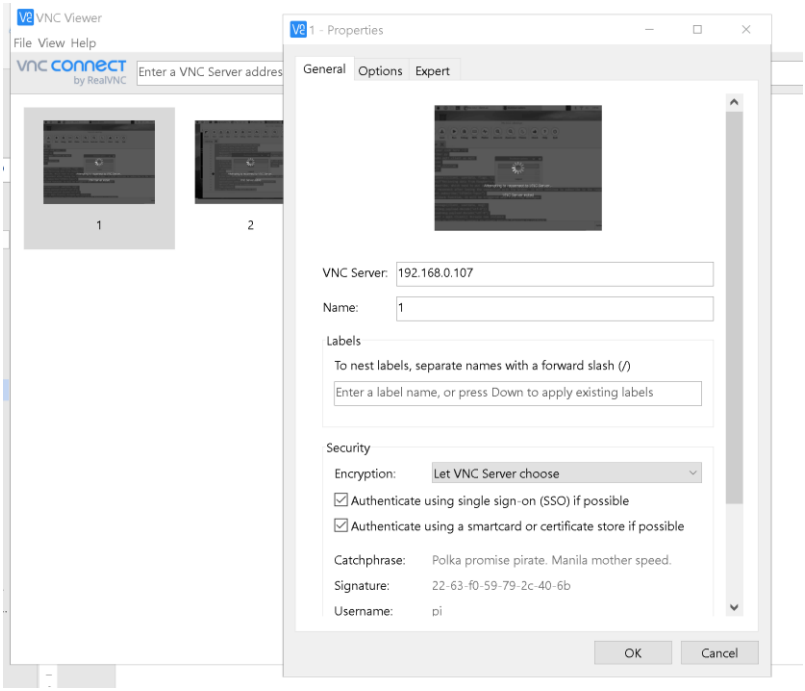
```
Ifconfig
```

Then We need to enable the VNC on the raspberry pi as well.



Once trying to connect to the raspberry pi, we need to have the following info:

- Ip address: (check through the ifconfig)
- Username: pi
- Password:raspberry



After connection to the device through the laptop we can set up the environment as follows:

Set up the environment of the Station1:

Then we need to update the device and reboot the system in the terminal.

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

How to Install and Secure the Mosquitto MQTT Messaging Broker on Raspberry pi, Debian OS

```
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable mosquitto.service
mosquitto -v
```

By the following command we can see if we have not installed pip3 and python3,

```
pip --version
python
```

If we don't have python3 and pip3, we need to install

```
sudo apt-get install python3-dev python3-pip
sudo python3 -m pip install --upgrade pip setuptools wheel
```

The Paho Python Client installation

Paho provides a client class with support for both MQTT v3.1 and v3.1.1 on Python 2.7 or 3.x. It also provides some helper functions to make publishing one off messages to an MQTT server very straightforward.

```
pip3 install paho-mqtt
```

Client1.py

The following file is the client1.py that is run in the station1 where we receive the data from station2 and pass the data to a node js (invoke.js) file to be inserted to the blockchain. These coding can be run through mu-editor or we can run it as follows in the terminal

```
python client1.py
```

```python
import paho.mqtt.client as mqtt
import json
import time,datetime
payload=""
def on_connect(client, userdata, flags, rc):
    print(f"Recieving data from Raspberry-pi client...")
    # subscribe, which need to put into on_connect
    # if reconnect after losing the connection with the broker, it will continue to subscribe to the raspberry/topic topic
    client.subscribe("IOTDATA-Topic")
# the callback function, it will be triggered when receiving messages
def on_message(client, userdata, msg):
    print(msg.payload.decode("utf-8"))
    payload=msg.payload.decode("utf-8")
    client1 = mqtt.Client() #create new instance
    client1.on_connect=on_connect_to_client #attach function to callback
    client1.connect(broker_address_client, keepalive=60, bind_address="") #connect to broker
    print('sending data to the Client...')
    if payload is not None:
      print(payload)
    else:
      print('Failed to get reading. Try again!')
    # the four parameters are topic, sending content, QoS and whether retaining the message respectively
    client1.publish('IOTDATA-Topic',payload=payload, qos=0, retain=False)
    time.sleep(5)
def on_connect_to_client(client, userdata, flags, rc):
    if rc == 0:
        print("Connected success")
    else:
        print("Connected fail with code")
broker_address="192.168.0.107"
broker_address_client="192.168.0.108"
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
# set the will message, when the Raspberry Pi is powered off, or the network is interrupted abnormally, it will send the will message to other clients
client.will_set('raspberry/status', b'{"status": "Off"}')
# create connection, the three parameters are broker address, broker port number, and keep-alive time respectively
client.connect(broker_address, keepalive=60)
# set the network loop blocking, it will not actively end the program before calling disconnect() or the program crash
client.loop_forever()

```

Station2:

IP address: 192.168.0.106

Set up the environment of the Station2:

The following commands are the same as station1:

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable mosquitto.service
mosquitto -v
pip --version
python
sudo apt-get install python3-dev python3-pip
sudo python3 -m pip install --upgrade pip setuptools wheel
pip3 install paho-mqtt
```

Adafruit installation

We need to do the same for the client2 as client1. Then we need to install adafruit for working with adafruit liberty and working from sensors.

```
pip3 install adafruit-circuitpython-dht
sudo apt-get install libgpiod2
```

Client2.py

The following file is the client2.py that is run in the station2 where we read the data from the sensor and pass them to station1. These coding can be run through mu-editor or we can run it as follows in the terminal.

```
python client2.py
```

```python
import paho.mqtt.client as mqtt #import the client1
import time,datetime
import json
import Adafruit_DHT

# Set sensor type : Options are DHT11,DHT22 or AM2302
sensor=Adafruit_DHT.DHT11
# Set GPIO sensor is connected to
gpio=14

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected success")
    else:
        print(f"Connected fail with code {rc}")

broker_address="192.168.0.107"

client = mqtt.Client() #create new instance
client.on_connect=on_connect #attach function to callback
client.connect(broker_address, keepalive=60, bind_address="") #connect to broker
print('sending data to the Broker...')
# send a message to the raspberry/topic every 1 second, 5 times in a row
for i in range(1):
    # Use read_retry method. This will retry up to 15 times to
    # get a sensor reading (waiting 2 seconds between each retry).
    humidity, temperature = Adafruit_DHT.read_retry(sensor, gpio)
    c_date_time=datetime.datetime.now()
    c_time=c_date_time.strftime("%X")
    c_date=c_date_time.strftime("%x")
    #payload='Timestamp= {0}-{1} Temp={2:0.1f}*C  Humidity={3:0.1f}%'.format(c_date, c_time,temperature, humidity)
    payload='{0}-{1},{2:0.1f}*C,{3:0.1f}%'.format(c_date, c_time,temperature, humidity)

    # Reading the DHT11 is very sensitive to timings and occasionally
    # the Pi might fail to get a valid reading. So check if readings are valid.
    if humidity is not None and temperature is not None:
      print(payload)
    else:
      print('Failed to get reading. Try again!')
    # the four parameters are topic, sending content, QoS and whether retaining the message respectively
    client.publish('IOTDATA-Topic', payload=json.dumps(payload), qos=0, retain=False)
    time.sleep(5)
client.loop_forever() #start the loop
```

Blockchain: Hyper ledger fabric

In this project, we have applied Ubuntu (version 16.4) as our operation system. Also, Visual Studio Code which is a code editor used for building and debugging our project.

1)Technical pre-requisites:

- Development environment:

downloading the script for Hyperledger composer repository that let us to install all the pre requisites. by the following script we can **install Node, nmp, Docker, Docker compose, and python** as shown in the pic.

```
sudo curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh
sudo chmod +x prereqs-ubuntu.sh
./prereqs-ubuntu.sh
```

** Key in `q` if need it stops
Installing **Hyperledger composer command line** interface.

```
npm i -g composer-cli@0.19.4
```

Installing **composer rest server** that is explore the rest for api

```
npm i -g composer-rest-server@0.19.4
```

Installing **generator** to generate business network

```
npm i -g generator-hyperledger-composer@0.19.4
```

To use the generator, we use **yo**

```
npm i -g yo
```

Installing GOLANG

```
wget https://storage.googleapis.com/golang/go1.9.2.linux-amd64.tar.gz
sudo tar -C /usr/local -xzf go1.9.2.linux-amd64.tar.gz
```

2) Set up the network:

a) Visual studio code

now, we have done with the installation and can open the project in the **VScode**
we need to **install the visual studio code**
after installation click on the **file → autosave**
extensions on the VSCode that should be searched and installed:

- Docker
- Go
- Go-outline
- Remote-containers

b) Blockchain

we should run the following scripts in the terminal once we are in the cd iot/iot-network

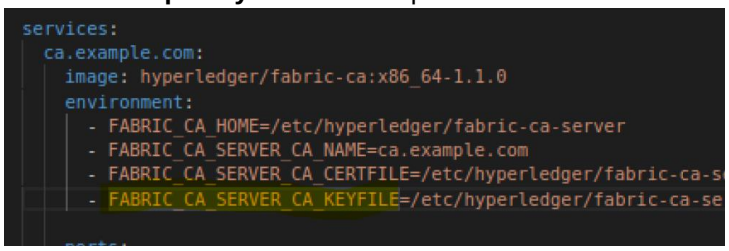
For **pulling the images of the docker** (before downloading, it's better to reboot)

```
./downloadFabric.sh
```

For generating the crypto material, genesis block, channel configuration, and anchor peer transaction

```
./generate.sh
cd crypto-config/peerOrganizations/org1.example.com/ca
ls
```

copy the _sk coding from "**crypto-config/peerOrganizations/org1.example.com/ca**" and past it in the "**docker-compose.yml"** in the **ca** part as follows:



To create the channel and Join peer0.org1.example.com to the channel.

```
./start.sh
```

```
2020-10-27 11:06:41.490 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer
connections initialized
2020-10-27 11:06:41.752 UTC [channelCmd] InitCmdFactory -> INFO 002 Endorser and orderer
connections initialized
2020-10-27 11:06:41.985 UTC [main] main -> INFO 003 Exiting.....
# # Join peer0.org1.example.com to the channel.
docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledge
r/msp/users/Admin@org1.example.com/msp" peer0.org1.example.com peer channel join -b mycha
nnel.block
2020-10-27 11:06:45.328 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer
connections initialized
2020-10-27 11:06:45.876 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted p
roposal to join channel
2020-10-27 11:06:45.876 UTC [main] main -> INFO 003 Exiting.....
```

3) Necessary commands to work with docker images and composers

To get the list of images:

```
docker images
```

To get the list of composers:

```
docker ps
```

To enter a composer (like peers):

```
docker exec -it <ID> bash
```

4) Channel:

By starting, peer0 has been already joined to the channel, named mychannel,
then we need to join peer1 to the channel as well
To join the **peer1** to the channel.

```
docker exec -it peer0.org1.example.com bash
rm /etc/hyperledger/configtx/mychannel.block
cp mychannel.block /etc/hyperledger/configtx/
exit
docker exec -it peer1.org1.example.com bash
export CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp
peer channel join -b /etc/hyperledger/configtx/mychannel.block
```

```
→ docker exec -it peer1.org1.example.com bash
root@5b2899leea0b:/opt/gopath/src/github.com/hyperledger/fabric# export CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp
root@5b2899leea0b:/opt/gopath/src/github.com/hyperledger/fabric# peer channel join -b /etc/hyperledger/configtx/mychannel.block
2020-10-27 11:47:33.316 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-10-27 11:47:33.785 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2020-10-27 11:47:33.785 UTC [main] main -> INFO 003 Exiting.....
root@5b2899leea0b:/opt/gopath/src/github.com/hyperledger/fabric# peer channel list
2020-10-27 11:47:46.214 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
Channels peers has joined:
mychannel
2020-10-27 11:47:46.233 UTC [main] main -> INFO 002 Exiting.....
root@5b2899leea0b:/opt/gopath/src/github.com/hyperledger/fabric#
```

To see the list of channels for the peer:

```
peer channel list
```

Once enter to the peer or composer, to see the environment set on the composers:

```
export
```

5) Chaincode:

Chaincodes are installed on every peer which has to endorse a given transaction. Endorsement involves execution of the chaincode with some given arguments, finding out the result and signing it (with MSP of the org that the peer is a part of). Chaincodes are installed on a per peer basis and can be instantiated on one or more channels. If peer is belonging to a channel but is not endorsing any transactions, then it doesn't need not the chaincode. It can just choose to store the ledger of the given channel (act as a committing peer).
If there is more than one endorsing peer per organization's MSP, you can sign and continue transactions smoothly. Setting up more than one peer per ORG MSP can help in crash fault tolerance as you have explained. Secondly, if there is only one peer per ORG MSP and your endorsement policy is such that you absolutely require the signature of that MSP for your transaction to go through only then the channel's transaction for that given chaincode will fail. Other chaincodes on the same channel, that does not require this signature will still continue to function fine.
We can **install the chaincode** on the peers once we are in the **cli**. From the cli, we can access to the peer number, 0 or 1, once the core peer address is set on that peer. As the defaults core peer address in the cli is set on the peer0, so by the following command we install the chaincode on the peer0 first. Then we change the core peer address to peer1 and then install the chaincode on peer1 as well.

```
docker exec -it cli bash
```

(we should change the directory to **/opt/gopath/src/github.com/hyperledger/**, in which the list of chain codes is in:

```
cd ../../..
ls
peer chaincode install -n mychain -p github.com/iot -v 1.0
export CORE_PEER_ADDRESS="peer1.org1.example.com:8051"
peer chaincode install -n mychain -p github.com/iot -v 1.0
```

```
2183ACD2D173ECB3C4068E6F
2020-10-27 12:09:53.473 UTC [chaincodeCmd] install -> DEBU 00f Installed remotely response:<status:200 payload:"OK
```

a) install the chaincode in the channel

to install the chaincode we need to install it once. For the installation we can enter to the **peer** composer, or else we can enter to the **cli**. In here the installation has been performed from **cli.**

```
docker exec -it cli bash
```

```
peer chaincode instantiate -n mychain -v 1.0 -o orderer.example.com:7050 -C mychannel -c
'{"Args":["initLedger"]}'
```

b) Update the chaincode

If we want to **change the iot.go** file and then upgrade the previous chaincode, we need **to install the chaincode with the new version (-v 1.2) then upgrade the chaincode in the channel** by the following coding:

```
peer chaincode upgrade -n mychain -v 1.1 -o orderer.example.com:7050 -C mychannel -c
'{"Args":["initLedger"]}'
```

```
root@8fdce6ad7c4e:/opt/gopath/src/github.com# peer chaincode instantiate -n mychain -v 1.0 -o orderer.example.com:7050 -C mychannel -c '{"Args":["initLedger"]}'
2020-11-12 14:13:49.668 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2020-11-12 14:13:49.668 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2020-11-12 14:13:49.684 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2020-11-12 14:13:49.685 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2020-11-12 14:13:49.685 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode disabled
2020-11-12 14:13:49.685 UTC [msp/identity] Sign -> DEBU 006 Sign: plaintext: 0AA7070A6708031A0C089D86B5FD0510...65720A000A04657363630A0476736363
2020-11-12 14:13:49.689 UTC [msp/identity] Sign -> DEBU 007 Sign: digest: 1506DAA21F4D66F9EE3DBE3AB0915ECBF6427EC17D8F55A8C01B38A8DBD63C2D
2020-11-12 14:15:27.772 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0AA7070A6708031A0C089D86B5FD0510...D8ABED6CBD2800EEEFEDF0784CBF5BCD
2020-11-12 14:15:27.772 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: CB9FFD75843658FB80B156A910D6C8996FAEA7F8C976F19D69204AB6697AEC54
2020-11-12 14:15:27.777 UTC [main] main -> INFO 00a Exiting.....
```

c) List of installed chaincode on the peers

To see the list of chaincode that are installed on each peer separately

```
export CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp
peer chaincode list --installed
```

```
root@2696ff395bf4:/opt/gopath/src/github.com/hyperledger/fabric# export CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp
root@2696ff395bf4:/opt/gopath/src/github.com/hyperledger/fabric# peer chaincode list --installed
Get installed chaincodes on peer:
Name: mychain, Version: 1.0, Path: github.com/iot, Id: 3d881e1252211681524470c462289e5ccecc3ffa655abc0dd52814e477b00e1b
2020-11-12 14:17:56.209 UTC [main] main -> INFO 001 Exiting.....
root@2696ff395bf4:/opt/gopath/src/github.com/hyperledger/fabric#
```

6) Insert the data into the ledger

we have two functions for inserting the data in the ledger and query them by the time stamp as the key. These two functions are reachable by invoke function as its first argument.

```
peer chaincode invoke -C mychannel -n mychain -c '{"Args":["createIotData","11/2/2020-
15:28:20","28","60"]}'
```

```
lVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1T\nYW4gRnJhbmNpc2NvMR8wHQYDVQQDExZwZWV
yMC5vcmcxLmV4YW1wbGUuY29tMFkw\nEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAELs41AKnm445vMesNPfIuN
yhOnrcK9gk8\nM6aUhsqrJc/08SIKKFW0YjWXT+jGpcFeP32+7uul39G8sKUj4NWYq6NNMEswDgYD\nVR0PA
QH/BAQDAgeAMAwGA1UdEwEB/wQCMAAwKwYDVR0jBCQwIoAgqJ5U2wRXn9DJ\n1r8zNVMT0RyBPDHggl3c8MV
dF5XS1EkwCgYIKoZIzj0EAwIDRwAwRAIgDL5mA+XP\nVfMrNwV7RRcQ7i1mWn7OqFIaW59CSLpRGmkCIFQGJ
jgr4ymY3mrcAY8rD9YXmbk4\nAu+lUnfq0ulOTr+x\n-----END CERTIFICATE-----\n" signature:"0
D\002 6\311Zr#!.<t\340@\372\311\t\250\215=\225\3242\350?:\006r\032\216`\206\255\253]
\002 @qx\265\356yX\306[o\311\266;\274\276\261\251\372\n+\247IzoI!&eK\020b\310" >
2020-11-02 07:28:52.737 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 063 Chainc
ode invoke successful. result: status:200
2020-11-02 07:28:52.739 UTC [main] main -> INFO 064 Exiting.....
root@3b21f95ca73d:/opt/gopath/src/github.com#
```

7) Query the data from the ledger by the timestamp

```
peer chaincode invoke -C mychannel -n mychain -c '{"Args":["queryIotData","11/2/2020-
15:28:20"]}'
```

```
uY29tMFkw\nEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAELs41AKnm445vMesNPfIuNyhOnrcK9gk8\nM6aUhsq
rJc/08SIKKFW0YjWXT+jGpcFeP32+7uul39G8sKUj4NWYq6NNMEswDgYD\nVR0PAQH/BAQDAgeAMAwGA1UdE
wEB/wQCMAAwKwYDVR0jBCQwIoAgqJ5U2wRXn9DJ\n1r8zNVMT0RyBPDHggl3c8MVdF5XS1EkwCgYIKoZIzj0
EAwIDRwAwRAIgDL5mA+XP\nVfMrNwV7RRcQ7i1mWn7OqFIaW59CSLpRGmkCIFQGJjgr4ymY3mrcAY8rD9YXm
bk4\nAu+lUnfq0ulOTr+x\n-----END CERTIFICATE-----\n" signature:"0D\002 C\200A;aP\226\
307\330fw\024z\271\257\245\r\034-\013\354G\373O\031\002@!\262\227\250/\002 \021\373k
.G\327\372Q\240,\205={\323?')\231\231\305\332x;O^\207@N\241/\033\326" >
2020-11-02 07:30:41.634 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 063 Chainc
ode invoke successful. result: status:200 payload:"{\"humidity\":\"60\",\"temperatur
e\":\"28\"}"
2020-11-02 07:30:41.639 UTC [main] main -> INFO 064 Exiting.....
root@3b21f95ca73d:/opt/gopath/src/github.com#
```

** we can invoke the data from the peers which are the in the channel. Once the chaincode is installed and instantiated on the peers and chaincode, we are able to see the composers of the chaincode for each peer.

c) List of installed chaincode on the channel

To see if we have instantiated any chaincode in the channel. (The channel can be empty of any chaincode)

```
peer chaincode list --instantiated -C mychannel
```

to see the list of installed chaincode

```
peer chaincode list --installed
```

delete the chaincode:

```
rm -f mychain.1.0
```

To access the list of chaincodes that can be intsalled in the peer

```
cd ..
cd /var/hyperledger/production/
cd chaincodes/
ls
```

8) Delete commands:

To remove all the images and composers:

```
docker rm -f $(docker ps -a -q)
docker images -a -q | xargs docker rmi
```

To remove one specific image or composer

```
docker rm <The id of the container>
docker rmi <The id of the image>
```

9) Some useful commands

To see the logs of one specific contatiner:

```
docker logs peer0.org1.example.com
```

we can set the core_peer in cli to the peer0 or 1 and change the port to the related peer and from the export can observe the changes

```
export CORE_PEER_ADDRESS="peer1.org1.example.com:8051"
```

To define the policy : -p flag in here for the instantiation can define the policy

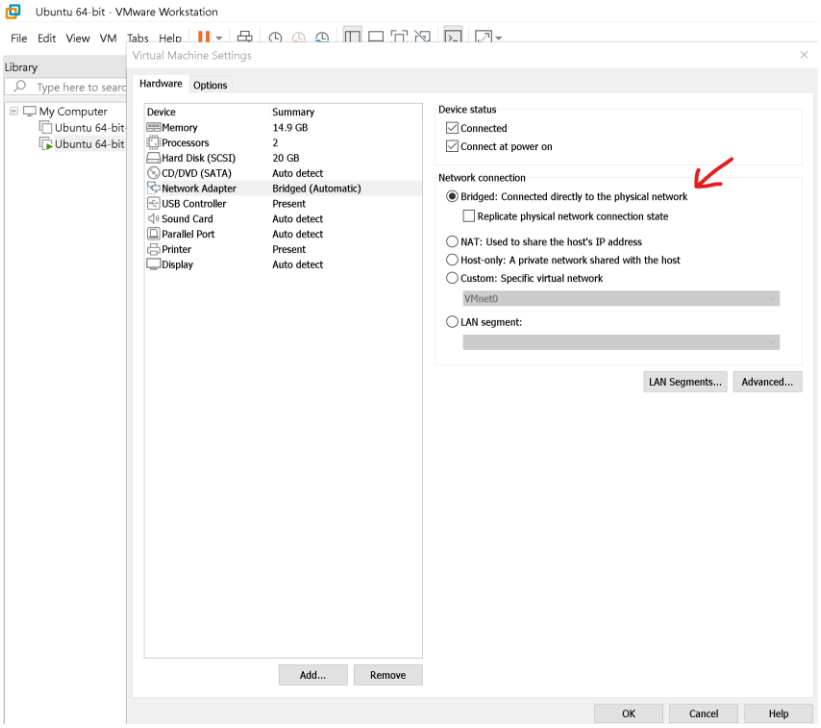
IOT sample Application:

We need to set the environment for the client. So that we are able to receive the data from raspberry pi station1.

Set up the environment in ubuntu for connecting the raspberry pi to the ubuntu

How to Install Mosquitto MQTT Broker/Server on Ubuntu 16.04:

In the network adapter in the VM we should select the bridge as shown in the picture



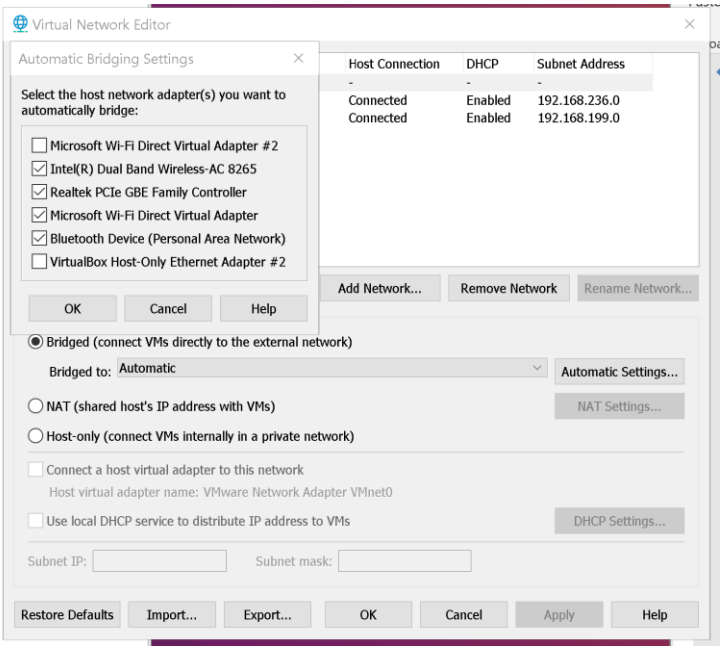
Then in the VM select:

```
Edit> Virtual Network Editor > change settings
```

Then select the VMnet0 on the top and click on the

```
> Automatic Settings
```

NB: the following picture will appear and only one ***Adaptor*** has to be selected and the rest have to be unchecked.



Install python3:

```
sudo apt-get update
sudo apt-get install python3.6
install pip and pip3:
sudo apt-get install python-pip
sudo apt-get install python3-pip
```

To make python3 use the new installed python 3.6 instead of the default 3.5 release, run following 2 commands:

```
Python3 -V
```

```
➜ python3 -V
Python 3.5.2
```

```
sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.5 1
sudo update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.6 2
sudo update-alternatives --config python3
```

key in 2

```
python3 -V
```

```
➜ python3 -V
Python 3.6.12
```

If we have got Problem in ubuntu 16.04 after updating python 3.5 to 3.6.

```
sudo apt-get --reinstall install python3-minimal
```

We need to install the paho in the right destination for python 2 and 3 in the ubuntu:

```
cd /usr/lib/python2.7/dist-packages
```

```
sudo pip install paho-mqtt -t ./
```

```
→ pip3 -V
pip 8.1.1 from /usr/lib/python3/dist-packages (python 3.6)
```

```
cd /usr/lib/python3/dist-packages
sudo pip3 install paho-mqtt -t ./
```

Then we should run the following command to install the mosquitto

```
sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa
sudo apt-get update
sudo apt-get install mosquitto
sudo apt-get install mosquitto-clients
sudo apt clean
mosquitto -v
```

once the mosquitto is installed correctly we can observe the IP address of the ubuntu that is in the virtual box and this IP address can be accessible by other devices that are connecter to the modem. The ip address of the ubuntu OS is 192.168.0.108 that is accessible by the port, 1883. Once we have the address, we can set it in the broker address in the client to subscribe the message.

```
netstat -at
```

```
→ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 ubuntu:domain           *:*                     LISTEN
tcp        0      0 localhost:ipp           *:*                     LISTEN
tcp        0      0 *:telnet                *:*                     LISTEN
tcp        0      0 *:1883                  *:*                     LISTEN
tcp        0      0 localhost:38203         *:*                     LISTEN
tcp        0      0 localhost:34601         localhost:59090         ESTABLISHED
tcp        0      0 localhost:59090         localhost:34601         ESTABLISHED
tcp        0      0 localhost:42826         localhost:41861         ESTABLISHED
tcp        0      0 192.168.0.108:38933     192.168.0.108:1883      ESTABLISHED
tcp        0      0 192.168.0.108:1883      192.168.0.108:38933     ESTABLISHED
tcp        0      0 localhost:36095         localhost:35078         ESTABLISHED
tcp        0      0 localhost:41861         localhost:42826         ESTABLISHED
tcp        0      0 localhost:46793         localhost:50938         ESTABLISHED
tcp        0      0 192.168.0.108:1883      192.168.0.108:59705     ESTABLISHED
tcp        0      0 localhost:45841         localhost:57854         ESTABLISHED
tcp        1      0 192.168.199.132:35643   192.168.199.132:1883    CLOSE_WAIT
tcp        0      0 localhost:33179         localhost:41064         ESTABLISHED
tcp        0      0 localhost:35078         localhost:36095         ESTABLISHED
tcp        0      0 localhost:41064         localhost:33179         ESTABLISHED
tcp        0      1 192.168.0.108:49687     192.168.199.132:1883    SYN_SENT
tcp        0      0 localhost:55058         localhost:32775         ESTABLISHED
tcp        0      0 localhost:32775         localhost:55058         ESTABLISHED
tcp        0      0 localhost:50938         localhost:46793         ESTABLISHED
tcp        0      0 localhost:57854         localhost:45841         ESTABLISHED
tcp        0      0 192.168.0.108:59705     192.168.0.108:1883      ESTABLISHED
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN
tcp6       0      0 [::]:1883               [::]:*                  LISTEN
```

The IP address can be reached by the following commands as well

```
route -n
netstat -nt
```

```
➜ netstat -nt
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:34601         127.0.0.1:59090         ESTABLISHED
tcp        0      0 127.0.0.1:59090         127.0.0.1:34601         ESTABLISHED
tcp        0      0 127.0.0.1:42826         127.0.0.1:41861         ESTABLISHED
tcp        0      1 192.168.0.108:60869     192.168.199.132:1883    SYN_SENT
tcp        0      0 127.0.0.1:41861         127.0.0.1:42826         ESTABLISHED
tcp        0      1 192.168.0.108:51119     192.168.199.132:1883    SYN_SENT
tcp        0      0 127.0.0.1:46793         127.0.0.1:50938         ESTABLISHED
tcp        0      0 127.0.0.1:45841         127.0.0.1:57854         ESTABLISHED
tcp        0      1 192.168.0.108:58941     192.168.199.132:1883    SYN_SENT
tcp        1      0 192.168.199.132:35643   192.168.199.132:1883    CLOSE_WAIT
tcp        0      0 127.0.0.1:33179         127.0.0.1:41064         ESTABLISHED
tcp        0      0 127.0.0.1:41064         127.0.0.1:33179         ESTABLISHED
tcp        0      0 127.0.0.1:50938         127.0.0.1:46793         ESTABLISHED
tcp        0      0 127.0.0.1:57854         127.0.0.1:45841         ESTABLISHED

~
➜ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.0.1     0.0.0.0         UG    100    0        0 ens33
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 ens33
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
192.168.0.0     0.0.0.0         255.255.255.0   U     100    0        0 ens33

~
➜
```

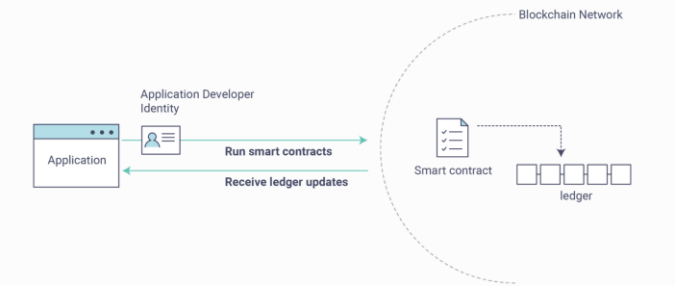
Extra info to be assured that our VM can be reachable is to set the telnet:

```
sudo apt-get install telnetd
sudo ufw allow 23/tcp
telnet 192.168.0.108
```

Installing mqtt for node:

```
npm install mqtt –save
npm install mqtt -g
```

1) bring External data (IoT) data into Blockchain



we should be in iot- api direction and run the following coding:

```
cd iot/iot-api
```

We should install the packages to work with APP to run the smart contract and receive ledger updates

```
npm install
npm install grpc
```

to install the crypto keys for admin and user1

```
node enrollAdmin.js
node registerUser.js
npm install mqtt –save
npm install mqtt -g
```

if receiving error once trying to publish the data we should run:

```
mosquitto -v
```

once we run the invoke.js, the client is waiting for the data to be received from station 1. When station 1 receives the data from station 2, it will pass the data to the invoke.js, and this node will insert the data into the ledger by using the user1.

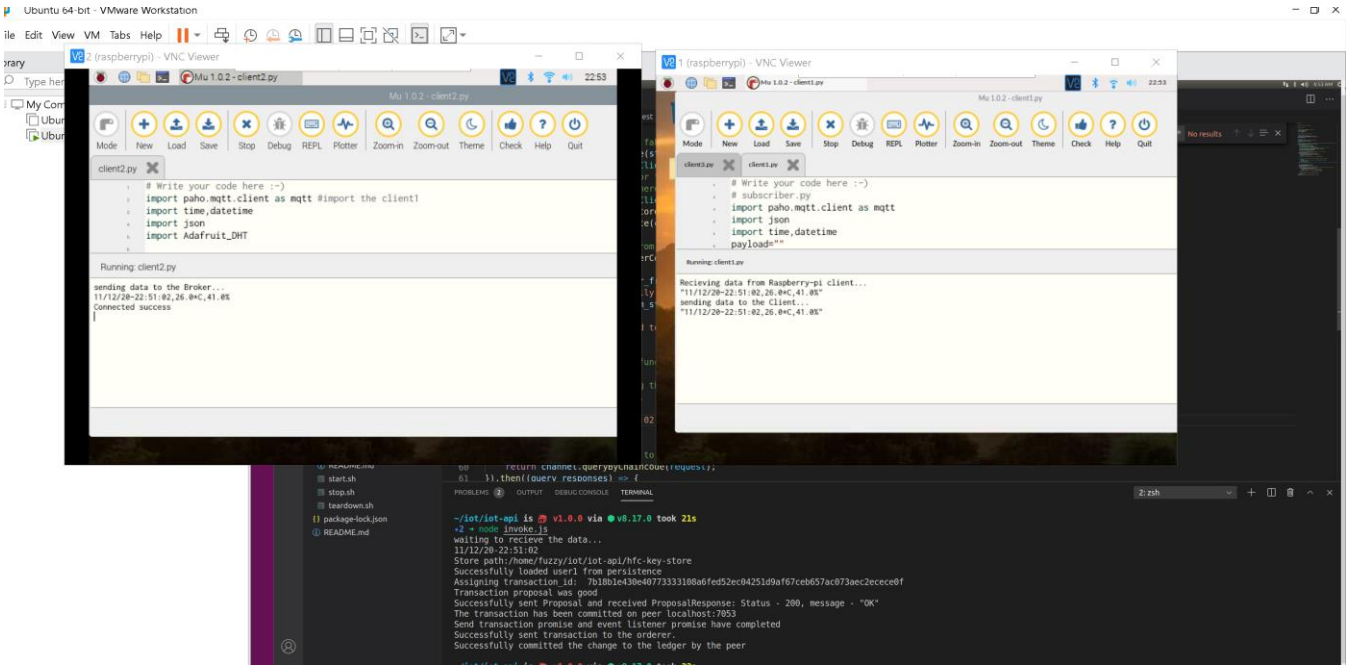
```
node invoke.js
python client1.py
python client2.py
```

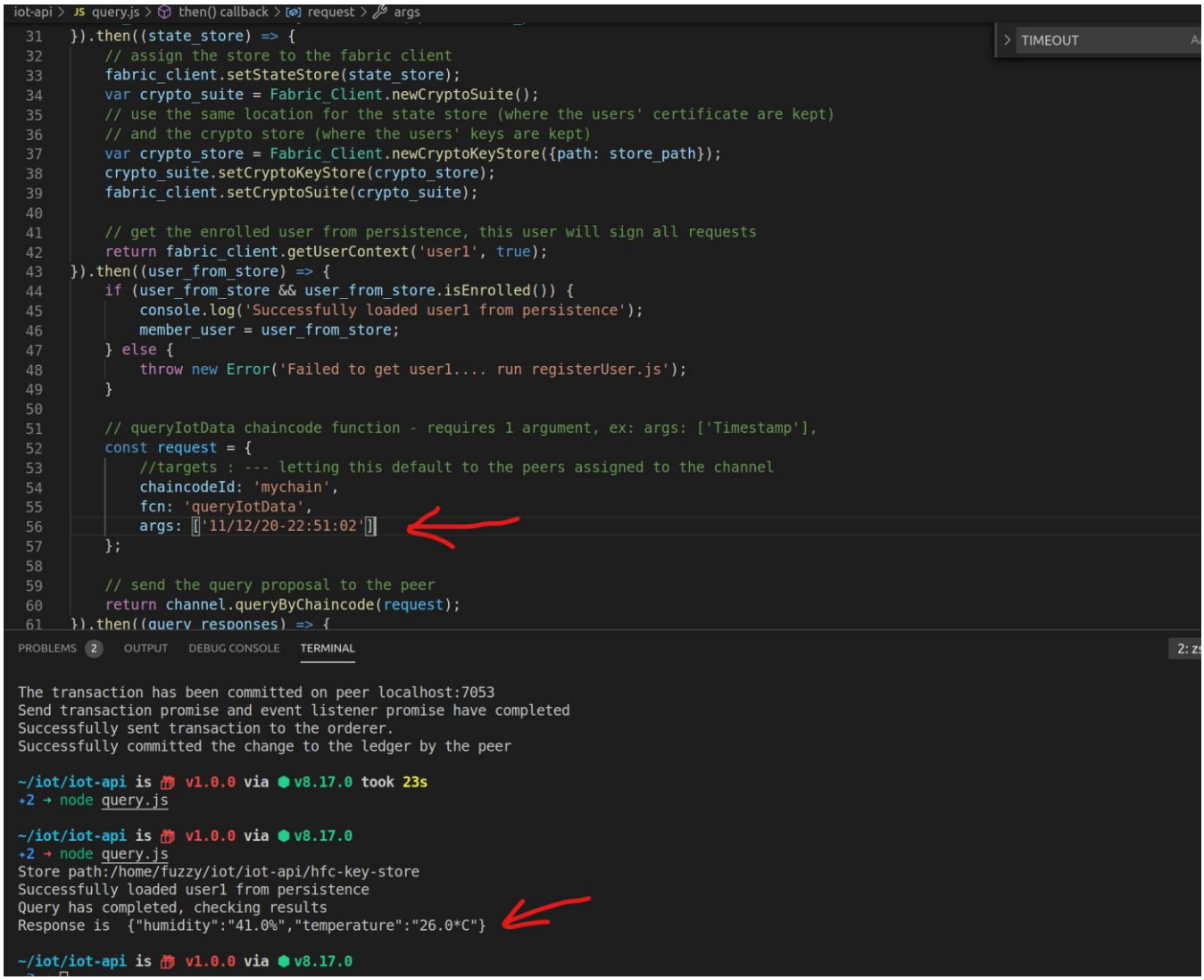
In order to query one data by time stamp, we can use the query.js and select the data.

```
node query.js
```

```
// queryIotData chaincode function - requires 1 argument, ex: args: ['Timestamp'],
const request = {
    //targets : --- letting this default to the peers assigned to the channel
    chaincodeId: 'mychain',
    fcn: 'queryIotData',
    args: ['11/12/20-22:51:02']
};
```

The following picture shows the overall picture of the three stations.




iot-api > JS query.js > then() callback > request > args
}).then((state_store) => {
// assign the store to the fabric client
fabric_client.setStateStore(state_store);
var crypto_suite = Fabric_Client.newCryptoSuite();
// use the same location for the state store (where the users' certificate are kept)
// and the crypto store (where the users' keys are kept)
var crypto_store = Fabric_Client.newCryptoKeyStore({path: store_path});
crypto_suite.setCryptoKeyStore(crypto_store);
fabric_client.setCryptoSuite(crypto_suite);
// get the enrolled user from persistence, this user will sign all requests
return fabric_client.getUserContext('user1', true);
}).then((user_from_store) => {
if (user_from_store && user_from_store.isEnrolled()) {
console.log('Successfully loaded user1 from persistence');
member_user = user_from_store;
} else {
throw new Error('Failed to get user1.... run registerUser.js');
}
// queryIotData chaincode function - requires 1 argument, ex: args: ['Timestamp'],
const request = {
//targets : --- letting this default to the peers assigned to the channel
chaincodeId: 'mychain',
fcn: 'queryIotData',
args: ['11/12/20-22:51:02']
};
// send the query proposal to the peer
return channel.queryByChaincode(request);
TIMEOUT
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
The transaction has been committed on peer localhost:7053
Send transaction promise and event listener promise have completed
Successfully sent transaction to the orderer.
Successfully committed the change to the ledger by the peer
~/iot/iot-api is v1.0.0 via v8.17.0 took 23s
+2 → node query.js
~/iot/iot-api is v1.0.0 via v8.17.0
+2 → node query.js
Store path:/home/fuzzy/iot/iot-api/hfc-key-store
Successfully loaded user1 from persistence
Query has completed, checking results
Response is {"humidity":"41.0%","temperature":"26.0*C"}
~/iot/iot-api is v1.0.0 via v8.17.0