Practical-1

Aim:

Write a code using any programming language to create Blockchain consists of at least five blocks. Verify the block before storing into Blockchain. Test using POSTMAN.

Code:-

```
import sys
import hashlib
import json

from time import time
from uuid import uuid4

from flask import Flask
from flask.globals import request
from flask.json import jsonify
from numpy import block

import requests
from urllib.parse import urlparse

class Blockchain(object):
   difficulty_target = "0000"

   def hash_block(self, block):
      block_encoded = json.dumps(block, sort_keys=True).encode()
      return hashlib.sha256(block_encoded).hexdigest()

   def __init__(self):
      self.nodes = set()

      self.chain = []

      self.current_transactions = []

      genesis_hash = self.hash_block("genesis_block")

      self.append_block (
         hash_of_previous_block = genesis_hash,
         nonce = self.proof_of_work(0, genesis_hash, [])
```

```
    )

def add_node(self, address):
    parse_url = urlparse(address)
    self.nodes.add(parse_url.netloc)
    print(parse_url.netloc)

def valid_chain(self, chain):
    last_block = chain[0]
    current_index = 1

    while current_index < len(chain):
        block = chain[current_index]


        if block['hash_of_previous_block'] != self.hash_block(last_block):
            return False

        if not self.valid_proof(
            current_index,
            block['hash_of_previous_block'],
            block['transaction'],
            block['nonce']):
            return False

        last_block = block
        current_index += 1

    return True

def update_blockchain(self):
    neighbours = self.nodes
    new_chain = None

    max_length = len(self.chain)

    for node in neighbours:
        response = requests.get(f'http://{node}/blockchain')

        if response.status_code == 200:
            length = response.json()['length']
            chain = response.json()['chain']
```

```
            if length > max_length and self.valid_chain(chain):
                max_length = length
                new_chain = chain

            if new_chain:
                self.chain = new_chain
                return True

        return False




    def proof_of_work(self, index, hash_of_previous_block, transactions):
        nonce = 0

        while self.valid_proof(index, hash_of_previous_block, transactions, nonce) is False:
            nonce += 1
        return nonce

    def valid_proof(self, index, hash_of_previous_block, transactions, nonce):
        content = f'{index}{hash_of_previous_block}{transactions}{nonce}'.encode()

        content_hash = hashlib.sha256(content).hexdigest()

        return content_hash[:len(self.difficulty_target)] == self.difficulty_target

    def append_block(self, nonce, hash_of_previous_block):
        block = {
            'index': len(self.chain),
            'timestamp': time(),
            'transaction': self.current_transactions,
            'nonce': nonce,
            'hash_of_previous_block': hash_of_previous_block
        }

        self.current_transactions = []

        self.chain.append(block)
        return block

    def add_transaction(self, sender, recipient, amount):
        self.current_transactions.append({
```

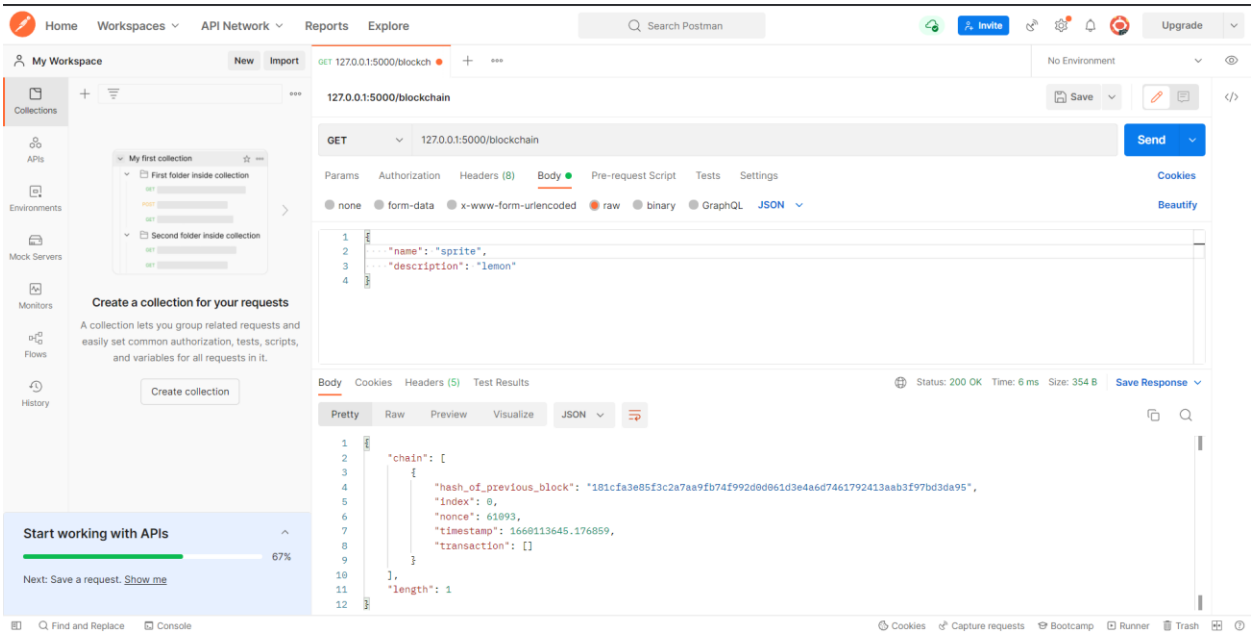
```
            'amount': amount,
            'recipient': recipient,
            'sender': sender
        })
        return self.last_block['index'] + 1

    @property
    def last_block(self):
        return self.chain[-1]

app = Flask(__name__)

node_identifier = str(uuid4()).replace('-', "")

blockchain = Blockchain()

#routes
@app.route('/blockchain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain)
    }
    return jsonify(response), 200

@app.route('/mine', methods=['GET'])
def mine_block():
    blockchain.add_transaction(
        sender = "0",
        recipient = node_identifier,
        amount = 1
    )

    last_block_hash = blockchain.hash_block(blockchain.last_block)

    index = len(blockchain.chain)
    nonce = blockchain.proof_of_work(index, last_block_hash, blockchain.current_transactions)

    block = blockchain.append_block(nonce, last_block_hash)
    response = {
        'message': "new block has been added (mined)",
        'index': block['index'],
        'hash_of_previous_block': block['hash_of_previous_block'],
```

```
        'nonce': block['nonce'],
        'transaction': block['transaction']
    }

    return jsonify(response), 200

@app.route('/transactions/new', methods=['POST'])
def new_transactions():
    values = request.get_json()

    required_fields = ['sender', 'recipient', 'amount']
    if not all (k in values for k in required_fields):
        return ('Missing fields', 400)

    index = blockchain.add_transaction(
        values['sender'],
        values['recipient'],
        values['amount']
    )

    response = {'message': f'Transaction will be added to the block {index}'}
    return (jsonify(response), 201)

@app.route('/nodes/add_nodes', methods=['POST'])
def add_nodes():
    values = request.get_json()
    nodes = values.get('nodes')

    if nodes is None:
        return "Error, missing node(s) info", 400

    for node in nodes:
        blockchain.add_node(node)

    response = {
        'message': 'New nodes have been added',
        'nodes': list(blockchain.nodes)
    }

    return jsonify(response), 200

@app.route('/nodes/sync', methods=['GET'])
def sync():
```

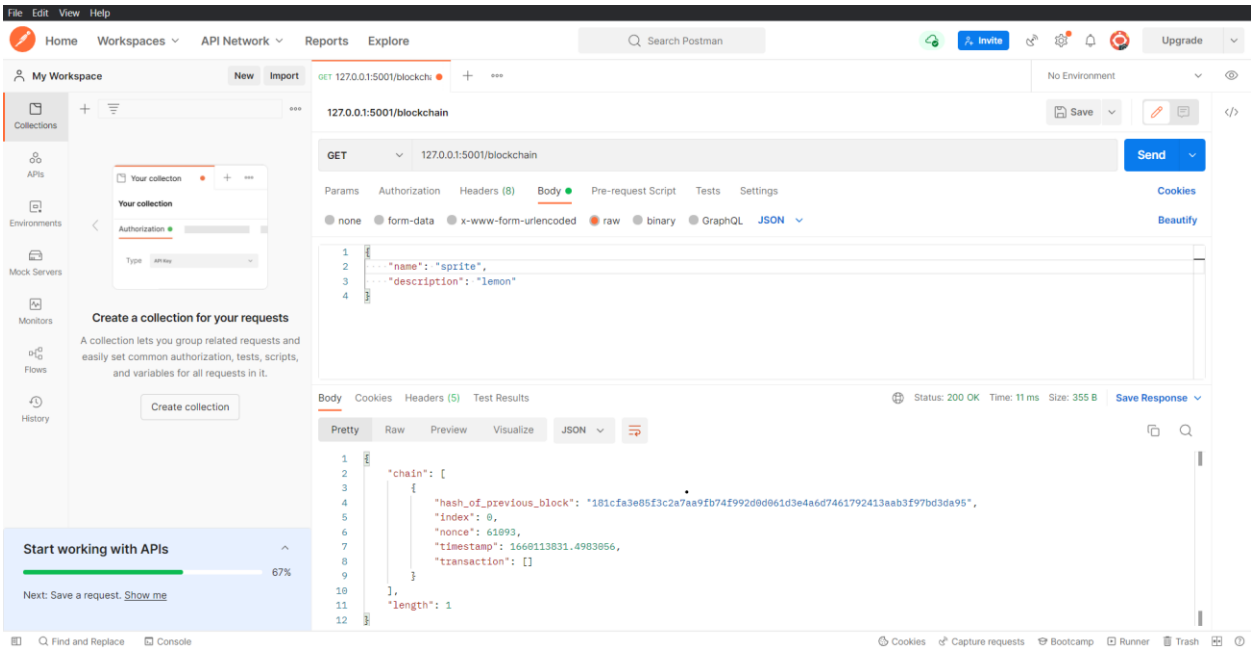
```
  updated = blockchain.update_blockchain()
  if updated:
    response = {
      'message': 'Blockchain has been updated with new data',
      'blockchain': blockchain.chain
    }
  else:
    response = {
      'message': 'blockchain already uses the latest data',
      'blockchain': blockchain.chain
    }

  return jsonify(response), 200



if __name__ == '__main__':
  app.run(host='0.0.0.0', port=int(sys.argv[1]))
```

Output:-

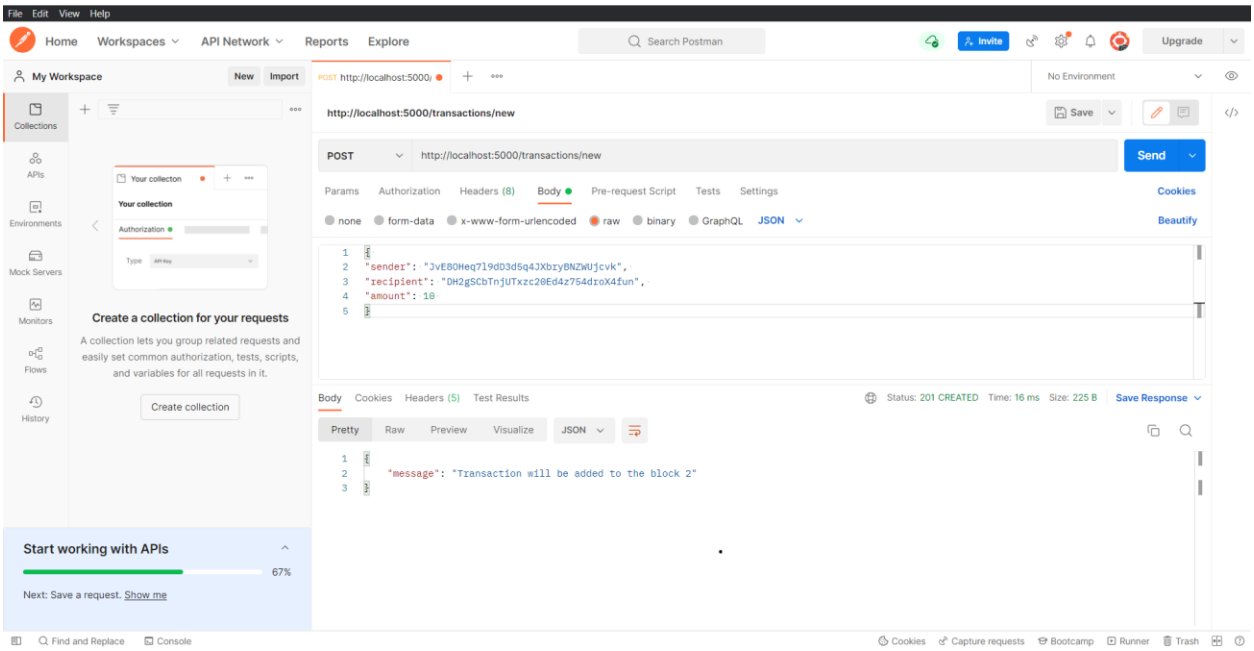
on the 5000 port:



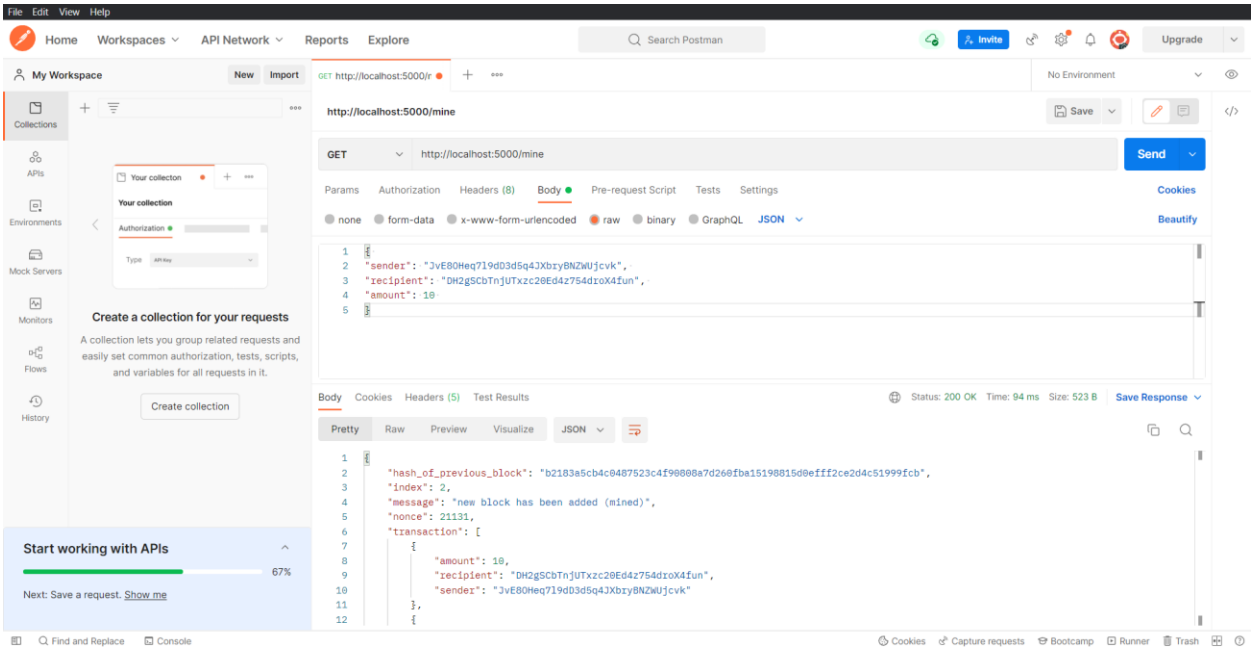
On the 5001 port:



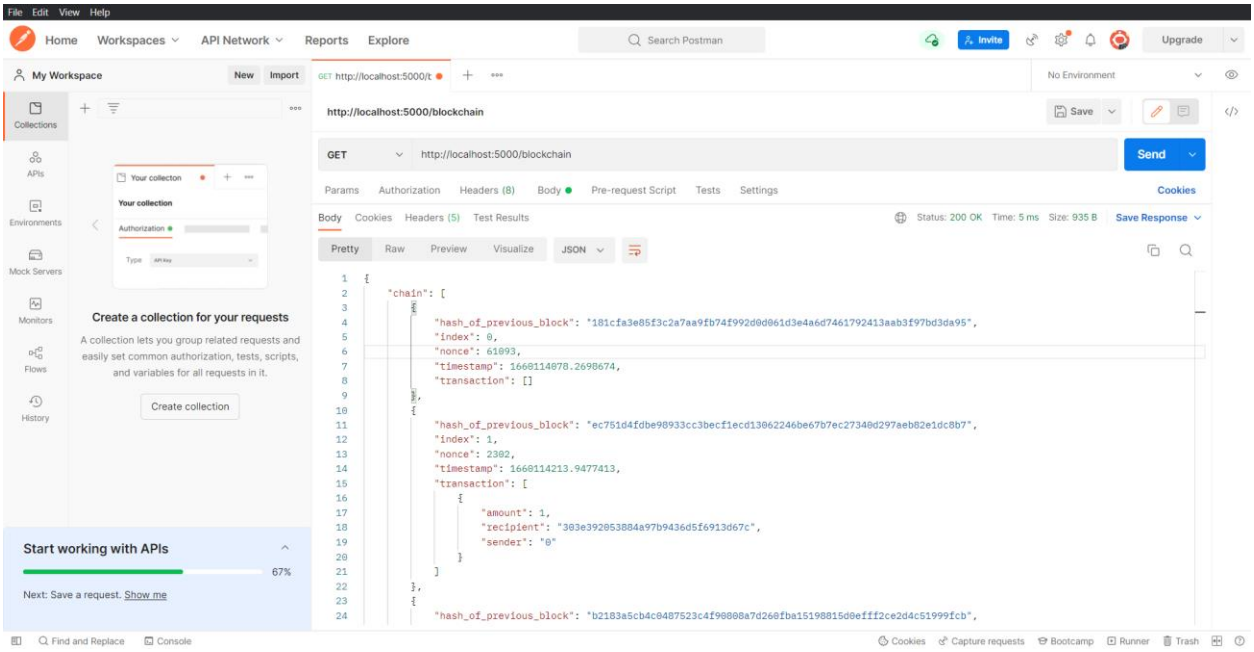
Adding new transaction:

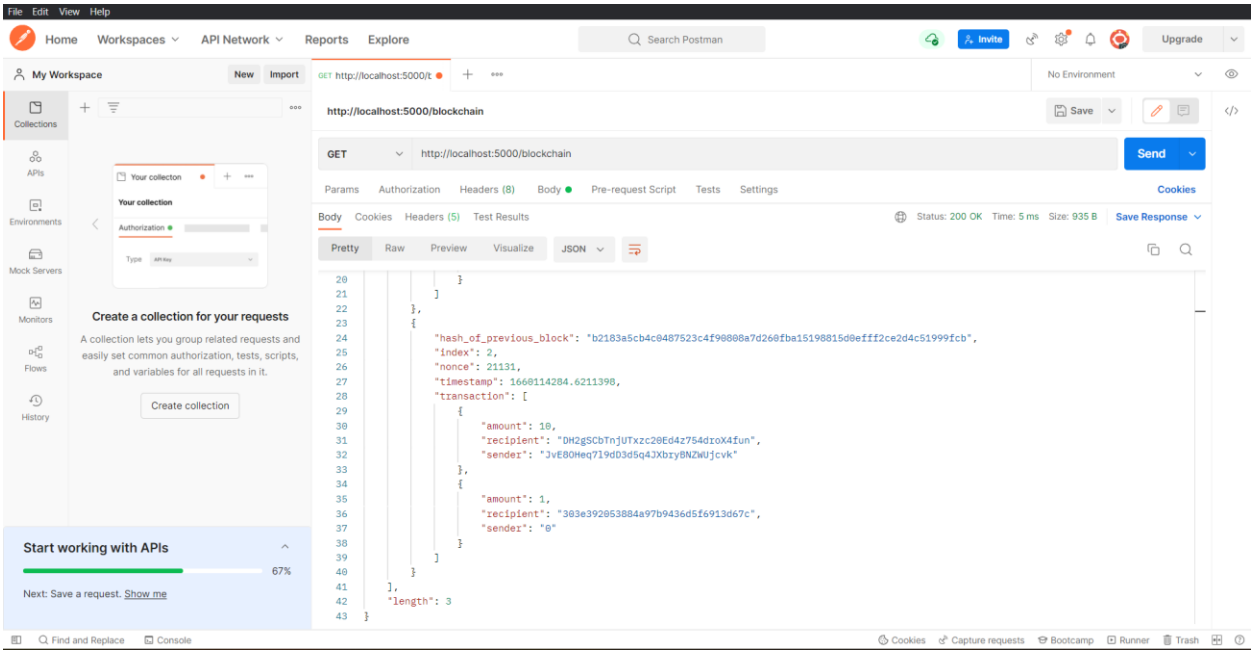


Mining a new block:

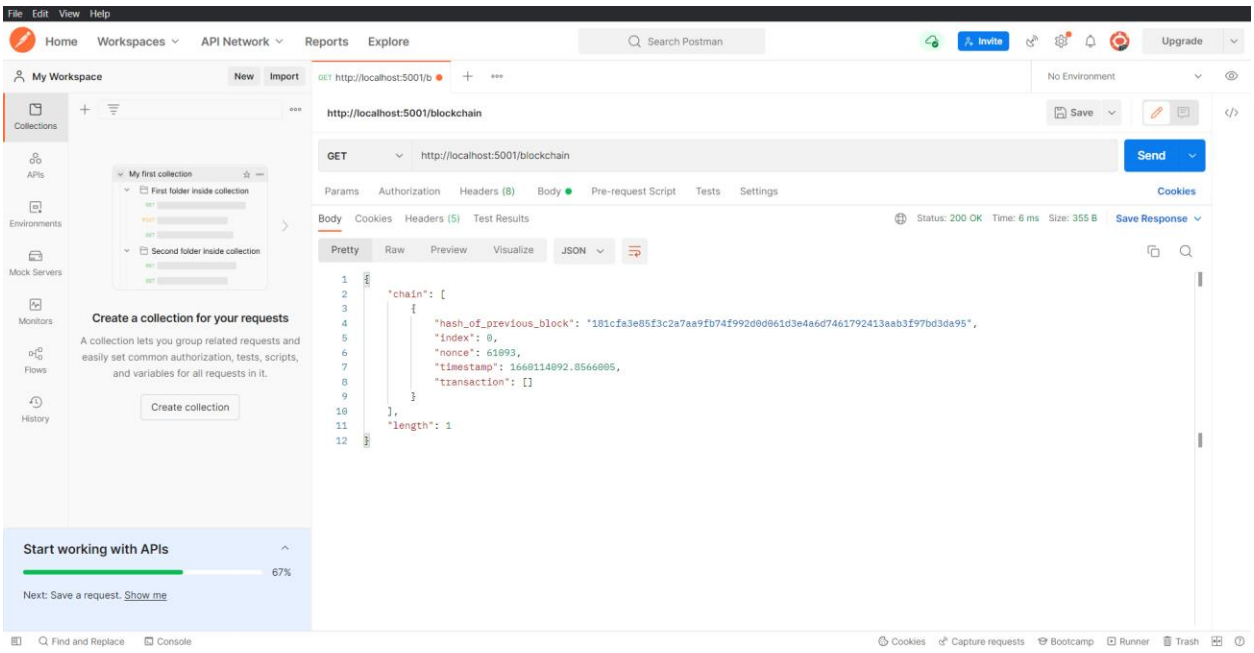


After adding new blocks:

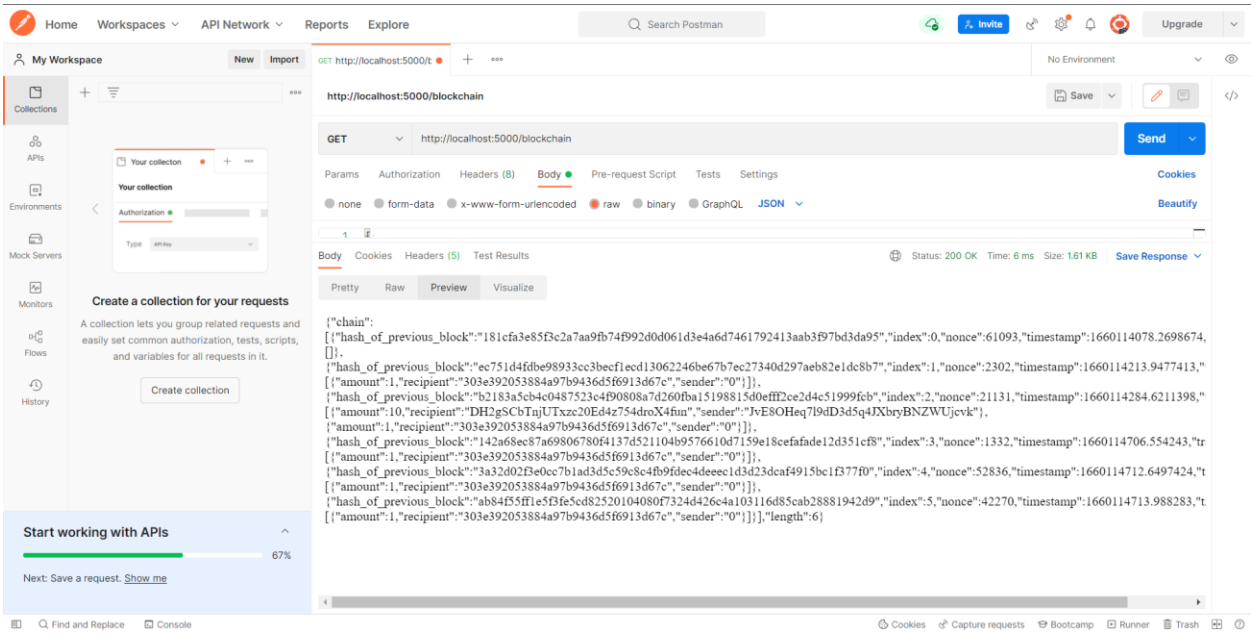




Block on 5001 port has not changed:



Total 6 blocks were made:



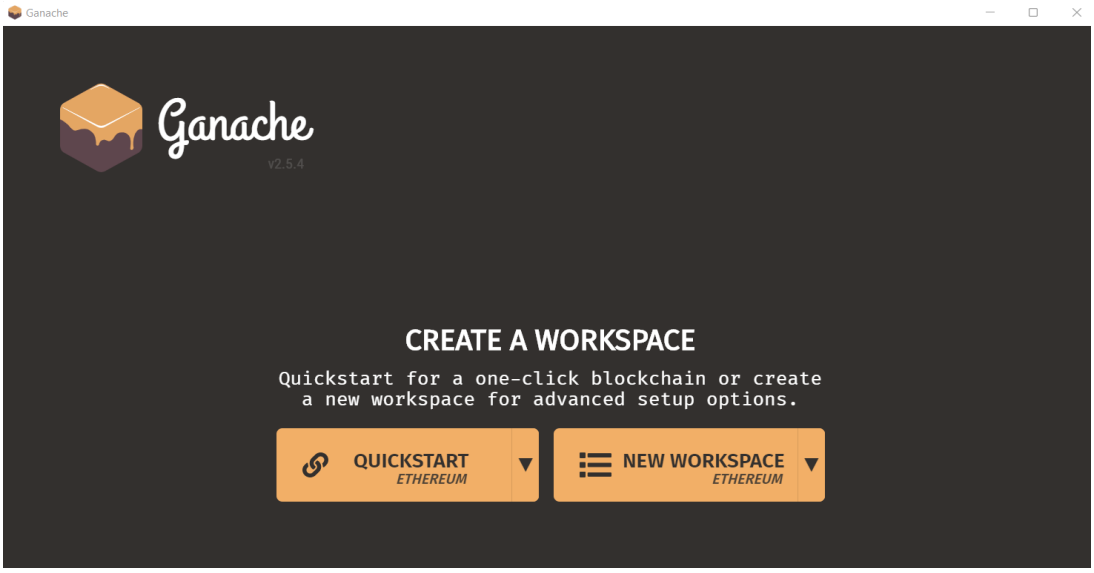
Practical-2

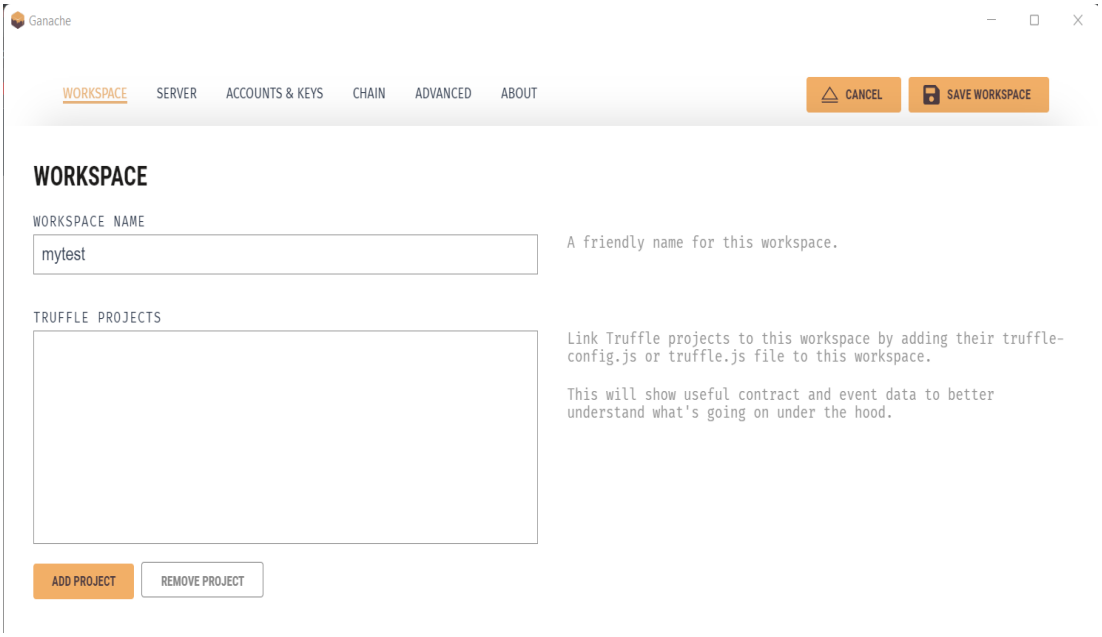
Aim:

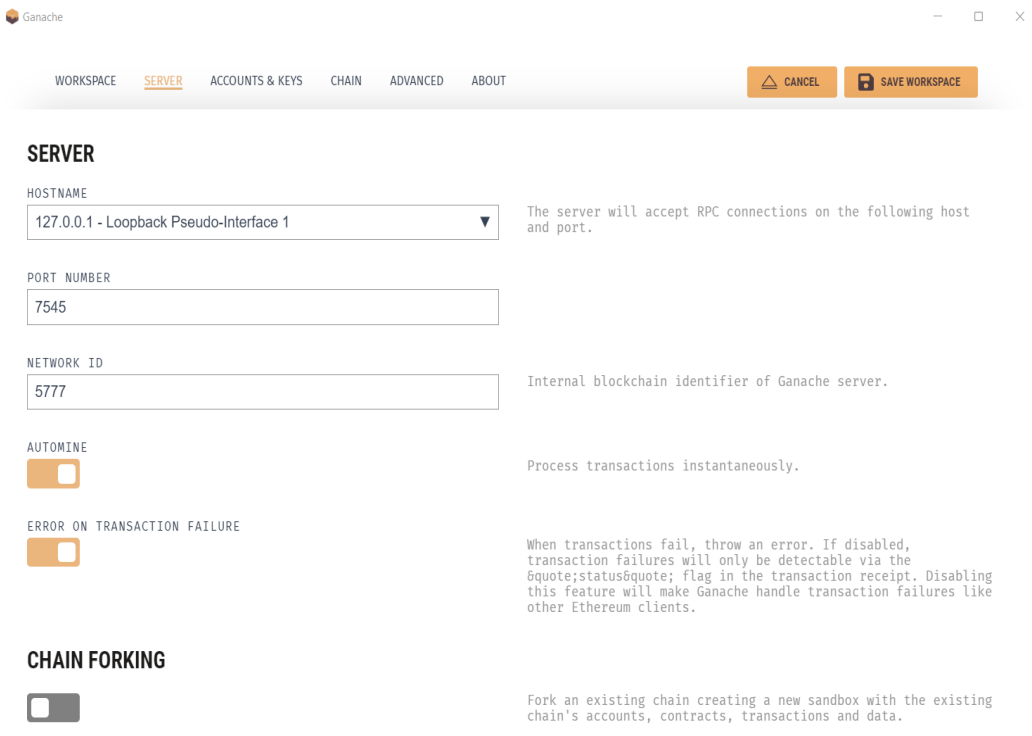
Study and configuration of Ganache, Geth and Meta mask. Import accounts and configure all the test networks.

Steps:

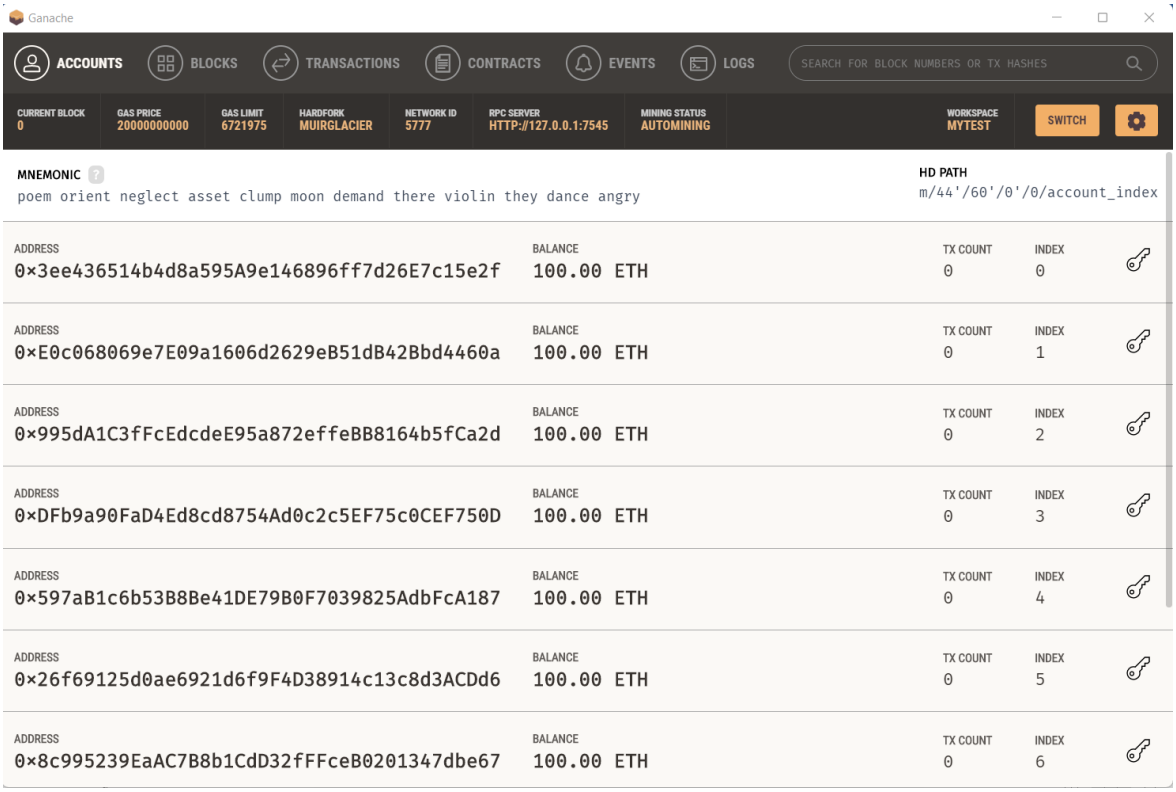
download ganache, Geth, Meta Mask.
configure ganache, Geth, Meta Mask and create new workspace following the steps




Ganache
WORKSPACE
SERVER
ACCOUNTS & KEYS
CHAIN
ADVANCED
ABOUT
CANCEL
SAVE WORKSPACE
WORKSPACE
WORKSPACE NAME
mytest
A friendly name for this workspace.
TRUFFLE PROJECTS
Link Truffle projects to this workspace by adding their truffle-config.js or truffle.js file to this workspace.
This will show useful contract and event data to better understand what's going on under the hood.
ADD PROJECT
REMOVE PROJECT

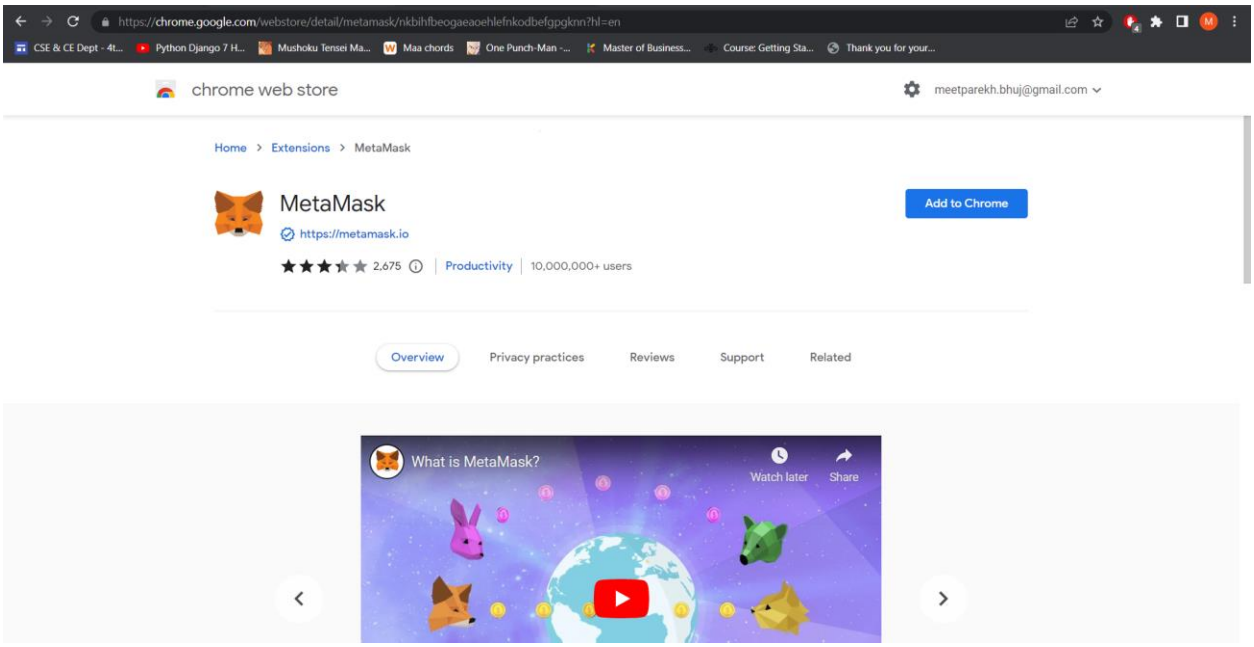

Ganache
WORKSPACE
SERVER
ACCOUNTS & KEYS
CHAIN
ADVANCED
ABOUT
CANCEL
SAVE WORKSPACE
SERVER
HOSTNAME
127.0.0.1 - Loopback Pseudo-Interface 1
The server will accept RPC connections on the following host and port.
PORT NUMBER
7545
NETWORK ID
5777
Internal blockchain identifier of Ganache server.
AUTOMINE
Process transactions instantaneously.
ERROR ON TRANSACTION FAILURE
When transactions fail, throw an error. If disabled, transaction failures will only be detectable via the &quote;status&quote; flag in the transaction receipt. Disabling this feature will make Ganache handle transaction failures like other Ethereum clients.
CHAIN FORKING
Fork an existing chain creating a new sandbox with the existing chain's accounts, contracts, transactions and data.

Home screen:

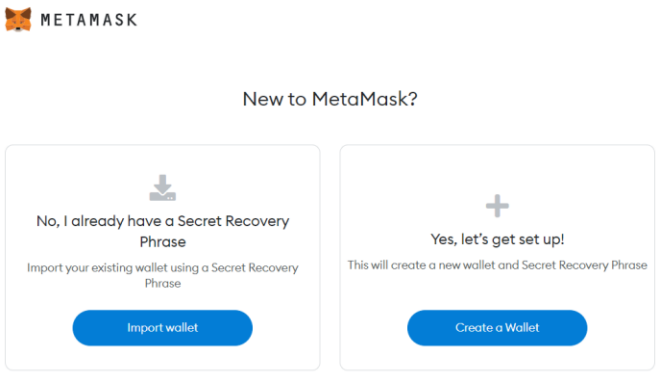


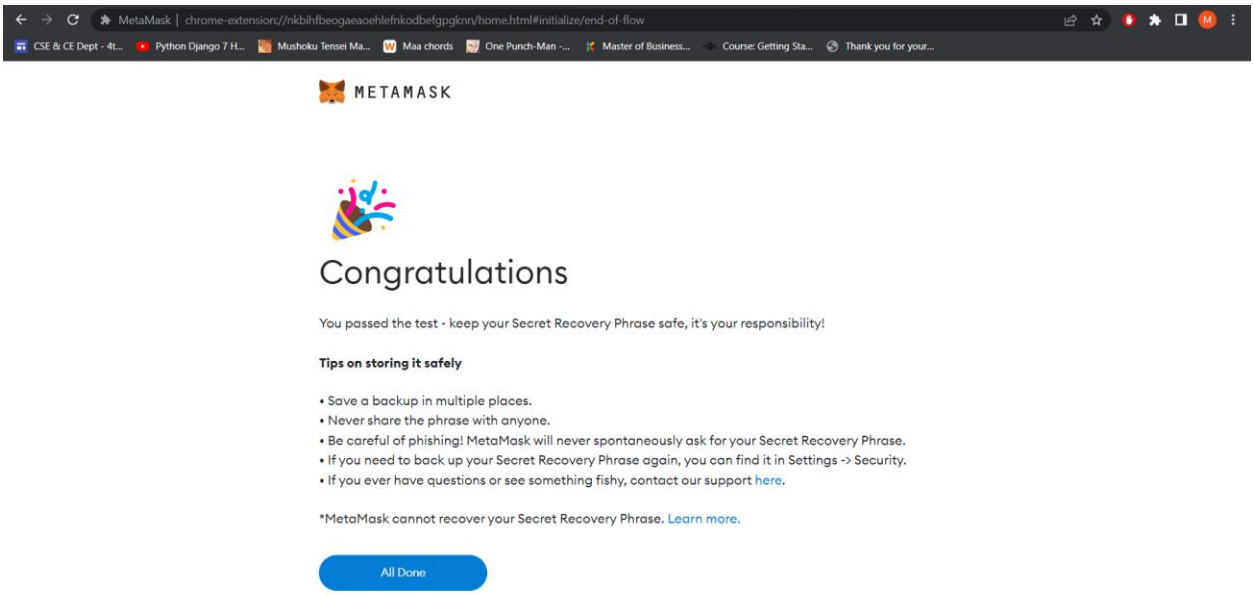
Download and install geth

Add Meta Mask to chrome

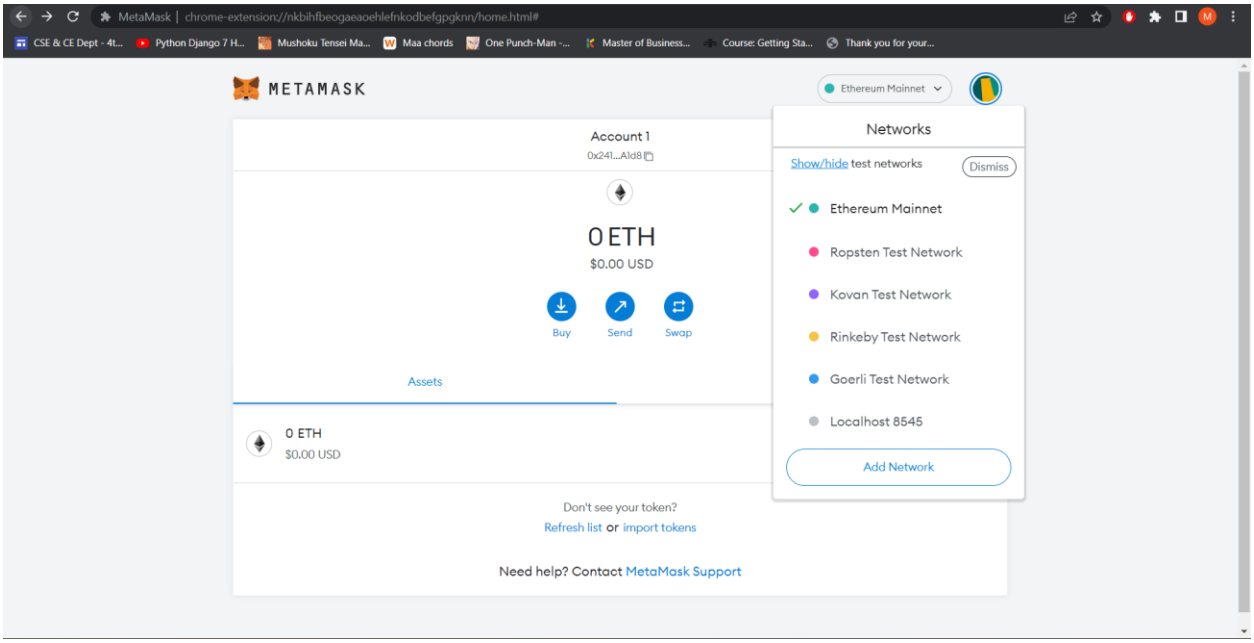


Create a wallet on metamask



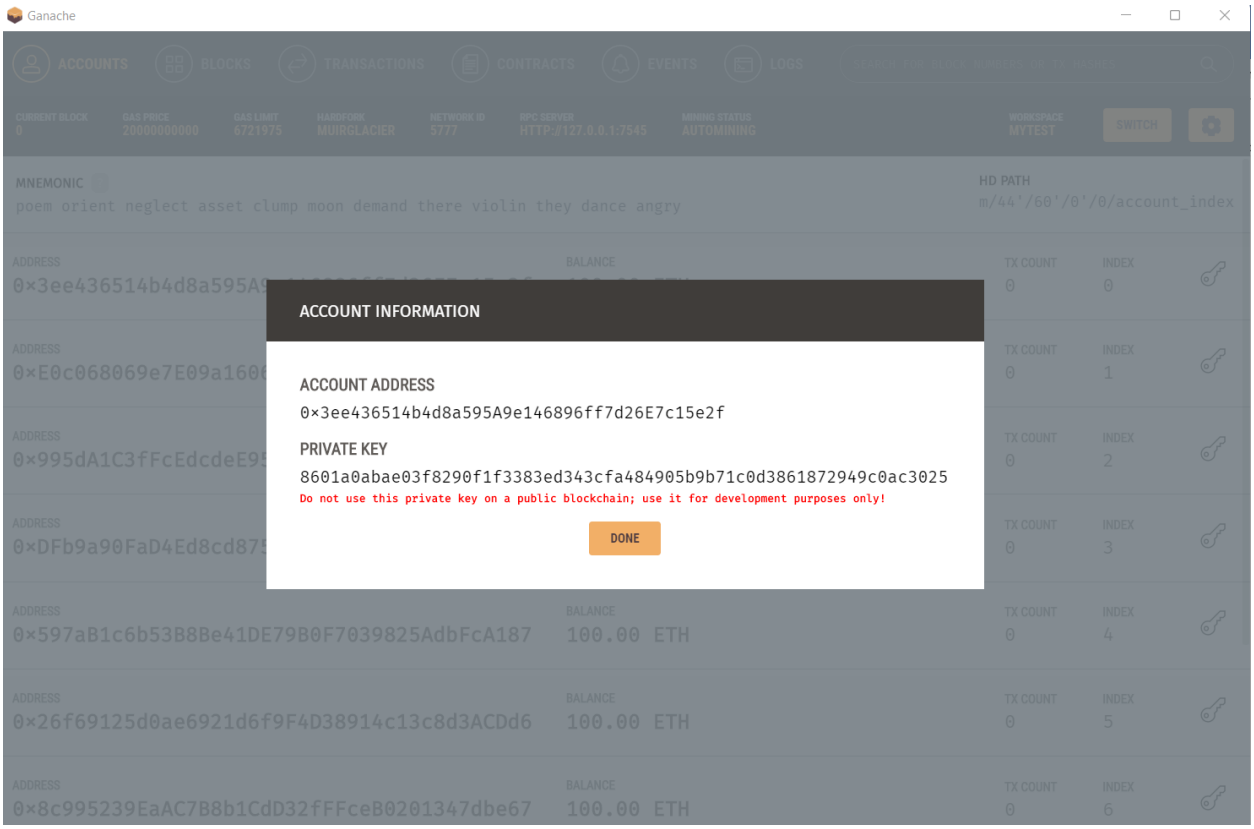


Home screen

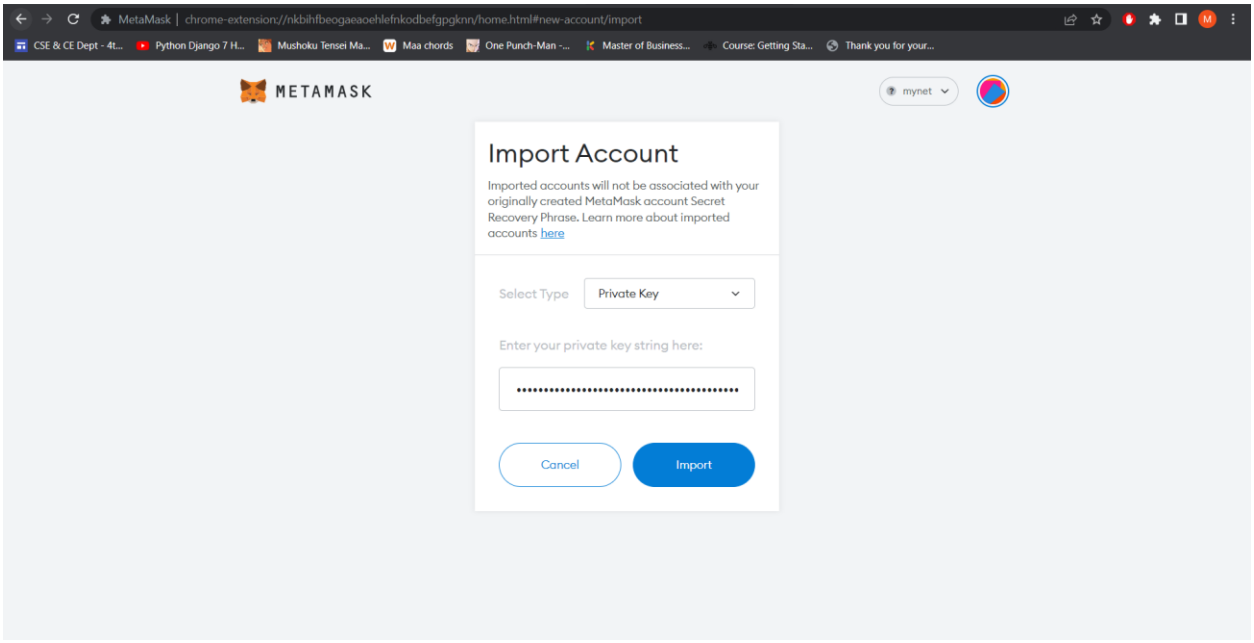


Importing account from ganache

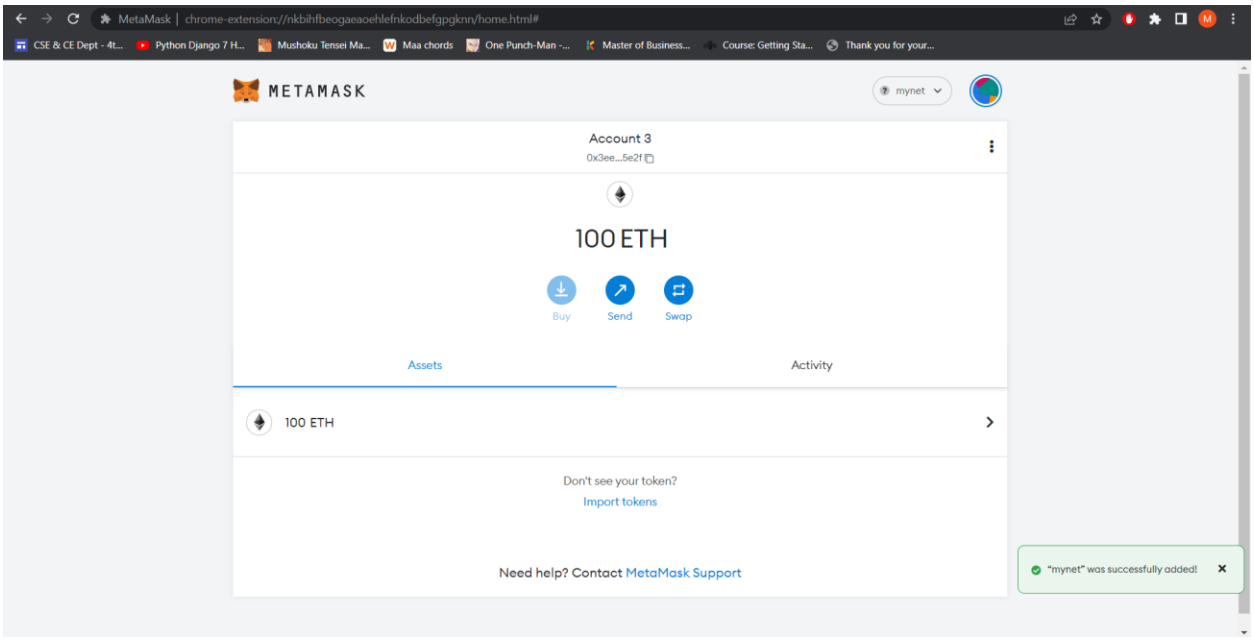
Copy the private key of environment



Paste private key



Account imported


MetaMask | chrome-extension://nkbihfbeogaeaoehlefnkodbefgpgknn/home.html#
CSE & CE Dept - 4t...
Python Django 7 H...
Mushoku Tensei Ma...
Maa chords
One Punch-Man -...
Master of Business...
Course: Getting Sta...
Thank you for your...
METAMASK
mynet
Account 3
0x3ee...5e2f
100 ETH
Buy
Send
Swap
Assets
Activity
100 ETH
Don't see your token?
Import tokens
Need help? Contact MetaMask Support
"mynet" was successfully added!

Practical-3

Aim:- Study and Configure Geth over Windows/Linux. Perform following tasks:

a. Build Your Own Ethereum Private Blockchain with Four peers.

b. Build Public Ethereum Blockchain using proof of work (POW) consensus mechanism.

c. Setup Proof of Authority (POA) Consensus on Ethereum Blockchain.

d. Self-study to build Ethereum 2.0 network using POS.

A. Build Your Own Ethereum Private Blockchain with Four peers.

Save this code as Gensis.json

```
{
   "config": {
      "chainId": 15,
      "homesteadBlock": 0,
      "eip155Block": 0,
      "eip158Block": 0
   },
   "difficulty": "0x400",
   "gasLimit": "0x2100000",
   "alloc": {
      "7a69b359e86893efa3d9732e4c65ced51567edd0":
       { "balance": "0x1337000000000000000000" }
   }
}
```

Then run the command on terminal

Geth –datadir node1 init Genesis.json

```
Command Prompt - cmd

C:\Users\ASUS\OneDrive\Desktop\sem 7\BCT\practical\practical-3>geth --datadir node2 init Genesis.json
INFO [08-13|13:32:40.992] Maximum peer count                       ETH=50 LES=0 total=50
INFO [08-13|13:32:41.002] Set global gas cap                       cap=50,000,000
INFO [08-13|13:32:41.012] Allocated cache and file handles         database="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\
BCT\\practical\\practical-3\\node2\\geth\\chaindata" cache=16.00MiB handles=16
INFO [08-13|13:32:41.046] Opened ancient database                  database="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\
BCT\\practical\\practical-3\\node2\\geth\\chaindata\\ancient" readonly=false
INFO [08-13|13:32:41.056] Writing custom genesis block
INFO [08-13|13:32:41.060] Persisted trie from memory database      nodes=1 size=150.00B time=0s gcnodes=0 gcsize=0.00B g
ctime=0s livenodes=1 livesize=0.00B
Fatal: Failed to write genesis block: unsupported fork ordering: eip150Block not enabled, but eip155Block enabled at 0

C:\Users\ASUS\OneDrive\Desktop\sem 7\BCT\practical\practical-3>
```

Now run the node with parameters given as
geth — datadir node1/ — syncmode “full” — port 30304 — http — http.addr “localhost” — http.port 8545 — http.api “personal,eth,net,web3,txpool,miner” — networkid 99 — identity node1 console

```
C:\Users\ASUS\OneDrive\Desktop\sem 7\BCT\practical\practical-3>geth --datadir node1/ --syncmode "full" --port 30304 --http --http.addr "localhost" --http.port 8545 --http.api "personal,eth,net,web3,txpool,miner"
 --networkid 99 --identity node1 console
INFO [08-13|13:34:33.783] Maximum peer count                       ETH=50 LES=0 total=50
INFO [08-13|13:34:33.793] Set global gas cap                       cap=50,000,000
INFO [08-13|13:34:33.800] Allocated trie memory caches             clean=154.00MiB dirty=256.00MiB
INFO [08-13|13:34:33.808] Allocated cache and file handles         database="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\BCT\\practical\\practical-3\\node1\\geth\\chaindata" cache=512.00MiB handles=8192
INFO [08-13|13:34:33.843] Opened ancient database                  database="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\BCT\\practical\\practical-3\\node1\\geth\\chaindata\\ancient" readonly=false
INFO [08-13|13:34:33.855] Writing default main-net genesis block
INFO [08-13|13:34:34.164] Persisted trie from memory database      nodes=12356 size=1.78MiB time=39.7158ms gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [08-13|13:34:34.195]
INFO [08-13|13:34:34.204] ---------------------------------------------------------------------------------------------------------------------------------------------------
INFO [08-13|13:34:34.213] Chain ID:  1 (mainnet)
INFO [08-13|13:34:34.217] Consensus: Ethash (proof-of-work)
INFO [08-13|13:34:34.220]
INFO [08-13|13:34:34.222] Pre-Merge hard forks:
INFO [08-13|13:34:34.225]  - Homestead:                    1150000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/homestead.md)
INFO [08-13|13:34:34.234]  - DAO Fork:                     1920000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/dao-fork.md)
INFO [08-13|13:34:34.242]  - Tangerine Whistle (EIP 150): 2463000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/tangerine-whistle.md)
INFO [08-13|13:34:34.252]  - Spurious Dragon/1 (EIP 155): 2675000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/spurious-dragon.md)
INFO [08-13|13:34:34.261]  - Spurious Dragon/2 (EIP 158): 2675000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/spurious-dragon.md)
INFO [08-13|13:34:34.271]  - Byzantium:                    4370000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/byzantium.md)
INFO [08-13|13:34:34.279]  - Constantinople:               7280000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/constantinople.md)
INFO [08-13|13:34:34.289]  - Petersburg:                   7280000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/petersburg.md)
INFO [08-13|13:34:34.299]  - Istanbul:                     9069000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/istanbul.md)
INFO [08-13|13:34:34.308]  - Muir Glacier:                 9200000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/muir-glacier.md)
INFO [08-13|13:34:34.315]  - Berlin:                       12244000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/berlin.md)
INFO [08-13|13:34:34.325]  - London:                       12965000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/london.md)
INFO [08-13|13:34:34.334]  - Arrow Glacier:                13773000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/arrow-glacier.md)
INFO [08-13|13:34:34.343]  - Gray Glacier:                 15050000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/gray-glacier.md)
INFO [08-13|13:34:34.352]
INFO [08-13|13:34:34.355] The Merge is not yet available for this network!
INFO [08-13|13:34:34.355]  - Hard-fork specification: https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/paris.md)
INFO [08-13|13:34:34.364] ---------------------------------------------------------------------------------------------------------------------------------------------------
INFO [08-13|13:34:34.379]
INFO [08-13|13:34:34.384] Disk storage enabled for ethash caches   dir="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\BCT\\practical\\practical-3\\node1\\geth\\ethash" count=3
INFO [08-13|13:34:34.393] Disk storage enabled for ethash DAGs     dir=C:\Users\ASUS\AppData\Local\Ethash count=2
INFO [08-13|13:34:34.398] Initialising Ethereum protocol           network=99 dbversion=<nil>
INFO [08-13|13:34:34.413] Loaded most recent local header          number=0 hash=d4e567..cb8fa3 td=17,179,869,184 age=53y4mo2w
INFO [08-13|13:34:34.423] Loaded most recent local full block      number=0 hash=d4e567..cb8fa3 td=17,179,869,184 age=53y4mo2w
INFO [08-13|13:34:34.433] Loaded most recent local fast block      number=0 hash=d4e567..cb8fa3 td=17,179,869,184 age=53y4mo2w
WARN [08-13|13:34:34.444] Failed to load snapshot, regenerating    err="missing or corrupted snapshot"
INFO [08-13|13:34:34.452] Rebuilding state snapshot
INFO [08-13|13:34:34.457] Regenerated local transaction journal    transactions=0 accounts=0
INFO [08-13|13:34:34.457] Resuming state snapshot generation       root=d7f897..0f0544 accounts=0 slots=0 storage=0.00B dangling=0 elapsed=1.638ms
INFO [08-13|13:34:34.460] Gasprice oracle is ignoring threshold set threshold=2
WARN [08-13|13:34:34.469] Error reading unclean shutdown markers   error="leveldb: not found"
```

```
Welcome to the Geth JavaScript console!

instance: Geth/node1/v1.10.21-stable-67109427/windows-amd64/go1.18.4
at block: 0 (Thu Jan 01 1970 05:30:00 GMT+0530 (IST))
 datadir: C:\Users\ASUS\OneDrive\Desktop\sem 7\BCT\practical\practical-3\node1
 modules: admin:1.0 debug:1.0 engine:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> INFO [08-13|13:34:37.139] New local node record                  seq=1,660,377,874,501 id=f468b9e2466ebce5 ip=114.31.176.20 udp=2872  tcp=30304
WARN [08-13|13:34:37.546] Snapshot extension registration failed  peer=3af25a30 err="peer connected on snap without compatible eth support"
> INFO [08-13|13:34:45.425] Looking for peers                     peercount=0 tried=132 static=0
WARN [08-13|13:34:50.603] Snapshot extension registration failed  peer=a4da81c9 err="peer connected on snap without compatible eth support"
WARN [08-13|13:34:50.772] Snapshot extension registration failed  peer=a05e766c err="peer connected on snap without compatible eth support"
INFO [08-13|13:34:55.580] Looking for peers                       peercount=1 tried=127 static=0
INFO [08-13|13:35:05.601] Looking for peers                       peercount=0 tried=89  static=0
INFO [08-13|13:35:15.657] Looking for peers                       peercount=0 tried=107 static=0
```

Now create a new node named node2 and follow the steps again

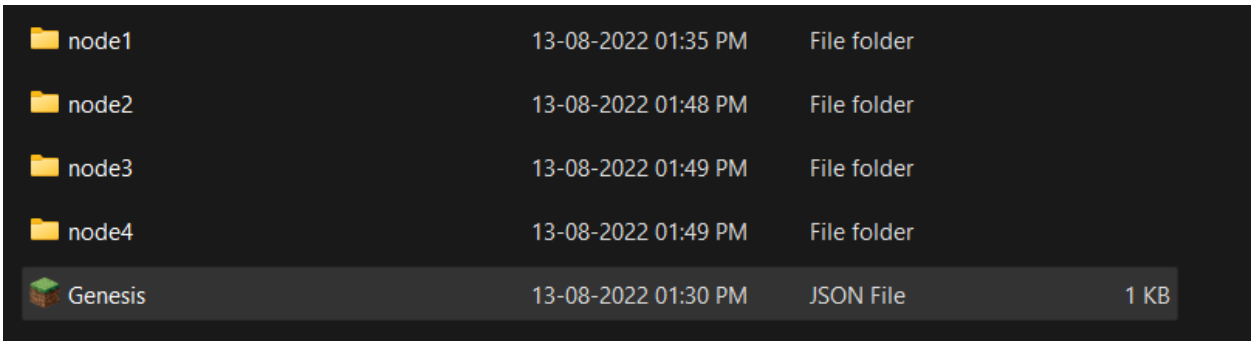
```
C:\Users\ASUS\OneDrive\Desktop\sem 7\BCT\practical\practical-3>Geth --datadir node2 init Genesis.json
INFO [08-13|13:36:36.203] Maximum peer count                       ETH=50 LES=0 total=50
INFO [08-13|13:36:36.213] Set global gas cap                       cap=50,000,000
INFO [08-13|13:36:36.221] Allocated cache and file handles         database="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\BCT\\practical\\practical-3\\node2\\geth\\chaindata" cache=16.00MiB handles=16
INFO [08-13|13:36:36.248] Opened ancient database                  database="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\BCT\\practical\\practical-3\\node2\\geth\\chaindata\\ancient" readonly=false
INFO [08-13|13:36:36.256] Writing custom genesis block
INFO [08-13|13:36:36.256] Persisted trie from memory database      nodes=1 size=150.00B time=0s gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
Fatal: Failed to write genesis block: unsupported fork ordering: eip150Block not enabled, but eip155Block enabled at 0
```

Now run the node with parameters given as geth — datadir node2/ — syncmode “full” — port 30305 — http — http.addr “localhost” — http.port 8546 — http.api “personal,eth,net,web3,txpool,miner” — networkid 99 — identity node2 console

```
C:\Users\ASUS\OneDrive\Desktop\sem 7\BCT\practical\practical-3>geth --datadir node2/ --syncmode "full" --port 30305 --http --http.addr "localhost" --http.port 8546 --http.api "personal,eth,net,web3,txpool,miner"
--networkid 99 --identity node2 console --ipcdisable
INFO [08-13|13:46:38.811] Maximum peer count                       ETH=50 LES=0 total=50
INFO [08-13|13:46:38.818] Set global gas cap                       cap=50,000,000
INFO [08-13|13:46:38.822] Allocated trie memory caches             clean=154.00MiB dirty=256.00MiB
INFO [08-13|13:46:38.832] Allocated cache and file handles         database="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\BCT\\practical\\practical-3\\node2\\geth\\chaindata" cache=512.00MiB handles=8192
INFO [08-13|13:46:38.860] Opened ancient database                  database="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\BCT\\practical\\practical-3\\node2\\geth\\chaindata\\ancient" readonly=false
INFO [08-13|13:46:38.868]
INFO [08-13|13:46:38.869] ---------------------------------------------------------------------------------------------------------------------------------------------------
INFO [08-13|13:46:38.876] Chain ID:  1 (mainnet)
INFO [08-13|13:46:38.877] Consensus: Ethash (proof-of-work)
INFO [08-13|13:46:38.880]
INFO [08-13|13:46:38.882] Pre-Merge hard forks:
INFO [08-13|13:46:38.884]  - Homestead:                    1150000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/homestead.md)
INFO [08-13|13:46:38.889]  - DAO Fork:                     1920000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/dao-fork.md)
INFO [08-13|13:46:38.895]  - Tangerine Whistle (EIP 150): 2463000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/tangerine-whistle.md)
INFO [08-13|13:46:38.899]  - Spurious Dragon/1 (EIP 155): 2675000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/spurious-dragon.md)
INFO [08-13|13:46:38.905]  - Spurious Dragon/2 (EIP 158): 2675000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/spurious-dragon.md)
INFO [08-13|13:46:38.910]  - Byzantium:                    4370000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/byzantium.md)
INFO [08-13|13:46:38.915]  - Constantinople:               7280000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/constantinople.md)
INFO [08-13|13:46:38.921]  - Petersburg:                   7280000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/petersburg.md)
INFO [08-13|13:46:38.926]  - Istanbul:                     9069000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/istanbul.md)
INFO [08-13|13:46:38.933]  - Muir Glacier:                 9200000  (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/muir-glacier.md)
INFO [08-13|13:46:38.938]  - Berlin:                       12244000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/berlin.md)
INFO [08-13|13:46:38.944]  - London:                       12965000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/london.md)
INFO [08-13|13:46:38.948]  - Arrow Glacier:                13773000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/arrow-glacier.md)
INFO [08-13|13:46:38.955]  - Gray Glacier:                 15050000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/gray-glacier.md)
INFO [08-13|13:46:38.959]
INFO [08-13|13:46:38.960] The Merge is not yet available for this network!
INFO [08-13|13:46:38.964]  - Hard-fork specification: https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/paris.md)
INFO [08-13|13:46:38.968] ---------------------------------------------------------------------------------------------------------------------------------------------------
INFO [08-13|13:46:38.974]
INFO [08-13|13:46:38.977] Disk storage enabled for ethash caches   dir="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\BCT\\practical\\practical-3\\node2\\geth\\ethash" count=3
INFO [08-13|13:46:38.983] Disk storage enabled for ethash DAGs     dir=C:\Users\ASUS\AppData\Local\Ethash count=2
INFO [08-13|13:46:38.987] Initialising Ethereum protocol           network=99 dbversion=8
INFO [08-13|13:46:38.987] Loaded most recent local header          number=0 hash=d4e567..cb8fa3 td=17,179,869,184 age=53y4mo2w
INFO [08-13|13:46:38.996] Loaded most recent local full block      number=0 hash=d4e567..cb8fa3 td=17,179,869,184 age=53y4mo2w
INFO [08-13|13:46:39.003] Loaded most recent local fast block      number=0 hash=d4e567..cb8fa3 td=17,179,869,184 age=53y4mo2w
WARN [08-13|13:46:39.013] Loaded snapshot journal                  diffs=missing
INFO [08-13|13:46:39.016] Loaded local transaction journal         transactions=0 dropped=0
INFO [08-13|13:46:39.019] Regenerated local transaction journal    transactions=0 accounts=0
INFO [08-13|13:46:39.023] Gasprice oracle is ignoring threshold set threshold=2
WARN [08-13|13:46:39.025] Unclean shutdown detected                booted=2022-08-13T13:40:23+0530 age=6m16s
WARN [08-13|13:46:39.029] Unclean shutdown detected                booted=2022-08-13T13:41:45+0530 age=4m54s
```

```
INFO [08-13|13:46:38.987] Initialising Ethereum protocol           network=99 dbversion=8
INFO [08-13|13:46:38.987] Loaded most recent local header          number=0 hash=d4e567..cb8fa3 td=17,179,869,184 age=53y4mo2w
INFO [08-13|13:46:38.996] Loaded most recent local full block      number=0 hash=d4e567..cb8fa3 td=17,179,869,184 age=53y4mo2w
INFO [08-13|13:46:39.003] Loaded most recent local fast block      number=0 hash=d4e567..cb8fa3 td=17,179,869,184 age=53y4mo2w
WARN [08-13|13:46:39.013] Loaded snapshot journal                  diffs=missing
INFO [08-13|13:46:39.016] Loaded local transaction journal         transactions=0 dropped=0
INFO [08-13|13:46:39.019] Regenerated local transaction journal    transactions=0 accounts=0
INFO [08-13|13:46:39.023] Gasprice oracle is ignoring threshold set threshold=2
WARN [08-13|13:46:39.025] Unclean shutdown detected                booted=2022-08-13T13:40:23+0530 age=6m16s
WARN [08-13|13:46:39.029] Unclean shutdown detected                booted=2022-08-13T13:41:45+0530 age=4m54s
WARN [08-13|13:46:39.032] Unclean shutdown detected                booted=2022-08-13T13:44:35+0530 age=2m4s
WARN [08-13|13:46:39.035] Unclean shutdown detected                booted=2022-08-13T13:45:26+0530 age=1m13s
WARN [08-13|13:46:39.039] Unclean shutdown detected                booted=2022-08-13T13:45:59+0530 age=40s
WARN [08-13|13:46:39.041] Unclean shutdown detected                booted=2022-08-13T13:46:19+0530 age=20s
WARN [08-13|13:46:39.045] Engine API enabled                       protocol=eth
WARN [08-13|13:46:39.046] Engine API started but chain not configured for merge yet
INFO [08-13|13:46:39.049] Starting peer-to-peer node               instance=Geth/node2/v1.10.21-stable-67109427/windows-amd64/go1.18.4
INFO [08-13|13:46:39.080] New local node record                    seq=1,660,378,305,930 id=2b0c959c5803f293 ip=127.0.0.1 udp=30305 tcp=30305
INFO [08-13|13:46:39.091] Started P2P networking                   self=enode://2fb75ce744cb36bfcc2d61328001b122c6c6c218a3c5e6dbc5eac31693182ae482bded3cf0fe7469698361dc6fd7163a361cb7e3559b50b9d863e50e47405fa0@127.0.0.1:30305
INFO [08-13|13:46:39.091] Loaded JWT secret file                   path="C:\\Users\\ASUS\\OneDrive\\Desktop\\sem 7\\BCT\\practical\\practical-3\\node2\\geth\\jwtsecret" crc32=0xfca2b6f
INFO [08-13|13:46:39.113] HTTP server started                      endpoint=127.0.0.1:8546 auth=false prefix= cors= vhosts=localhost
INFO [08-13|13:46:39.118] WebSocket enabled                        url=ws://127.0.0.1:8551
INFO [08-13|13:46:39.121] HTTP server started                      endpoint=127.0.0.1:8551 auth=true  prefix= cors=localhost vhosts=localhost
WARN [08-13|13:46:39.175] Served eth_coinbase                      reqid=3 duration=0s err="etherbase must be explicitly specified"
Welcome to the Geth JavaScript console!

instance: Geth/node2/v1.10.21-stable-67109427/windows-amd64/go1.18.4
at block: 0 (Thu Jan 01 1970 05:30:00 GMT+0530 (IST))
 datadir: C:\Users\ASUS\OneDrive\Desktop\sem 7\BCT\practical\practical-3\node2
 modules: admin:1.0 debug:1.0 engine:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> WARN [08-13|13:46:42.191] Snapshot extension registration failed   peer=3af25a30 err="peer connected on snap without compatible eth support"
INFO [08-13|13:46:42.406] New local node record                    seq=1,660,378,305,931 id=2b0c959c5803f293 ip=114.31.176.20 udp=42374 tcp=30305
WARN [08-13|13:46:43.080] Snapshot extension registration failed   peer=1d4fc8e3 err="peer connected on snap without compatible eth support"
WARN [08-13|13:46:44.774] Snapshot extension registration failed   peer=1fdbbcbd err="peer connected on snap without compatible eth support"
INFO [08-13|13:46:49.127] Looking for peers                        peercount=0 tried=130 static=0
INFO [08-13|13:46:59.150] Looking for peers                        peercount=0 tried=99  static=0
INFO [08-13|13:47:09.187] Looking for peers                        peercount=2 tried=38  static=0
> INFO [08-13|13:47:19.353] Looking for peers                        peercount=0 tried=137 static=0
```

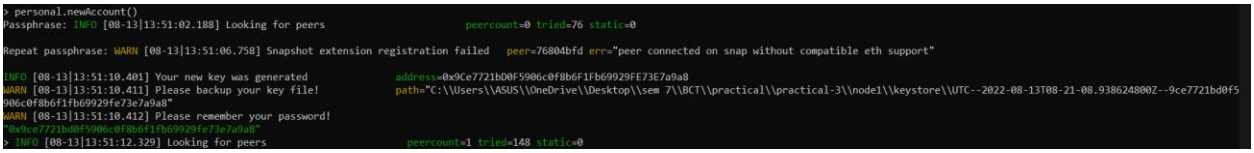
Now create 3rd and 4th node following the same steps



For node-1

Create your new account by executing following command

Personal.newAccount()

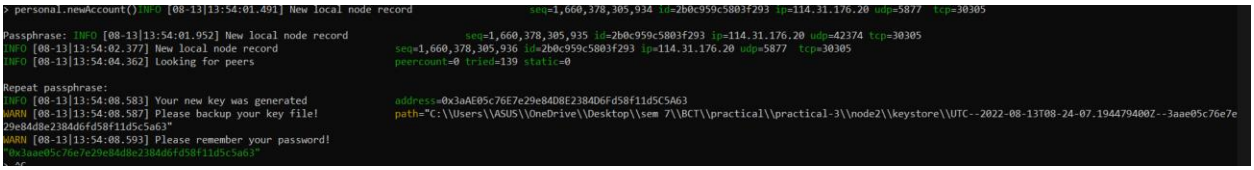


Key is: "0x9ce7721bd0f5906c0f8b6f1fb69929fe73e7a9a8"

For node -2

Create your new account by executing following command

Personal.newAccount()



Key is : "0x3aae05c76e7e29e84d8e2384d6fd58f11d5c5a63"

For node -3

Create your new account by executing following command

Personal.newAccount()

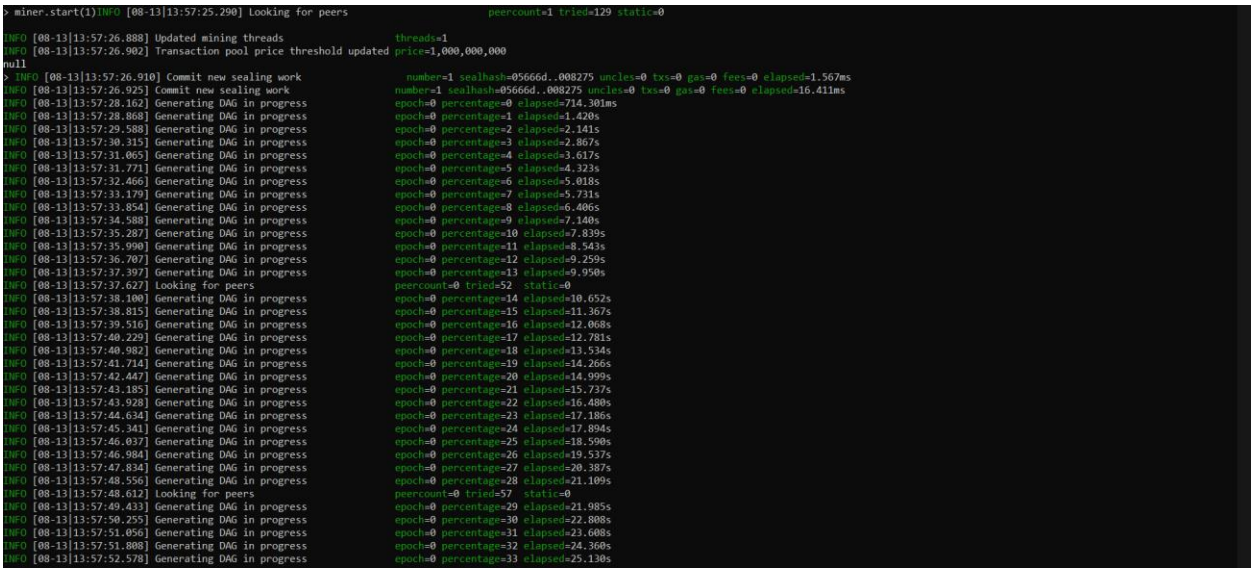
Key is: "0x88b62790dc14f3cec68f44bcc4ebfdae361965ba"

For node -4

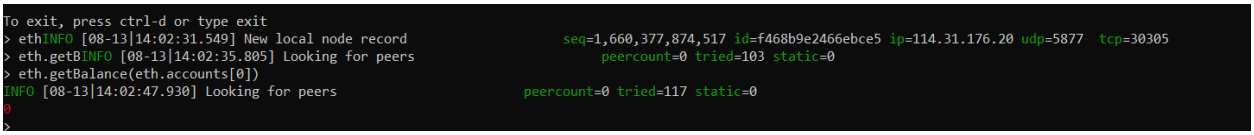
Create your new account by executing following command
Personal.newAccount()

Key is: "0x7bbba3731493e16bc7b9db2c46c8e6140e6cbdfa"

Now start mining in any node
Execute the command
Start.miner(1)



Check balance
Execute the command
eth.getBalance(eth.accounts[0])



Now add peers to node 1 blockchain

Copy the enode id of node 2 3 and 4

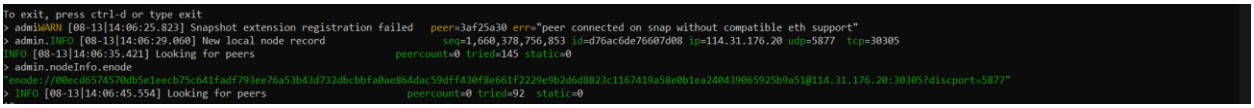
node2:
"enode://2fb75ce744cb36bfcc2d61328001b122c6c6c218a3c5e6dbc5eac31693182ae482bded3cf0fe7469698361dc6fd7163a361cb7e3559b50b9d863e50e47405fa0@114.31.176.20:30305?discport=5877"

node3:

"enode://8ae6ce43da5011202f67403a1ce92fc2c9b8d80c401b86eff9902ffe18a909f9aef449dd5f5a71c7649c33e6111bee463461ce8f4934f7a8db5ebe354b41efc4@114.31.176.20:30305?discport=5877"

node4:

"enode://00ecd6574570db5e1eecb75c641fadf793ee76a53b43d732dbcbbfa0ae864dac59dff430f8e661f2229e9b2d6d8823c1167419a58e0b1ea240439065925b9a51@114.31.176.20:30305?discport=5877"



Now add node 2 3 and 4 to node 1 as
admin.addPeer("enode://2fb75ce744cb36bfcc2d61328001b122c6c6c218a3c5e6dbc5eac31693182ae482bded3cf0fe7469698361dc6fd7163a361cb7e3559b50b9d863e50e47405fa0@localhost:30305?discport=5877")

after adding node 2 in node 1
admin.addPeer("enode://2fb75ce744cb36bfcc2d61328001b122c6c6c218a3c5e6dbc5eac31693182ae482bded3cf0fe7469698361dc6fd7163a3

61cb7e3559b50b9d863e50e47405fa0@localhost:30305?discport=5877")

```
To exit, press ctrl-d or type exit
> INFO [08-13|14:19:31.542] New local node record                 seq=1,660,377,874,529 id=f468b9e2466ebce5 ip=114.310 05
> admin.addPeer("enode://2fb75ce744cb36bfcc2d61328001b122c6c6c218a3c5e6dbc5eac31693182ae482bded3cf0fe7469698361dc6fd7163a361cb7e3559b50b9d863e50e47405fa0@localhost:30305?discport=5877")
true
```

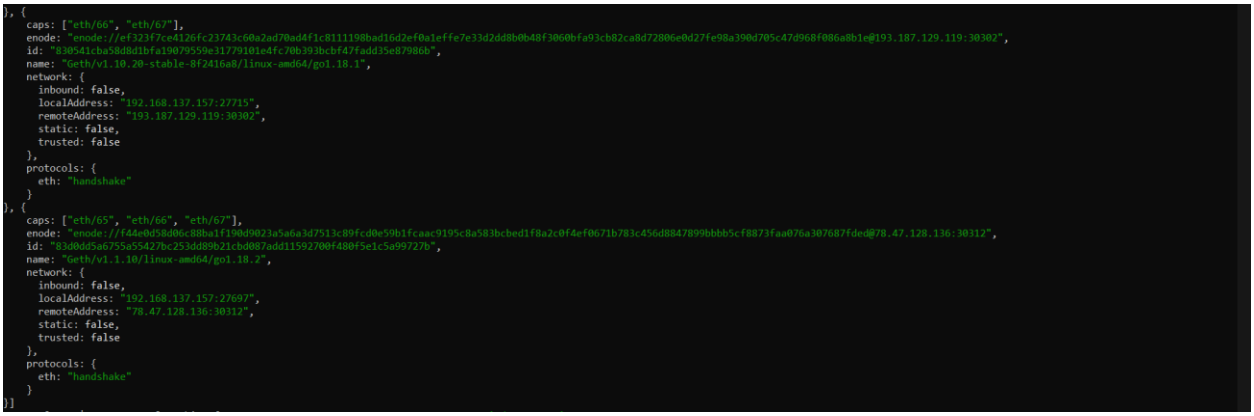
checking peers


```
> admin.peers
[{
    caps: ["eth/66", "snap/1"],
    enode: "enode://3021bee26ac6829196e044c80ebc9002b03130e16ecd446eb81766db0038d2215e08c5edb537ddf1f3ad96f2ce52f63c3df1f844726168bb88c6e2404426bac1@46.101.181.185:30388",
    id: "8391178994c13bf33b65e9bed99d382ebff127a31ba8a9f3b31260a212f03462",
    name: "Gubiq/v7.0.0-monoceros-c9009e89/linux-amd64/go1.17.6",
    network: {
      inbound: false,
      localAddress: "192.168.137.157:25697",
      remoteAddress: "46.101.181.185:30388",
      static: false,
      trusted: false
    },
    protocols: {
      eth: "handshake",
      snap: "handshake"
    }
}, {
    caps: ["eth/64", "eth/65", "eth/66", "istanbul/64", "istanbul/99", "istanbul/100", "snap/1"],
    enode: "enode://39c65456646f7eda23aece226544c54b6fe7a6e6bb62d0067874e1be1f7a0e1912da642a0de4b77e87bb061e1733a07af09afa61f207bd1e588818e35f7833ca@167.86.102.42:30300",
    id: "83bb98776d39f63a17d8f69a27a45037d48fc126f5ad6734ab120faa01aa1d13",
    name: "Geth/-pug/v1.10.0-stable(quorum-v22.1.0)/linux-amd64/go1.17.11",
    network: {
      inbound: false,
      localAddress: "192.168.137.157:25702",
      remoteAddress: "167.86.102.42:30300",
      static: false,
      trusted: false
    },
    protocols: {
      eth: "handshake",
      snap: "handshake"
    }
```


after adding node 3 in node 1

admin.addPeer("enode://8ae6ce43da5011202f67403a1ce92fc2c9b8d80c401b86eff9902ffe18a909f9aef449dd5f5a71c7649c33e6111bee463461ce8f4934f7a8db5ebe354b41efc4@localhost:30305?discport=5877")

```
> admin.addPeer("enode://8ae6ce43da5011202f67403a1ce92fc2c9b8d80c401b86eff9902ffe18a909f9aef449dd5f5a71c7649c33e6111bee463461ce8f4934f7a8db5ebe354b41efc4@localhost:30305?discport=5877")
true
```



```
}, {
    caps: ["eth/66", "eth/67"],
    enode: "enode://ef323f7ce4126fc23743c60a2ad70ad4f1c8111198bad16d2ef0a1effe7e33d2dd8b0b48f3060bfa93cb82ca8d72806e0d27fe98a390d705c47d968f086a8b1e@193.187.129.119:30302",
    id: "830541cba58d8d1bfa19079559e31779101e4fc70b393bcbf47fadd35e87986b",
    name: "Geth/v1.10.20-stable-8f2416a8/linux-amd64/go1.18.1",
    network: {
      inbound: false,
      localAddress: "192.168.137.157:27715",
      remoteAddress: "193.187.129.119:30302",
      static: false,
      trusted: false
    },
    protocols: {
      eth: "handshake"
    }
}, {
    caps: ["eth/65", "eth/66", "eth/67"],
    enode: "enode://f44e0d58d06c88ba1f190d9823a5a6a3d7513c89fcd0e59b1fcaac9195c8a583bcbed1f8a2c0f4ef0671b783c456d8847899bbbb5cf8873faa076a307687fded@78.47.128.136:30312",
    id: "83d0dd5a6755a55427bc253dd89b21cbd087add1592700f480f5e1c5a99727b",
    name: "Geth/v1.1.10/linux-amd64/go1.18.2",
    network: {
      inbound: false,
      localAddress: "192.168.137.157:27697",
      remoteAddress: "78.47.128.136:30312",
      static: false,
      trusted: false
    },
    protocols: {
      eth: "handshake"
    }
}]
```


after adding node 3 in node 1

admin.addPeer("enode://00ecd6574570db5e1eecb75c641fadf793ee76a53b43d732dbcbbfa0ae864dac59dff430f8e661f2229e9b2d6d8823c1167419a58e0b1ea240439065925b9a51@localhost:30305?discport=5877")

```
}, {
    caps: ["eth/65", "eth/66", "snap/1"],
    enode: "enode://5733cbe45e96bdfdb537e931406f4f02e273361941edb74f54225f8b1aad8f7d99912991a1ffd13b0d1faf66fc34a709e3b1f8f0b2a2ddcb007606e45a11baa7@13.37.113.217:30300",
    id: "4d6b3c95aa2ce17c9f45ef1a6d1eae276f09e94689b90bcf344e637a797b6d08",
    name: "Geth/xt_smartchain_node8/v1.1.2-bed14ec4-20211123/linux-amd64/go1.17.3",
    network: {
      inbound: false,
      localAddress: "192.168.137.157:30527",
      remoteAddress: "13.37.113.217:30300",
      static: false,
      trusted: false
    },
    protocols: {
      eth: "handshake",
      snap: "handshake"
    }
}]
```


Now all the three peers are added

Which are

[{
caps: ["diff/1", "eth/65", "eth/66", "eth/67", "snap/1"],
enode:
"enode://1082ed929b9d6cfcd0ce4a0c1865dd5021e0edfc873360d2fac3a61cf73953c014ca2e79bc578162840b5f2387aa03e46597002a8fcc271da560a39287e69719@18.182.65.149:30311",
id: "48331b1c365ec7c9cda2ffa6f9f3eefdcc06cbb7b5085dfae46b10cf3d2526c0",
name: "Geth/v1.1.11/linux-amd64/go1.13.8",
network: {
inbound: false,
localAddress: "192.168.137.157:30521",
remoteAddress: "18.182.65.149:30311",
static: false,

```
    trusted: false
   },
   protocols: {
    eth: "handshake",
    snap: "handshake"
   }
}, {
   caps: ["diff/1", "eth/65", "eth/66", "eth/67", "les/2", "les/3", "les/4", "snap/1"],
   enode:
"enode://47de9d7808f339b55c5d958ba3a644c2423731de269fa926d8c78eb0b864e4c78734314dd
1fc6439a99f1d4c0dab48d57f8a0bfa4b82ffcbf6547f880c41d079@52.202.229.96:30311",
   id: "49a5130511f5365731f188ae485f8e3d93f7b3c314a4de2ec863926fdb7a2cd8",
   name: "Geth/v1.1.11-f940fb8e/linux-amd64/go1.18.3",
   network: {
    inbound: false,
    localAddress: "192.168.137.157:30537",
    remoteAddress: "52.202.229.96:30311",
    static: false,
    trusted: false
   },
   protocols: {
    eth: "handshake",
    snap: "handshake"
   }
}, {
   caps: ["eth/65", "eth/66", "snap/1"],
   enode:
"enode://5733cbe45e96bdfdb537e931406f4f02e273361941edb74f54225f8b1aad8f7d99912991a1f
fd13b0d1faf66fc34a709e3b1f8f0b2a2ddcb007606e45a11baa7@13.37.113.217:30300",
   id: "4d6b3c95aa2ce17c9f45ef1a6d1eae276f09e94689b90bcf344e637a797b6d08",
   name: "Geth/xt_smartchain_node8/v1.1.2-bed14ec4-20211123/linux-amd64/go1.17.3",
   network: {
    inbound: false,
    localAddress: "192.168.137.157:30527",
    remoteAddress: "13.37.113.217:30300",

    static: false,

    trusted: false

   },

   protocols: {

    eth: "handshake",

    snap: "handshake"

   }

}]
```


Here we have built a Ethereum network with 4 peers

B. Build Public Ethereum Blockchain using proof of work (POW) consensus mechanism.

The idea for Proof of Work(PoW) was first published in 1993 by Cynthia Dwork and Moni Naor and was later applied by Satoshi Nakamoto in the Bitcoin paper in 2008. Proof of Work consensus is the mechanism of choice for the majority of cryptocurrencies currently in circulation. The term "proof of work" was first used by Markus Jakobsson and Ari Juels in a publication in 1999.

A solution that is difficult to find but is easy to verify.

The purpose of a consensus mechanism is to bring all the nodes in agreement, that is, trust one another, in an environment where the nodes don't trust each other.

All the transactions in the new block are then validated and the new block is then added to the blockchain. Note that, the block will get added to the chain which has the longest block height(see blockchain forks to understand how multiple chains can exist at a point of time). Miners(special computers on the network) perform computation work in solving a complex mathematical problem to add the block to the network, hence named, Proof-of-Work. With time, the mathematical problem becomes more complex.

How PoW works?

An edX course describes on Blockchain PoW as:

"The Proof of Work consensus algorithm involves solving a computational challenging puzzle in order to create new blocks in the Bitcoin blockchain. Colloquially, the process is known as 'mining', and the nodes in the network that engage in mining are known as 'miners'. The incentive for mining transactions lies in economic payoffs, where competing miners are rewarded with 12.5 bitcoins(at the time of writing this article; this reward will get reduced by half its current value with time) and a small transaction fee."

The process of verifying the transactions in the block to be added, organizing these transactions in a chronological order in the block and announcing the newly mined block to the entire network does not take much energy and time. The energy consuming part is solving the 'hard mathematical problem' to link the new block to the last block in the valid blockchain.

When a miner finally finds the right solution, the node broadcasts it to the whole network at the same time, receiving a cryptocurrency prize (the reward) provided by the PoW protocol. At the time of writing this article, mining a block in the bitcoin network gives the winning miner 12.5 bitcoins. The amount of bitcoins won halves every four years or so(thats how the bitcoin network is designed). So, the next deduction in the amount of bitcoin is due at around 2020-21(with the current rate and growth).

With more miners comes the inevitability of the time it takes to mine the new block getting shorter. This means that the new blocks are found faster. In order to consistently find 1 block every 10 minutes (That is the amount of time that the bitcoin developers think is necessary for a steady and diminishing flow of new coins until the maximum number of 21 million is reached (expected some time with the current rate in around 2140)), the Bitcoin network regularly changes the difficulty level of mining a new block.

To see more about what a block contains, visit Blockchain Explorer through Google for more interesting surfing

The fact that Block GFG1 is connected to Block GFG2 through its hash number is important. The significance lies in the fact that this 'hash number' connects new block to the last block in the valid blockchain. If, on the other hand, the Block GFG1 Hash number on Block GFG2 had a different hash number than Block GFG1 they would not match up, and Block GFG2 would not be verified.

First block in the blockchain is called the Genesis Block and has no Prev Block Hash value.

Changing a block (which can only be done by making a new block containing the same predecessor) requires regenerating all successors and redoing the work they contain (amounting to calculating the entire chain of 'hard mathematical problems') which is practically impossible. This protects the blockchain from tampering.

Bitcoin's Proof-of-Work system:

Bitcoin uses the Hash cash Proof of Work system as the mining basis. The 'hard mathematical problem' can be written in an abstract way like below :

Given data A, find a number x such as that the hash of x appended to A results is a number less than B.

The miners bundle up a group of transactions into a block and try to mine. To mine it, a hard mathematical problem has to be solved. This problem is called the proof of work problem which has to be solved to show that the miner has done some work in finding out the solution to the problem and hence the mined block must be valid.
The answer to the problem needs to be a lower number than the hash of the block for it to be accepted, known as the 'target hash'.A target hash is a number that the header of a hashed block must be equal to or less than for a new block, along with the reward, to be awarded to a miner. The lower a target is, the more difficult it is to generate a block.

A miner continues testing different unique values (known as nonce(s)) until a suitable one is produced.
The miner who manages to solve the problem gets the bitcoin reward and adds the block into the blockchain by broadcasting that the block has been mined.

The target hash adjusts once every 2016 blocks or approximately once every 2 weeks. All the miners immediately stop work on the said block and start mining the next block.

Common cryptographic protocols used in Proof of Work systems:

The most widely used proof-of-work consensus is based on SHA-256 and was introduced as a part of Bitcoin. Others include Scrypt, SHA-3, scrypt-jane, scrypt-n, etc.

Features of Proof of Work system:

There are mainly two features that have contributed to the wide popularity of this consensus protocol and they are:

It is hard to find a solution for the mathematical problem
It is easy to verify the correctness of that solution

Main issues with the Proof-of-Work consensus:

The Proof-of-Work consensus mechanism has some issues which are as follows:

The 51% risk: If a controlling entity owns 51% or more than 51% of nodes in the network, the entity can corrupt the blockchain by gaining the majority of the network.
Time consuming: Miners have to check over many nonce values to find the right solution to the puzzle that must be solved to mine the block, which is a time consuming process.
Resource consumption: Miners consume high amounts of computing power in order to find the solution to the hard mathematical puzzle. It leads to a waste of precious

resources(money, energy, space, hardware). It is expected that the 0.3% of the world's electricity will be spent to verify transactions by the end of 2018.
Transaction confirmation takes about 10–60 minutes. So, it is not an instantaneous transaction; because it takes some time to mine the transaction and add it to the blockchain thus committing the transaction.

Cryptocurrencies using PoW:

Litecoin
Ethereum
Monero coin
Dogecoin

C. Setup Proof of Authority (POA) Consensus on Ethereum Blockchain.

In blockchain platforms, consensus mechanisms can be divided into permissionless (eg., Ethereum, Bitcoin) and permissioned (eg Hyperledger, Ethereum Private). Unlike permissionless blockchain where anyone can become node, in permissioned blockchain all nodes are pre-selected. This allows to use consensus types with high scalability and bandwidth. One of these consensus types is Proof-of-Authority (PoA) consensus which provides high performance and fault tolerance. Term was proposed in 2017 by co-founder of Ethereum and Parity Technologies Gavin Wood.

Working of PoA :

In PoA, rights to generate new blocks are awarded to nodes that have proven their authority to do so. These nodes are referred to as "Validators" and they run software allowing them to put transactions in blocks. Process is automated and does not require validators to be constantly monitoring their computers but does require maintaining the computer uncompromised. PoA is suited for both private networks and public networks, like POA Network, where trust is distributed.
PoA consensus algorithm leverages value of identities, which means that block validators are not staking coins but their own reputation instead. PoA is secured by trust on the identities selected.

PoA consensus and common attacks :

Distributed Denial-of-service attacks(DDos) :

A Distributed Denial of Service (DDoS) attack is an attempt to make an online service unavailable by overwhelming it with traffic from multiple sources. An attacker sends large

number of transactions and blocks to targeted network node in an attempt to disrupt its operation and make it unavailable.

PoA mechanism makes it possible to defend against this attack because network nodes are pre-authenticated, block generation rights can be granted only to nodes that can withstand DoS attacks.

51% attack :

In PoA consensus, 51% attack requires an attacker to obtain control over 51% of network nodes. This is different from 51% attack for the Proof-of-Work consensus types where an attacker needs to obtain 51% of network computational power. Obtaining control of the nodes in permissioned blockchain network is much harder than obtaining computational power.
With PoA, individuals earn right to become validators, so there is an incentive to retain position that they have gained. Validators are incentivized with reputation which lets them retain their authority as a node. PoA only allows non-consecutive block approval from any one validator, meaning that the risk of serious damage is centralized to the authority node.

Conditions for PoA consensus :

PoA consensus may vary according to different implementation but generally they are applied through the following conditions :
Validators need to confirm their real identities.
A candidate must be willing to invest money and put his reputation at stake. A tough process reduces risks of selecting questionable validators and incentivize long-term commitment to the blockchain.
Method for selecting validators must be equal to all candidates.
Identity of validators must be verified to maintain integrity of blockchain. Some sort of process should be their to select honest validators.

Advantages of PoA consensus :

High risk tolerance as long as 51% of the nodes are not acting maliciously.
Interval of time at which new blocks are generated is predictable. For PoW and PoS consensuses, this time varies.
High transaction rate.
Far more sustainable than algorithms like Proof of Work which require computational power.

Limitations :

PoA is not decentralized but is just an effort to make centralized systems more efficient.
PoA validators are visible to anyone. Knowing validators identities could potentially lead to third-party manipulation.

Application of PoA consensus :

PoA consensus algorithm may be applied in variety of scenarios and is deemed great option for logistical applications such as supply chains.
Proof of Authority model enables companies to maintain their privacy while availing benefits of blockchain technology. Microsoft Azure is another example where PoA is being implemented. Azure platform provides solutions for private networks, with system that does not require native currency like ether 'gas' on Ethereum, since there is no need for mining. Azure nodes are pre-selected.

D. Self-study to build Ethereum 2.0 network using POS.

What is Ethereum 2.0?

Ethereum 2.0 is a planned sequence of changes to the Ethereum protocol intended to improve its overall performance.
If the creation of Ethereum ushered in a new era of experimentation in financial applications, Ethereum 2.0 seeks to continue this progress by altering the network's design. This includes migrating to a new consensus mechanism that will govern how transactions are approved by the network and its cryptocurrency is distributed.
But before we dive into its new iteration, it is important to understand how Ethereum, one of the most ambitious cryptocurrency projects to date, operates.
Ethereum was built to be a kind of operating system for hosting any number of custom assets and programs, called decentralized applications (dapps).
To create dapps, developers write programs, called smart contracts, and deploy this code to the Ethereum blockchain. These dapps are essentially large constructions of smart contracts that can be set in motion if and when specific outcomes are met.
Today, Ethereum uses proof-of-work mining (in which computers burn energy to solve puzzles needed to create blocks) to power its blockchain. Miners batch transactions into new blocks roughly every 12 seconds.
Using this design, the Ethereum blockchain currently processes 12 transactions per second over its distributed network, a figure that may prove higher once Ethereum 2.0 is enacted.

How does Ethereum 2.0 Work?

Ethereum 2.0 will bring three major upgrades to the Ethereum blockchain.

Ethereum Proof-of-Stake (PoS)

Under the Ethereum PoS model, users, called validator nodes, can lock ETH cryptocurrency in a smart contract, which then would earn rewards for solving computations needed to add new blocks to the blockchain.

The minimum amount of cryptocurrency required to become a validator node is 32 ETH, which allows them to support the network by verifying transactions, storing data and adding new blocks to the blockchain.

Of note, Kraken users can stake less than 32 ETH on its platform, contribute to its network and earn a portion of the staking rewards. For more information on Ethereum staking, please visit our crypto staking page.

Ethereum Sharding

Sharding refers to the process by which Ethereum's infrastructure will be split into several interconnected pieces to support more transactions.

Each piece, or shard, acts as its own blockchain and allows validators assigned to them to store data, process transactions and add new blocks to their specific shard chain.

This means that shard nodes contain a specific subset of the Ethereum blockchain, and are no longer required to store Ethereum's entire history.

ETH 2.0 Beacon Chain

The Beacon Chain is the foundational chain of Ethereum 2.0 and plays an important managerial role by coordinating validator nodes in the network and keeping shards secure and in sync.

After a user stakes 32 ETH, the Beacon Chain randomly assigns validator nodes to specific shard chains, and then ensures that the nodes act in good faith.

For example, validator nodes can lose ETH for malicious actions, going offline or failing to validate transactions.

Each shard will be linked to the Beacon Chain, where validators will take turns creating and validating new shard blocks before adding them to the rest of the network.

Who created Ethereum?

Ethereum was the brainchild of then 20-year-old Russian-Canadian Vitalik Buterin.

Ethereum Roadmap to 2.0

Developers are currently working on new Ethereum 2.0 features, to be rolled out during phases.

Phase 0

This phase saw the launch of the beacon chain, enabling ETH holders to become validator nodes and stake their ETH to earn additional income.

For the beacon chain to launch, 16,384 validators needed to stake a minimum of 32 ETH on the beacon chain, a milestone that occurred on November 24, 2020. The code was then deployed on December 1, 2020.

The beacon chain has limited functionality during phase 0 since there is no transaction or smart contract support.

Phase 1

This phase would split the Ethereum blockchain into 64 shard chains (a number likely to increase once the full version of Ethereum 2.0 is released).

Further, phase 1 will extend the beacon chain's PoS consensus across all shards, allowing validators to create blocks on their specific chain.

While shard chains will still not be able to process transactions or smart contract support, they will be able to store Ethereum data.

Phase 1.5

In Phase 1.5, Ethereum will be merged into a single, unified network.

This phase will see the addition of the current Ethereum chain, which contains the full Ethereum history, as one of the shards in the new ethereum 2.0 system and fully transition Ethereum from proof-of-work to proof-of-stake.

This is the final phase of Ethereum's roadmap to 2.0, where the upgrade will become the official Ethereum network.

Phase 2 involves adding functionality to shard chains, enabling them to process transactions and execute smart contracts so that dapp developers can deploy their applications on shards.

Practical 4

Aim :

Study and implements basic of Solidity programming : Syntax, Variables, Functions, mapping, access modifiers using Remix online IDE.

Solution:

What is Solidity ?

- Solidity is an object-oriented, high-level language for implementing smart contracts. It is designed to target Ethereum Virtual Machine(EVM)
- It is statically typed, supports inheritance, libraries and complex user-defined types among other features.

Building in Solidity

Solidity's code is encapsulated in contracts. A contract is the fundamental building block of Ethereum applications — all variables and functions belong to a contract, and this will be the starting point of all your projects.

Initializing smart contracts

An empty contract named HelloWorld would look like this:

```
// Define the compiler version you would be using
pragma solidity ^0.8.10;

// Start by creating a contract named HelloWorld
contract HelloWorld {

}
```

- **Version Pragma** - All solidity source code should start with a "version pragma" — a declaration of the version of the Solidity compiler this code should use. This is to prevent issues with future compiler versions potentially introducing changes that would break your code.

Variables and types

There are 3 types of variables in Solidity

- **Local**
    - Declared inside a function and are not stored on blockchain

- **State**
  - Declared outside a function to maintain the state of the smart contract
  - Stored on the blockchain
- **Global**
  - Provide information about the blockchain. They are injected by the Ethereum Virtual Machine during runtime.
  - Includes things like transaction sender, block timestamp, block hash, etc.
  - Examples of global variables

The scope of variables is defined by where they are declared, not their value. Setting a local variable's value to a global variable does not make it a global variable, as it is still only accessible within it's scope.

```
// Define the compiler version you would be using
pragma solidity ^0.8.10;

// Start by creating a contract named Variables
contract Variables {
    /*
      ******** State variables **********
    */
    /*
    uint stands for unsigned integer, meaning non negative integers
    different sizes are available. Eg
       - uint8   ranges from 0 to 2 ** 8 - 1
       - uint256 ranges from 0 to 2 ** 256 - 1
    `public` means that the variable can be accessed internally
     by the contract and can also be read by the external world
    */
    uint8 public u8 = 10;
    uint public u256 = 600;
    uint public u = 1230; // uint is an alias for uint256

    /*
    Negative numbers are allowed for int types. Eg
    - int256 ranges from -2 ** 255 to 2 ** 255 - 1
    */
    int public i = -123; // int is same as int256

    // address stands for an ethereum address
    address public addr = 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c;

    // bool stands for boolean
    bool public defaultBoo1 = false;

    // Default values
    // Unassigned variables have a default value in Solidity
    bool public defaultBoo2; // false
    uint public defaultUint; // 0
    int public defaultInt; // 0
```

```
    address public defaultAddr; // 0x0000000000000000000000000000000000000000

    function doSomething() public {
        /*
        ******** Local variable **********
        */
        uint ui = 456;

        /*
        ******** Global variables **********
        */

        /*
            block.timestamp tells us whats the timestamp for the current block
            msg.sender tells us which address called the doSomething function
        */
        uint timestamp = block.timestamp; // Current block timestamp
        address sender = msg.sender; // address of the caller
    }
}
```

Functions, Loops and If/Else

```
// Define the compiler version you would be using
pragma solidity ^0.8.10;

// Start by creating a contract named Conditions
contract Conditions {
    // State variable to store a number
    uint public num;

    /*
        Name of the function is set.
        It takes in a uint and sets the state variable num.
        It is a declared as a public function meaning
        it can be called from within the contract and also externally.
    */
    function set(uint _num) public {
        num = _num;
    }

    /*
        Name of the function is get.
        It returns the value of num.
        It is declared as a view function meaning
        that the function doesnt change the state of any variable.
        view functions in solidity do not require gas.
```

```
    */
    function get() public view returns (uint) {
        return num;
    }

    /*
        Name of the function is foo.
        It takes in  uint and returns an uint.
        It compares the value of x using if/else
    */
    function foo(uint x) public returns (uint) {
        if (x < 10) {
            return 0;
        } else if (x < 20) {
            return 1;
        } else {
            return 2;
        }
    }

    /*
        Name of the function is loop.
        It runs a loop till 10
    */
    function loop() public {
        // for loop
        for (uint i = 0; i < 10; i++) {
            if (i == 3) {
                // Skip to next iteration with continue
                continue;
            }
            if (i == 5) {
                // Exit loop with break
                break;
            }
        }
    }
}
```

Well, there are two ways in which we can pass an argument to a Solidity function:

- **By value** - which means that the Solidity compiler creates a new copy of the parameter's value and passes it to your function. This allows your function to modify the value without worrying that the value of the initial parameter gets changed.
- **By reference -** which means that your function is called with a... reference to the original variable. Thus, if your function changes the value of the variable it receives, the value of the original variable gets changed.

Mappings

- Mappings in Solidity act like hashmaps or dictionaries in other programming languages. They are used to store the data in key-value pairs.
- Mappings are created with the syntax mapping (keyType => valueType)

```
// For a financial app, storing a uint that holds the user's account balance:
mapping (address => uint) public accountBalance;
// Or could be used to store / lookup usernames based on userId
mapping (uint => string) userIdToName;
```

In the first example, the key is an address and the value is a uint, and in the second example the key is a uint and the value a string.

Visibility of Function and Use of view , pure , constant

Visibility types:

- **Internal** - can be accessed in the contract in which it is defined and in its subclasses.
- **External** – can be accessed from other contracts using transactions. This type of function can be accessed using the this.functionName() notation and not just functionName().
- **Private** – cannot be access from outside and cannot be access from inhert and can only be called from the contract in which it is defined
- **Public** – it is public ,can be access from outside ,can be called from inside and from inherit contract also.

If anything is not defined than it will consider as private by default

Example

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 <0.9.0;

contract A{

unit private a;

function f1(unit x, unit z) internal returns (unit){
return x + z; }
```

```
function f2(unit y) private  returns(uint b){
return 6*y ; }
}

contract B {

function f3() public{

A crtA = new A();
unit var = crtA.f2(89);  // compilation error
}
}

contract C is A {

   function f4() public {
      A contA = new A();
      uint var2 = f1(74, 51);
 } }
```

Access modifier

- **View**: Functions which do not change any state values
- **Pure**: Functions which do not change any state values, but also don't read any state values
- **Constant** – same as view but it is used in older version of solidity

  If anything is not defined than we can modify variables value

Example

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;

contract ViewAndPure {
   // Declare a state variable
   uint public x = 1;

   // Promise not to modify the state (but can read state)
   function addToX(uint y) public view returns (uint) {
      return x + y;
   }

   // Promise not to modify or read from state
   function add(uint i, uint j) public pure returns (uint) {
```

```
        return i + j;
    }
}
```

Practical 5

Aim :

Design and develop smart contract using solidity for Product Market-Placeover REMIX IDE.

Solution:

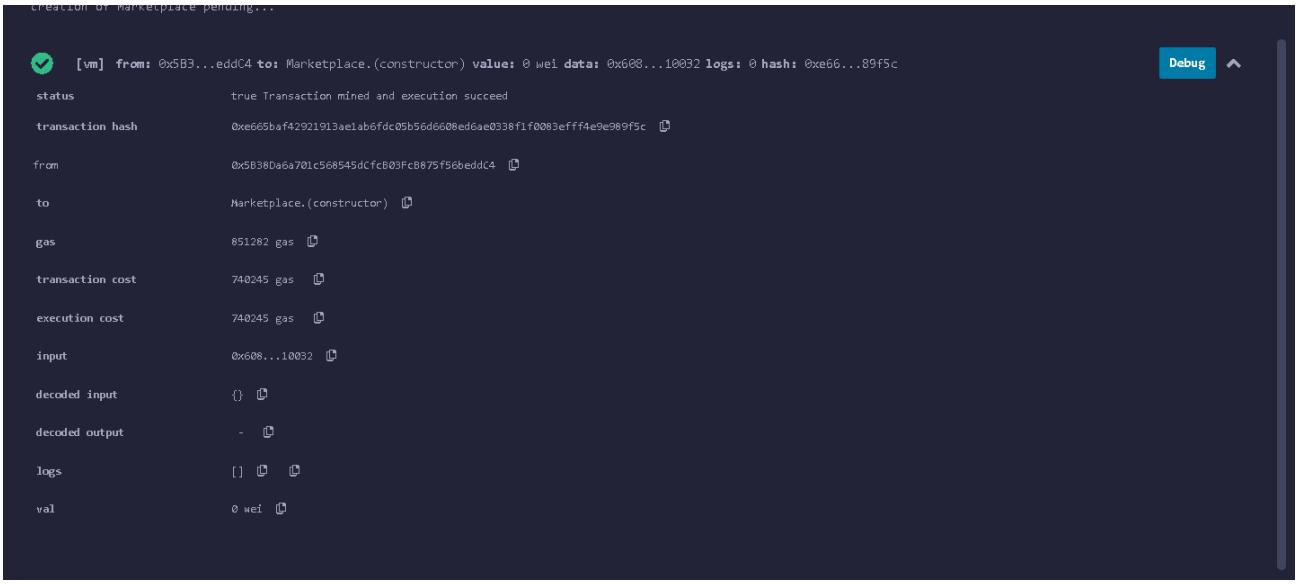
Code:

```
pragma solidity ^0.5.0;

contract Marketplace {
  string public name;
  uint public productCount = 0;
  mapping(uint => Product) public products;

  struct Product {
    uint id;
    string name;
    uint price;
    address payable owner;
    bool purchased;
  }

  event ProductCreated(
    uint id,
    string name,
    uint price,
    address payable owner,
    bool purchased
  );

  event ProductPurchased(
    uint id,
    string name,
    uint price,
    address payable owner,
    bool purchased
  );

  constructor() public {
```

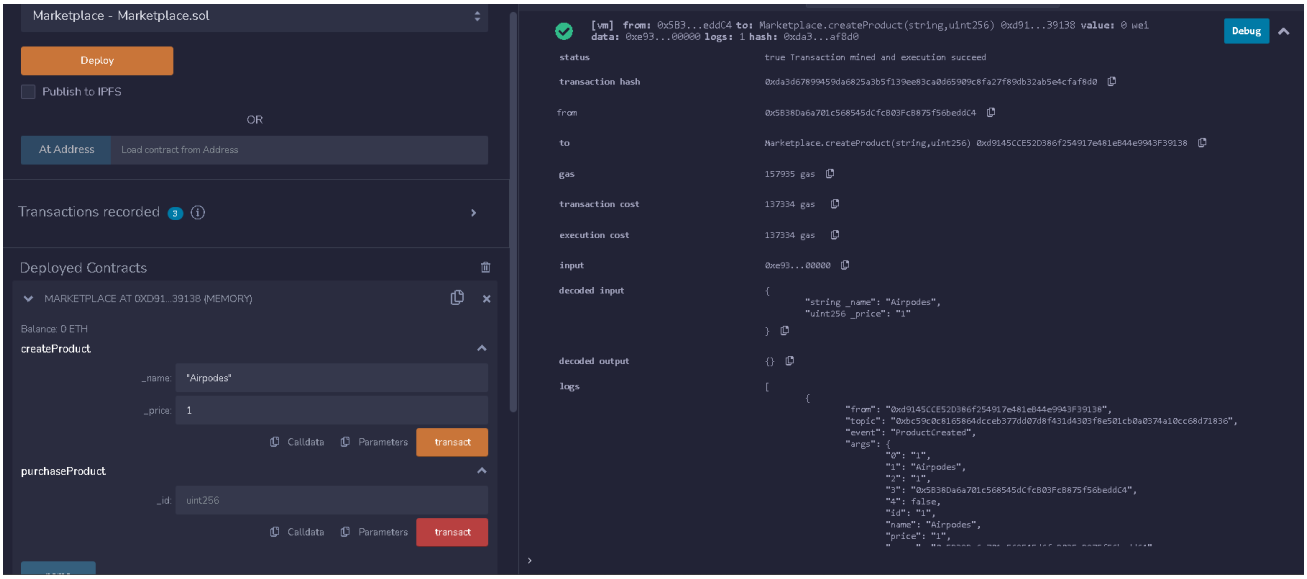
```
    name = "Dapp University Marketplace";
  }

  function createProduct(string memory _name, uint _price) public {
    // Require a valid name
    require(bytes(_name).length > 0);
    // Require a valid price
    require(_price > 0);
    // Increment product count
    productCount ++;
    // Create the product
    products[productCount] = Product(productCount, _name, _price, msg.sender, false);
    // Trigger an event
    emit ProductCreated(productCount, _name, _price, msg.sender, false);
  }

  function purchaseProduct(uint _id) public payable {
    // Fetch the product
    Product memory _product = products[_id];
    // Fetch the owner
    address payable _seller = _product.owner;
    // Make sure the product has a valid id
    require(_product.id > 0 && _product.id <= productCount);
    // Require that there is enough Ether in the transaction
    require(msg.value >= _product.price);
    // Require that the product has not been purchased already
    require(!_product.purchased);
    // Require that the buyer is not the seller
    require(_seller != msg.sender);
    // Transfer ownership to the buyer
    _product.owner = msg.sender;
    // Mark as purchased
    _product.purchased = true;
    // Update the product
    products[_id] = _product;
    // Pay the seller by sending them Ether
    address(_seller).transfer(msg.value);
    // Trigger an event
    emit ProductPurchased(productCount, _product.name, _product.price, msg.sender, true);
  }
}
```

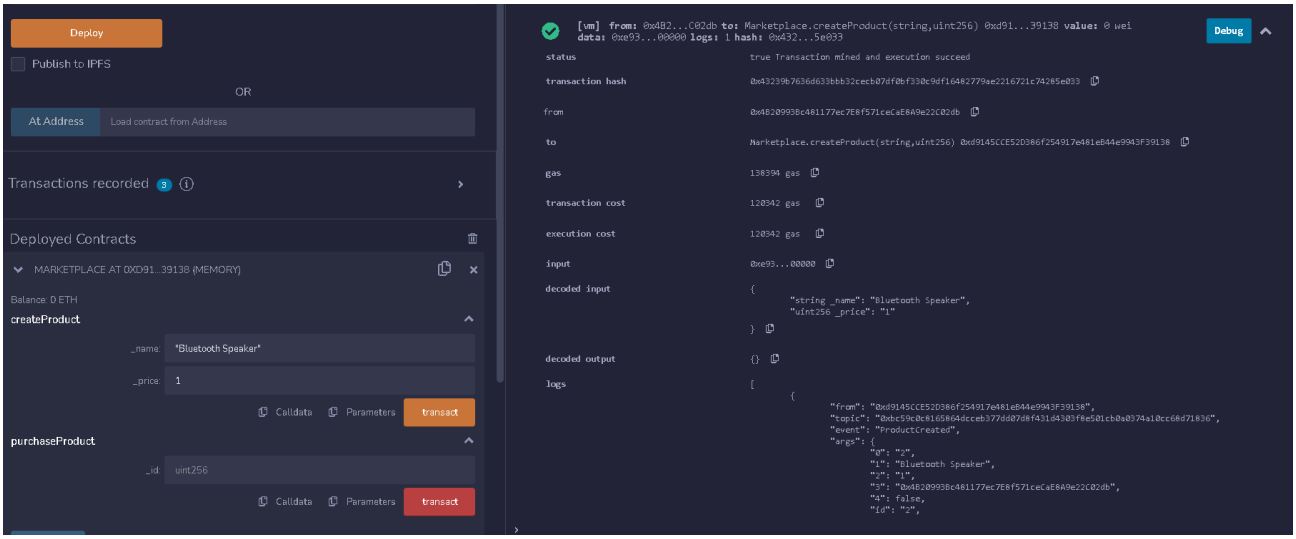
Output :

Deployment of smart contract

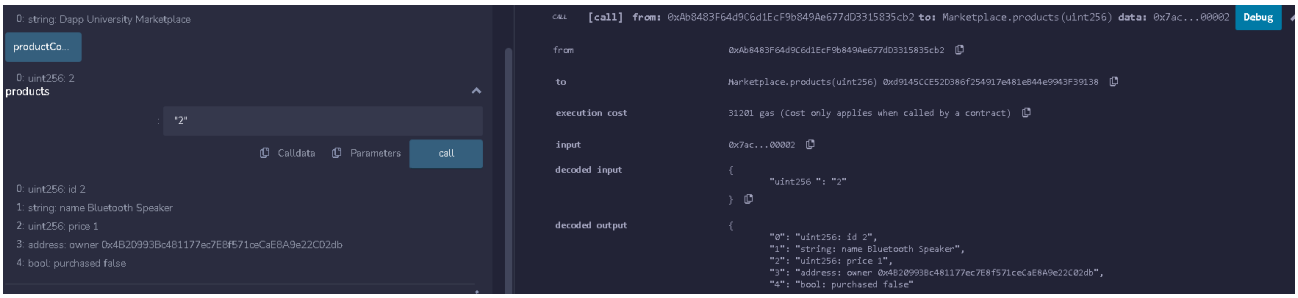


Create products

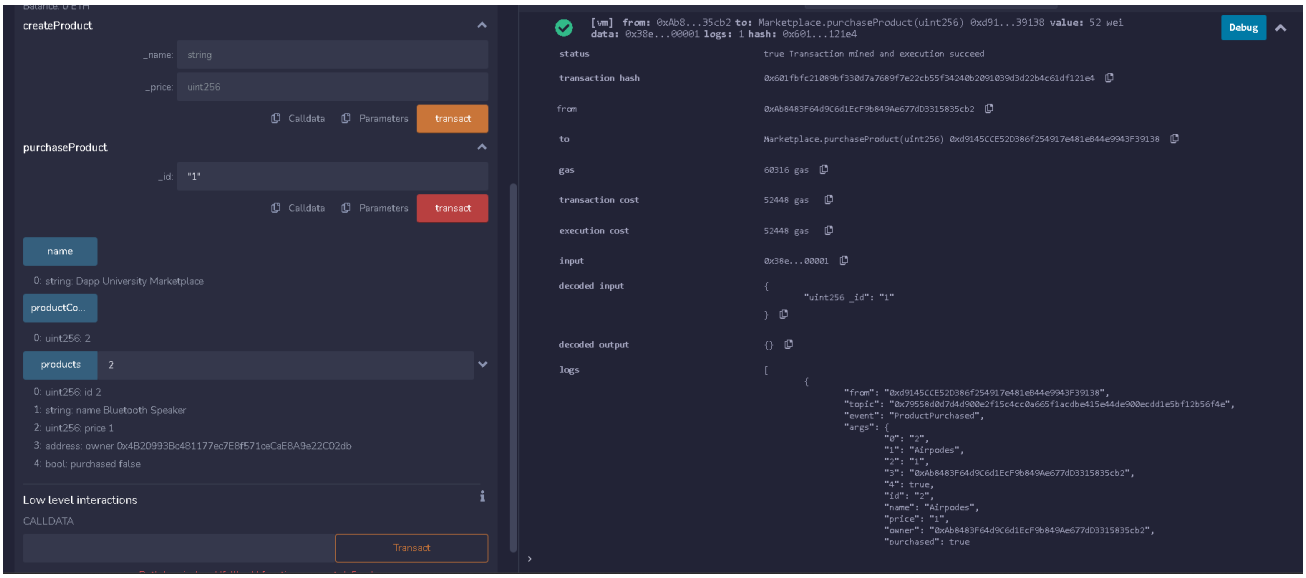




Get Products Info



Purchase Product 1



Practical 6

Aim :

Design and develop smart contract using solidity for Supply Chain Management using Truffle framework.

Solution:

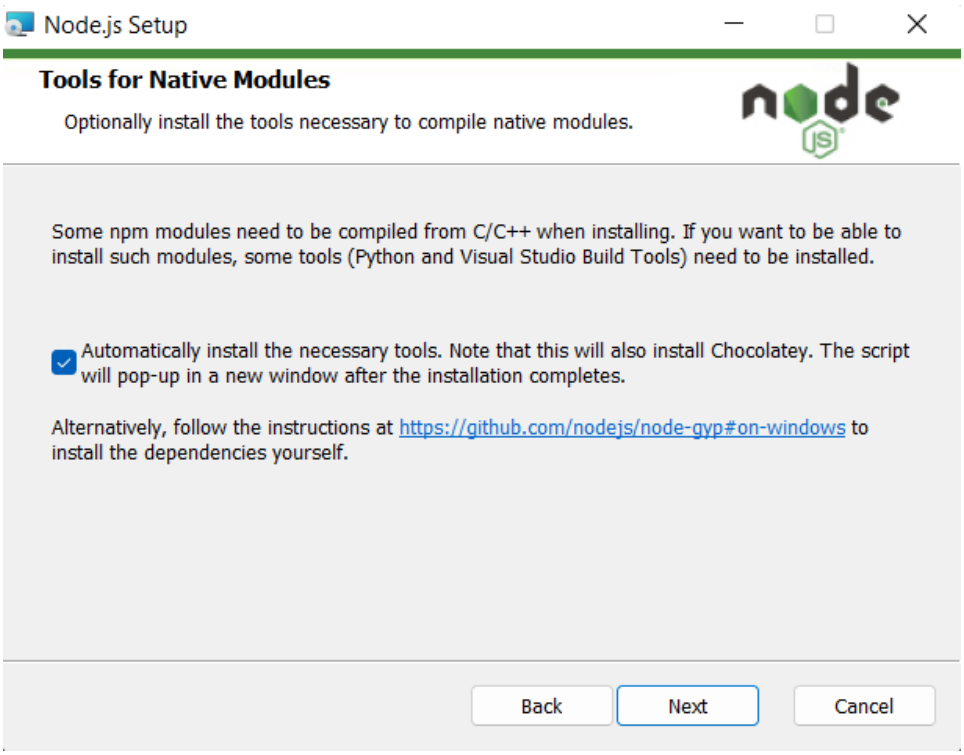
Perform the following Steps :

1. **Install Truffle**
2. **Create a project**
3. **Create Smart Contract**
4. **Compile Smart contract**

Installing Truffle:

Truffle recommends using the installer available from the Node.js site.

Ensure you select Automatically install the necessary tools... during the install to install the required Visual Studio build tools, Python, and Chocolately package manager.



Install Truffle

In a terminal, use NPM to install Truffle:

- npm install -g truffle

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM> npm i truffle -g
npm WARN deprecated mkdirp-promise@5.0.1: This package is broken and no longer maintained. 'mkdirp' itself supports promises now, please switch to that.
npm WARN deprecated @types/keyv@4.2.0: This is a stub types definition. keyv provides its own type definitions, so you do not need this installed.
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated multicodec@1.0.4: This module has been superseded by the multiformats module
npm WARN deprecated uuid@2.0.1: Please upgrade  to version 7 or higher.  Older versions may use Math.random() in certain circumstances, which is known to be
problematic.  See https://v8.dev/blog/math-random for details.
npm WARN deprecated uuid@3.4.0: Please upgrade  to version 7 or higher.  Older versions may use Math.random() in certain circumstances, which is known to be
problematic.  See https://v8.dev/blog/math-random for details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated multibase@0.6.1: This module has been superseded by the multiformats module
npm WARN deprecated multibase@0.7.0: This module has been superseded by the multiformats module

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```

To confirm that Truffle was installed correctly, run:

- truffle version

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM> truffle version
Truffle v5.6.2 (core: 5.6.2)
Ganache v7.4.4
Solidity v0.5.16 (solc-js)
Node v16.16.0
Web3.js v1.7.4
```

Creating a project

Create a new directory for your Truffle project:

- mkdir SCM
- cd SCM

create a bare project template using

- truffle init

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM> mkdir SCM


    Directory: C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        22-10-2022     13:07                SCM


PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM> cd SCM
```

Create Smart Contract

Create a smart contract under
SCM/contracts/supplyChain.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.21 <0.6.0;

contract supplyChain {
  uint32 public product_id = 0; // Product ID
  uint32 public participant_id = 0; // Participant ID
  uint32 public owner_id = 0; // Ownership ID
  struct product {
    string modelNumber;
    string partNumber;
    string serialNumber;
    address productOwner;
    uint32 cost;
    uint32 mfgTimeStamp;
}
mapping(uint32 => product) public products;
  struct participant {
```

```
    string userName;
    string password;
    string participantType;
    address participantAddress;
}
mapping(uint32 => participant) public participants;

    struct ownership {
    uint32 productId;
    uint32 ownerId;
    uint32 trxTimeStamp;
    address productOwner;
}
mapping(uint32 => ownership) public ownerships; // ownerships by ownership ID (owner_id)
mapping(uint32 => uint32[]) public productTrack; // ownerships by Product ID (product_id) /
Movement //track for a product

event TransferOwnership(uint32 productId);

function addParticipant(string memory _name, string memory _pass, address _pAdd, string memory
_pType) public returns
(uint32){
    uint32 userId = participant_id++;
    participants[userId].userName = _name;
    participants[userId].password = _pass;
    participants[userId].participantAddress = _pAdd;
    participants[userId].participantType = _pType;
    return userId;
}
function getParticipant(uint32 _participant_id) public view returns (string memory,address,string
memory) {
    return (participants[_participant_id].userName,
    participants[_participant_id].participantAddress,
    participants[_participant_id].participantType);
}

function addProduct(uint32 _ownerId,string memory _modelNumber,string memory
_partNumber,string memory _serialNumber,uint32 _productCost) public returns (uint32) {
    if(keccak256(abi.encodePacked(participants[_ownerId].participantType)) ==
keccak256("Manufacturer")) {
        uint32 productId = product_id++;
        products[productId].modelNumber = _modelNumber;
        products[productId].partNumber = _partNumber;
        products[productId].serialNumber = _serialNumber;
        products[productId].cost = _productCost;
```

```
        products[productId].productOwner = participants[_ownerId].participantAddress;
        products[productId].mfgTimeStamp = uint32(now);
        return productId;
    }
    return 0;
    }

    modifier onlyOwner(uint32 _productId) {
    require(msg.sender == products[_productId].productOwner,"");
    _;
    }

    function getProduct(uint32 _productId) public view returns (string memory,string memory,string
    memory,uint32,address,uint32){
    return (products[_productId].modelNumber,
    products[_productId].partNumber,
    products[_productId].serialNumber,
    products[_productId].cost,
    products[_productId].productOwner,
    products[_productId].mfgTimeStamp);
    }

    function newOwner(uint32 _user1Id,uint32 _user2Id, uint32 _prodId) onlyOwner(_prodId) public
    returns (bool) {
    participant memory p1 = participants[_user1Id];
    participant memory p2 = participants[_user2Id];
    uint32 ownership_id = owner_id++;

    if(keccak256(abi.encodePacked(p1.participantType)) == keccak256("Manufacturer")&&
    keccak256(abi.encodePacked(p2.participantType))==keccak256("Supplier")){
        ownerships[ownership_id].productId = _prodId;
        ownerships[ownership_id].productOwner = p2.participantAddress;
        ownerships[ownership_id].ownerId = _user2Id;
        ownerships[ownership_id].trxTimeStamp = uint32(now);
        products[_prodId].productOwner = p2.participantAddress;
        productTrack[_prodId].push(ownership_id);
        emit TransferOwnership(_prodId);
        return (true);
    }
    else if(keccak256(abi.encodePacked(p1.participantType)) == keccak256("Supplier") &&
        keccak256(abi.encodePacked(p2.participantType))==keccak256("Supplier")){
        ownerships[ownership_id].productId = _prodId;
        ownerships[ownership_id].productOwner = p2.participantAddress;
        ownerships[ownership_id].ownerId = _user2Id;
        ownerships[ownership_id].trxTimeStamp = uint32(now);
```

```
        products[_prodId].productOwner = p2.participantAddress;
        productTrack[_prodId].push(ownership_id);
        emit TransferOwnership(_prodId);
        return (true);
    }
    else if(keccak256(abi.encodePacked(p1.participantType)) == keccak256("Supplier") &&
        keccak256(abi.encodePacked(p2.participantType))==keccak256("Consumer")){
        ownerships[ownership_id].productId = _prodId;
        ownerships[ownership_id].productOwner = p2.participantAddress;
        ownerships[ownership_id].ownerId = _user2Id;
        ownerships[ownership_id].trxTimeStamp = uint32(now);
        products[_prodId].productOwner = p2.participantAddress;
        productTrack[_prodId].push(ownership_id);
        emit TransferOwnership(_prodId);
        return (true);
    }
    return (false);
}

function getProvenance(uint32 _prodId) external view returns (uint32[] memory) {
    return productTrack[_prodId];
}

function getOwnership(uint32 _regId) public view returns (uint32,uint32,address,uint32) {
    ownership memory r = ownerships[_regId];
    return (r.productId,r.ownerId,r.productOwner,r.trxTimeStamp);
}

function authenticateParticipant(uint32 _uid,string memory _uname,string memory _pass,string
memory _utype) public view returns (bool){
    if(keccak256(abi.encodePacked(participants[_uid].participantType)) ==
keccak256(abi.encodePacked(_utype))) {
        if(keccak256(abi.encodePacked(participants[_uid].userName)) ==
keccak256(abi.encodePacked(_uname))) {
            if(keccak256(abi.encodePacked(participants[_uid].password)) ==
keccak256(abi.encodePacked(_pass))) {
            return (true);
            }
        }
    }
    return (false);
}
}
```

Compiling Smart contract

Let's try compiling our smart contract! Execute the command:

- truffle compile

Now, compilation could fail and show an error similar to this:

- ParserError: Source file requires different compiler version

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM\SCM> truffle compile

Compiling your contracts...
===========================
> Compiling .\contracts\suppluChain.sol

CompileError: ParserError: Source file requires different compiler version (current compiler is 0.8.17+commit.8df45f5f.Emscripten.clang) - note that nightly
builds are considered to be strictly less than the released version
 --> project:/contracts/suppluChain.sol:2:1:
  |
2 | pragma solidity >=0.4.21 <0.6.0;
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


Error: Truffle is currently using solc 0.8.17, but one or more of your contracts specify "pragma solidity >=0.4.21 <0.6.0".
Please update your truffle config or pragma statement(s).
(See https://trufflesuite.com/docs/truffle/reference/configuration#compiler-configuration for information on
configuring Truffle to use a specific solc compiler version.)

Compilation failed. See above.
    at C:\Users\RAj\AppData\Roaming\npm\node_modules\truffle\build\webpack:\packages\compile-solidity\dist\run.js:95:1
    at Generator.next (<anonymous>)
    at fulfilled (C:\Users\RAj\AppData\Roaming\npm\node_modules\truffle\build\webpack:\packages\compile-solidity\dist\run.js:28:43)
Truffle v5.6.2 (core: 5.6.2)
Node v16.16.0
```

Compiling again

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM\SCM> truffle compile

Compiling your contracts...
===========================
> Compiling .\contracts\suppluChain.sol
> Artifacts written to C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM\SCM\build\contracts
> Compiled successfully using:
   - solc: 0.5.1+commit.c8a2cb62.Emscripten.clang
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM\SCM>
```

Practical 7

Aim :

Test the smart contract implemented in practical number 5th and 6th. Write a nodejs code for testing the contract.

Solution:

Prerequest :
Perform the following Steps from Practical-6 :

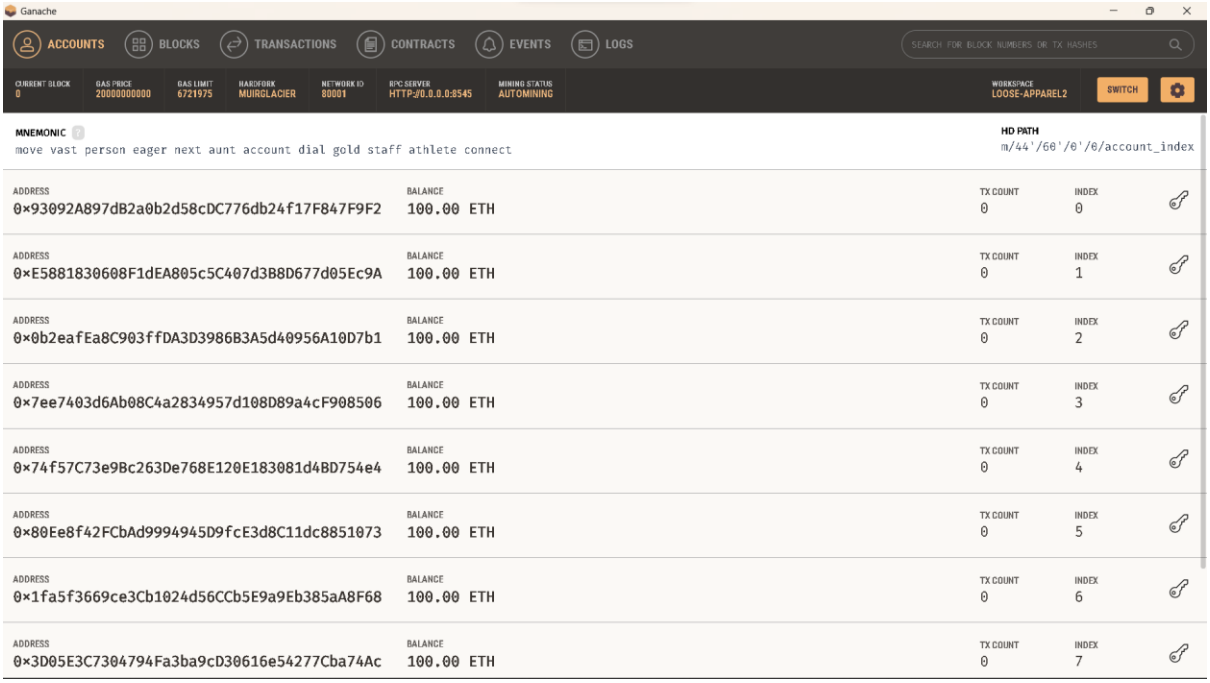
- Install Truffle
- Create a project
- Create Smart Contract
- Compile Smart contract

Now perform the following Steps :

1. **Creating a Ganache Ethereum blockchain instance**
2. **Deploying the smart contract to Ganache**
3. **Testing the Smart Contract**

Creating a Ganache Ethereum blockchain instance

After download Ganache, quick-start a blockchain instance. You should see a screen like this:



Check the port in RPC Server configuration., which is usually 7545. Now, make the following change in truffle-config.js and fill out the correct port number:

```
networks: {
  development: {
   host: "127.0.0.1",     // Localhost (default: none)
   port: 8545,           // Standard Ethereum port (default: none)
   network_id: "*",       // Any network (default: none)
  },
},
```

Deploying the smart contract to the Ganache Ethereum local test network

Now, let's deploy this smart contract to the blockchain instance started by Ganache local test network.

In the /migrations folder, you'll see the initial migration. You'll have to create a new migration file for this new smart contract.

Create 2_deploy_contracts.js ,and place the following code:

```
// Help Truffle find `supplyChain.sol` in the `/contracts` directory
```

```
const supplyChain = artifacts.require("supplyChain");

module.exports = function(deployer) {
  // Command Truffle to deploy the Smart Contract
  deployer.deploy(supplyChain);
};
```

Truffle migrations are basically scripts that help us deploy our contracts to the blockchain. Let's try deploying by executing the command:

- truffle migrate

It will get the output like this

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM\SCM> truffle migrate

Compiling your contracts...
   - solc: 0.5.1+commit.c8a2cb62.Emscripten.clang


Starting migrations...
======================
> Network name:    'development'
> Network id:      80001
> Block gas limit: 6721975 (0x6691b7)


2_deploy_contracts.js
=====================

   Replacing 'supplyChain'
   -----------------------
   > transaction hash:    0x1eec5976bdf65fa520a77a5a222e531a58476fa4ec8fc259081d6747eaba9e19
   > Blocks: 0            Seconds: 0
   > contract address:    0x8B49E285f3b301B8561A0aF54785839ac3Ae08Ac
   > block number:        2
   > block timestamp:     1666433589
   > account:             0x93092A897dB2a0b2d58cDC776db24f17F847F9F2
   > balance:             99.87395492
   > gas used:            3151127 (0x301517)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.06302254 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.06302254 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.06302254 ETH
```

Testing the smart contract

We will be writing tests in the /test folder. Create a new file there named supplyChain_test.js, and place the following code :

```
var SupplyChain = artifacts.require('./supplyChain.sol');

contract('supplyChain', async accounts => {

   it("should create a Participant", async () => {
   let instance = await SupplyChain.deployed();

   let participantId = await
instance.addParticipant("A","passA","0x8858d98eC700363a2A1D9308c7312653d186f9B0","Manufactu
rer");
   let participant = await instance.participants(0);
   assert.equal(participant[0], "A");
   assert.equal(participant[2], "Manufacturer");

   participantId = await
instance.addParticipant("B","passB","0xd295d0BF5Fb583219CB7b8AB1a3F3f5E218D0442","Supplier"
);
   participant = await instance.participants(1);
   assert.equal(participant[0], "B");
   assert.equal(participant[2], "Supplier");

   participantId = await
instance.addParticipant("C","passC","0x9c4c246bca58D3b821bFFdbdB88D60E8E2727E84","Consumer
");
   participant = await instance.participants(2);
   assert.equal(participant[0], "C");
   assert.equal(participant[2], "Consumer");
});

   it("should return Participant details", async () => {
   let instance = await SupplyChain.deployed();
   let participantDetails = await instance.getParticipant(0);
   assert.equal(participantDetails[0], "A");
   instance = await SupplyChain.deployed();
   participantDetails = await instance.getParticipant(1);
   assert.equal(participantDetails[0], "B");
   instance = await SupplyChain.deployed();
   participantDetails = await instance.getParticipant(2);
   assert.equal(participantDetails[0], "C");
})
});
```

Now, run the following code in terminal :

- truffle test

You should see all of the tests pass:

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\SCM\SCM\SCM> truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Compiling .\contracts\suppluChain.sol
> Artifacts written to C:\Users\RAj\AppData\Local\Temp\test--18712-cQxajZv35Whh
> Compiled successfully using:
   - solc: 0.5.1+commit.c8a2cb62.Emscripten.clang


  Contract: supplyChain
    ✓ should create a Participant (3015ms)
    ✓ should return Participant details (424ms)


  2 passing (4s)
```

Practical 8

Aim :

Deploy the tested smart contract code in practical 7th over the Blockchain network created in practical 3rd.

Solution:

Prerequest :

Install the following :

- Ganache Personal Blockchain
- Node.JS
- Truffle Framework

Now perform the following Steps :

1. **Create a project**
2. **Create Smart Contract**
3. **Compile Smart contract**
4. **Creating a Ganache Ethereum blockchain instance**
5. **Deploying the smart contract to Ganache**
6. **Testing the Smart Contract**

Creating a project

Create a new directory for your Truffle project:

- mkdir SCM
- cd SCM

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT> mkdir product-marketplace


    Directory: C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        22-10-2022     16:08                product-marketplace


PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT> cd .\product-marketplace\
```

create a bare project template using

- truffle init

```
Starting init...
================

> Copying project files to C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\product-marketplace

Init successful, sweet!

Try our scaffold commands to get started:
  $ truffle create contract YourContractName # scaffold a contract
  $ truffle create test YourTestName         # scaffold a test

http://trufflesuite.com/docs
```

Create Smart Contract

Create a smart contract under *productMaeketplace/contracts/ Marketplace.sol* and copy the following code :

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.5.0;

contract Marketplace {
  string public name;
  uint public productCount = 0;
  mapping(uint => Product) public products;

  struct Product {
    uint id;
    string name;
    uint price;
    address payable owner;
    bool purchased;
  }
```

```
event ProductCreated(
    uint id,
    string name,
    uint price,
    address payable owner,
    bool purchased
);

event ProductPurchased(
    uint id,
    string name,
    uint price,
    address payable owner,
    bool purchased
);

constructor() public {
    name = "Dapp University Marketplace";
}

function createProduct(string memory _name, uint _price) public {
    // Require a valid name
    require(bytes(_name).length > 0);
    // Require a valid price
    require(_price > 0);
    // Increment product count
    productCount ++;
    // Create the product
    products[productCount] = Product(productCount, _name, _price, msg.sender, false);
    // Trigger an event
    emit ProductCreated(productCount, _name, _price, msg.sender, false);
}

function purchaseProduct(uint _id) public payable {
    // Fetch the product
    Product memory _product = products[_id];
    // Fetch the owner
    address payable _seller = _product.owner;
    // Make sure the product has a valid id
    require(_product.id > 0 && _product.id <= productCount);
    // Require that there is enough Ether in the transaction
    require(msg.value >= _product.price);
    // Require that the product has not been purchased already
    require(!_product.purchased);
    // Require that the buyer is not the seller
```

```
    require(_seller != msg.sender);
    // Transfer ownership to the buyer
    _product.owner = msg.sender;
    // Mark as purchased
    _product.purchased = true;
    // Update the product
    products[_id] = _product;
    // Pay the seller by sending them Ether
    address(_seller).transfer(msg.value);
    // Trigger an event
    emit ProductPurchased(productCount, _product.name, _product.price, msg.sender, true);
  }
}
```

Compiling Smart contract

Let's try compiling our smart contract! Execute the command:

- truffle compile

Now, compilation could fail and show an error similar to this:

- ParserError: Source file requires different compiler version

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\product-marketplace> truffle compile

Compiling your contracts...
===========================
> Compiling .\contracts\productMarketplace.sol

CompileError: ParserError: Source file requires different compiler version (current compiler is 0.8.17+commit.8df45f5f.Emscripten.clang) - note that nightly
builds are considered to be strictly less than the released version
 --> project:/contracts/productMarketplace.sol:2:1:
  |
2 | pragma solidity ^0.5.0;
  | ^^^^^^^^^^^^^^^^^^^^^^^


Error: Truffle is currently using solc 0.8.17, but one or more of your contracts specify "pragma solidity ^0.5.0".
Please update your truffle config or pragma statement(s).
(See https://trufflesuite.com/docs/truffle/reference/configuration#compiler-configuration for information on
configuring Truffle to use a specific solc compiler version.)

Compilation failed. See above.
    at C:\Users\RAj\AppData\Roaming\npm\node_modules\truffle\build\webpack:\packages\compile-solidity\dist\run.js:95:1
    at Generator.next (<anonymous>)
    at fulfilled (C:\Users\RAj\AppData\Roaming\npm\node_modules\truffle\build\webpack:\packages\compile-solidity\dist\run.js:28:43)
Truffle v5.6.2 (core: 5.6.2)
Node v16.16.0
```

To solve the error change the compiler version in truffle-config.js file ,as below:

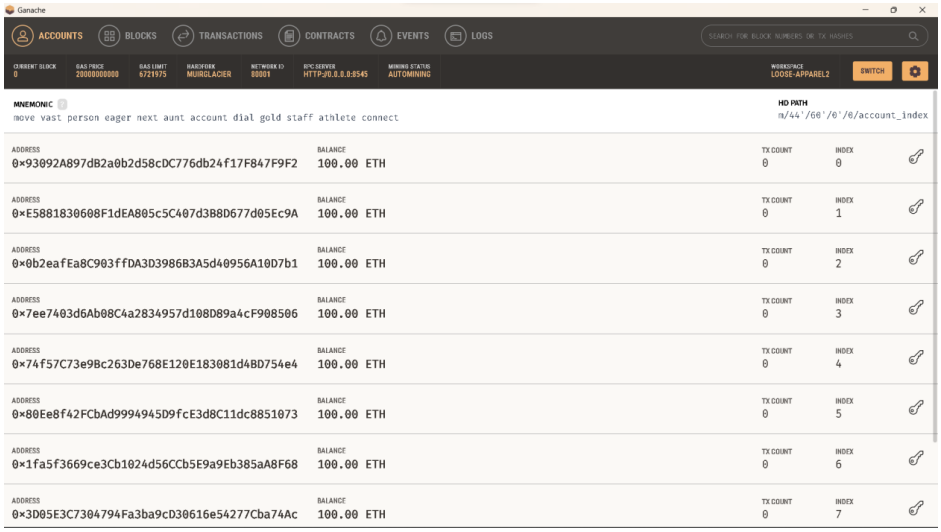
```
// Configure your compilers
compilers: {
 solc: {
  version: "0.5.0"
 }
}
```

Compiling again

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\product-marketplace> truffle compile

Compiling your contracts...
===========================
> Compiling .\contracts\productMarketplace.soltempt #1
> Artifacts written to C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\product-marketplace\build\contracts
> Compiled successfully using:
   - solc: 0.5.0+commit.1d4f565a.Emscripten.clang
```

Creating a Ganache Ethereum blockchain instance

After [download Ganache](), quick-start a blockchain instance. You should see a screen like this:



Check the port in RPC Server configuration., which is usually 7545. Now, make the following change in truffle-config.js and fill out the correct port number:

```
networks: {
  development: {
   host: "127.0.0.1",     // Localhost (default: none)
   port: 8545,          // Standard Ethereum port (default: none)
   network_id: "*",       // Any network (default: none)
  },
},
```

Deploying the smart to the Ganache Ethereum local test network

Now, let's deploy this smart contract to the blockchain instance started by Ganache local test network.

In the /migrations folder, you'll see the initial migration. You'll have to create a new migration file for this new smart contract.
Create 2_deploy_contracts.js ,and place the following code:

```
// Help Truffle find `Marketplace.sol` in the `/contracts` directory
const Marketplace = artifacts.require("Marketplace");

module.exports = function(deployer) {
  // Command Truffle to deploy the Smart Contract
  deployer.deploy(Marketplace);
};
```

Truffle migrations are basically scripts that help us deploy our contracts to the blockchain. Let's try deploying by executing the command:

- truffle migrate

It will get the output like this

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\product-marketplace> truffle migrate

> Network name:    'development'
> Network id:      80001
> Block gas limit: 6721975 (0x6691b7)


2_deploy_contracts.js
=====================

   Deploying 'Marketplace'
   -----------------------
   > transaction hash:    0xea1d4a38e7b2d721bbb8a7cf451f9285e2c99e9ed17adf41520a60cab8eb036c
   > Blocks: 0            Seconds: 0
   > contract address:    0xfD22EEFd70de480a880367DA9B1AAE0d96B28823
   > block number:        7
   > block timestamp:     1666436378
   > account:             0x93092A897dB2a0b2d58cDC776db24f17F847F9F2
   > balance:             99.78874934
   > gas used:            746110 (0xb627e)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0149222 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:           0.0149222 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.0149222 ETH
```

Testing the smart contract

We will be writing tests in the /test folder. Create a new file there named marketplace_test.js, and place the following code :

```
const Marketplace = artifacts.require('./Marketplace.sol')

require('chai')
 .use(require('chai-as-promised'))
 .should()

contract('Marketplace', ([deployer, seller, buyer]) => {
 let marketplace

 before(async () => {
  marketplace = await Marketplace.deployed()
 })

 describe('deployment', async () => {
  it('deploys successfully', async () => {
   const address = await marketplace.address
   assert.notEqual(address, 0x0)
   assert.notEqual(address, '')
   assert.notEqual(address, null)
   assert.notEqual(address, undefined)
  })

  it('has a name', async () => {
   const name = await marketplace.name()
   assert.equal(name, 'Dapp University Marketplace')
  })
 })

 describe('products', async () => {
  let result, productCount

  before(async () => {
   result = await marketplace.createProduct('iPhone X', web3.utils.toWei('1', 'Ether'), { from: seller })
   productCount = await marketplace.productCount()
  })

  it('creates products', async () => {
   // SUCCESS
   assert.equal(productCount, 1)
   const event = result.logs[0].args
   assert.equal(event.id.toNumber(), productCount.toNumber(), 'id is correct')
   assert.equal(event.name, 'iPhone X', 'name is correct')
   assert.equal(event.price, '1000000000000000000', 'price is correct')
   assert.equal(event.owner, seller, 'owner is correct')
   assert.equal(event.purchased, false, 'purchased is correct')

   // FAILURE: Product must have a name
```

```
    await await marketplace.createProduct('', web3.utils.toWei('1', 'Ether'), { from: seller
}).should.be.rejected;
    // FAILURE: Product must have a price
    await await marketplace.createProduct('iPhone X', 0, { from: seller }).should.be.rejected;
  })

  it('lists products', async () => {
    const product = await marketplace.products(productCount)
    assert.equal(product.id.toNumber(), productCount.toNumber(), 'id is correct')
    assert.equal(product.name, 'iPhone X', 'name is correct')
    assert.equal(product.price, '1000000000000000000', 'price is correct')
    assert.equal(product.owner, seller, 'owner is correct')
    assert.equal(product.purchased, false, 'purchased is correct')
  })

  it('sells products', async () => {
    // Track the seller balance before purchase
    let oldSellerBalance
    oldSellerBalance = await web3.eth.getBalance(seller)
    oldSellerBalance = new web3.utils.BN(oldSellerBalance)

    // SUCCESS: Buyer makes purchase
    result = await marketplace.purchaseProduct(productCount, { from: buyer, value:
web3.utils.toWei('1', 'Ether')})

    // Check logs
    const event = result.logs[0].args
    assert.equal(event.id.toNumber(), productCount.toNumber(), 'id is correct')
    assert.equal(event.name, 'iPhone X', 'name is correct')
    assert.equal(event.price, '1000000000000000000', 'price is correct')
    assert.equal(event.owner, buyer, 'owner is correct')
    assert.equal(event.purchased, true, 'purchased is correct')

    // Check that seller received funds
    let newSellerBalance
    newSellerBalance = await web3.eth.getBalance(seller)
    newSellerBalance = new web3.utils.BN(newSellerBalance)

    let price
    price = web3.utils.toWei('1', 'Ether')
    price = new web3.utils.BN(price)

    const exepectedBalance = oldSellerBalance.add(price)

    assert.equal(newSellerBalance.toString(), exepectedBalance.toString())

    // FAILURE: Tries to buy a product that does not exist, i.e., product must have valid id
    await marketplace.purchaseProduct(99, { from: buyer, value: web3.utils.toWei('1',
'Ether')}).should.be.rejected;     // FAILURE: Buyer tries to buy without enough ether
    // FAILURE: Buyer tries to buy without enough ether
```

```
    await marketplace.purchaseProduct(productCount, { from: buyer, value: web3.utils.toWei('0.5',
'Ether') }).should.be.rejected;
    // FAILURE: Deployer tries to buy the product, i.e., product can't be purchased twice
    await marketplace.purchaseProduct(productCount, { from: deployer, value: web3.utils.toWei('1',
'Ether') }).should.be.rejected;
    // FAILURE: Buyer tries to buy again, i.e., buyer can't be the seller
    await marketplace.purchaseProduct(productCount, { from: buyer, value: web3.utils.toWei('1',
'Ether') }).should.be.rejected;
  })

 })
})
```

Now, run the following code in terminal :

- truffle test

Note:
After running truffle test if you get error like Error: Cannot find module 'chai', then run this command

- npm install --save-dev chai
- npm install --save-dev chai-as-promised

now again run truffle-test command

You should see all of the tests pass:

```
PS C:\Users\RAj\Dropbox\My PC (DESKTOP-GI226UK)\Documents\BCT\product-marketplace> truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Compiling .\contracts\Marketplace.sol
> Artifacts written to C:\Users\RAj\AppData\Local\Temp\test--23096-t1YF1iJWyHZD
> Compiled successfully using:
   - solc: 0.5.0+commit.1d4f565a.Emscripten.clang


  Contract: Marketplace
    deployment
      ✓ deploys successfully
      ✓ has a name (215ms)
    products
      ✓ creates products (2210ms)
      ✓ lists products (114ms)
      ✓ sells products (1772ms)


  5 passing (7s)
```

Practical 9

Aim :

Create dapp (Decentralized Application) and link your client-side application with Blockchain network created in practical 8th.

Solution:

Prerequest :

- Perform the previous practical
- Install Metamask Ethereum Wallet (Steps)

We will perform the following steps :

1. **Project Setup**
2. **Frontend Code**
3. **Connect Frontend with Blockchain**
4. **Dapp Demo**

Project Setup

Open github-bash terminal and clone the starter-kit from this link
https://github.com/dappuniversity/starter_kit

```
RAj@DESKTOP-GI226UK MINGW64 ~/Dropbox/My PC (DESKTOP-GI226UK)/Documents/BCT/product-marketplace
$ cd ..

RAj@DESKTOP-GI226UK MINGW64 ~/Dropbox/My PC (DESKTOP-GI226UK)/Documents/BCT
$ git clone https://github.com/dappuniversity/starter_kit.git
Cloning into 'starter_kit'...
remote: Enumerating objects: 87, done.
remote: Total 87 (delta 0), reused 0 (delta 0), pack-reused 87
Receiving objects: 100% (87/87), 417.00 KiB | 588.00 KiB/s, done.
Resolving deltas: 100% (30/30), done.
```

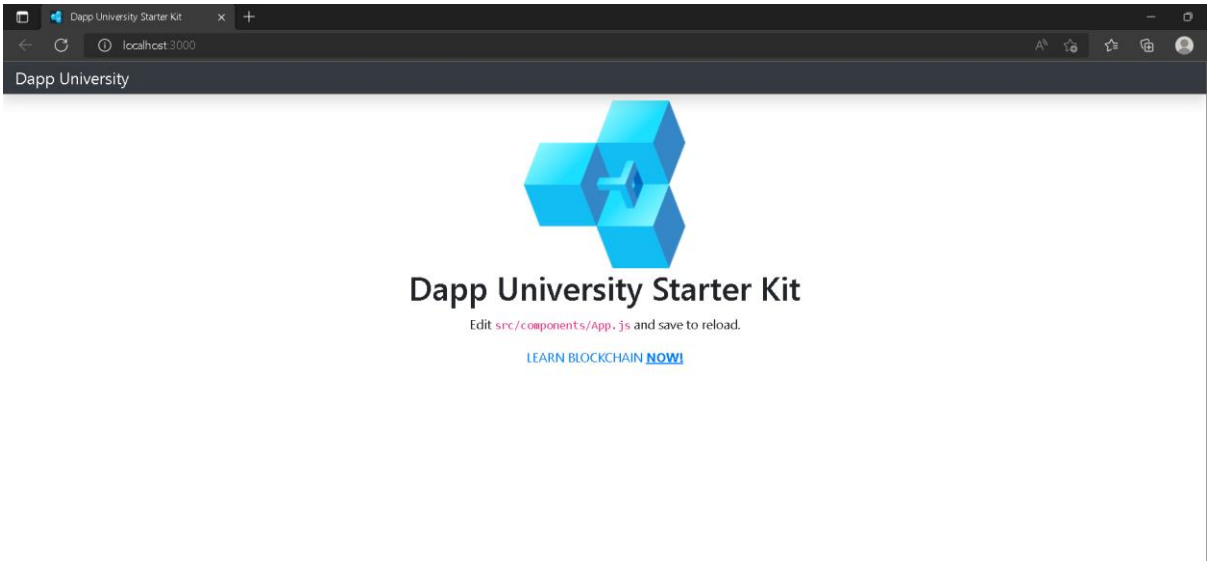
Open the starter kit in VS code and install the dependencies through terminal using command:

- npm install

Before we begin, make sure that your development server is running:

- npm run start

This should start your web server and automatically open the website in your browser like this:



Now let's connect our web browser to the blockchain and we'll use Metamask for this. To do this we'll need to do two things:

- Connect Metamask to our Ganache personal blockchain instance
- Import some accounts from Ganache into Metamask so that we can act on their behalf as users of our marketplace application

---

Frontend Code

App.js

Replace the code of App.js with below code :

```
import React, { Component } from 'react';
import Web3 from 'web3'
import logo from '../logo.png';
import './App.css';
import Marketplace from '../abis/Marketplace.json'
import Navbar from './Navbar'
import Main from './Main'

class App extends Component {

 async componentWillMount() {
  await this.loadWeb3()
  await this.loadBlockchainData()
 }

 async loadWeb3() {
  if (window.ethereum) {
   window.web3 = new Web3(window.ethereum)
```

```
    await window.ethereum.enable()
  }
  else if (window.web3) {
    window.web3 = new Web3(window.web3.currentProvider)
  }
  else {
    window.alert('Non-Ethereum browser detected. You should consider trying MetaMask!')
  }
}

async loadBlockchainData() {
  const web3 = window.web3
  // Load account
  const accounts = await web3.eth.getAccounts()
  this.setState({ account: accounts[0] })
  const networkId = await web3.eth.net.getId()
  const networkData = Marketplace.networks[networkId]
  if(networkData) {
    const marketplace = web3.eth.Contract(Marketplace.abi, networkData.address)
    this.setState({ marketplace })
    const productCount = await marketplace.methods.productCount().call()
    this.setState({ productCount })
    // Load products
    for (var i = 1; i <= productCount; i++) {
      const product = await marketplace.methods.products(i).call()
      this.setState({
        products: [...this.state.products, product]
      })
    }
    this.setState({ loading: false})
  } else {
    window.alert('Marketplace contract not deployed to detected network.')
  }
}

constructor(props) {
  super(props)
  this.state = {
    account: '',
    productCount: 0,
    products: [],
    loading: true
  }

  this.createProduct = this.createProduct.bind(this)
  this.purchaseProduct = this.purchaseProduct.bind(this)
}

createProduct(name, price) {
  this.setState({ loading: true })
  this.state.marketplace.methods.createProduct(name, price).send({ from: this.state.account })
  .once('receipt', (receipt) => {
```

```
    this.setState({ loading: false })
  })
 }

 purchaseProduct(id, price) {
  this.setState({ loading: true })
  this.state.marketplace.methods.purchaseProduct(id).send({ from: this.state.account, value: price })
  .once('receipt', (receipt) => {
    this.setState({ loading: false })
  })
 }

 render() {
  return (
    <div>
      <Navbar account={this.state.account} />
      <div className="container-fluid mt-5">
        <div className="row">
          <main role="main" className="col-lg-12 d-flex">
            { this.state.loading
              ? <div id="loader" className="text-center"><p className="text-center">Loading...</p></div>
              : <Main
                products={this.state.products}
                createProduct={this.createProduct}
                purchaseProduct={this.purchaseProduct} />
            }
          </main>
        </div>
      </div>
    </div>
  );
 }
}

export default App;
```

Main.js

Create a file name Main.js inside src/components folder and paste the below code :

```
import React, { Component } from 'react';

class Main extends Component {

 render() {
  return (
   <div id="content">
    <h1>Add Product</h1>
    <form onSubmit={(event) => {
     event.preventDefault()
     const name = this.productName.value
     const price = window.web3.utils.toWei(this.productPrice.value.toString(), 'Ether')
     this.props.createProduct(name, price)
    }}>
     <div className="form-group mr-sm-2">
      <input
       id="productName"
       type="text"
       ref={(input) => { this.productName = input }}
       className="form-control"
       placeholder="Product Name"
       required />
     </div>
     <div className="form-group mr-sm-2">
      <input
       id="productPrice"
       type="text"
       ref={(input) => { this.productPrice = input }}
       className="form-control"
       placeholder="Product Price"
       required />
     </div>
     <button type="submit" className="btn btn-primary">Add Product</button>
    </form>
    <p> </p>
    <h2>Buy Product</h2>
    <table className="table">
     <thead>
      <tr>
       <th scope="col">#</th>
       <th scope="col">Name</th>
       <th scope="col">Price</th>
       <th scope="col">Owner</th>
       <th scope="col"></th>
      </tr>
     </thead>
     <tbody id="productList">
      { this.props.products.map((product, key) => {
```

```
          return(
            <tr key={key}>
              <th scope="row">{product.id.toString()}</th>
              <td>{product.name}</td>
              <td>{window.web3.utils.fromWei(product.price.toString(), 'Ether')} Eth</td>
              <td>{product.owner}</td>
              <td>
               { !product.purchased
                ? <button
                    name={product.id}
                    value={product.price}
                    onClick={(event) => {
                      this.props.purchaseProduct(event.target.name, event.target.value)
                    }}
                  >
                    Buy
                  </button>
                : null
               }
              </td>
            </tr>
          )
        })}
      </tbody>
    </table>
  </div>
  );
 }
}

export default Main;
```

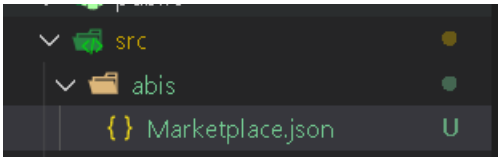
Navbar.js

Create a file name Navbar.js inside src/components folder and paste the below code :

```
import React, { Component } from 'react';

class Navbar extends Component {

 render() {
  return (
    <nav className="navbar navbar-dark fixed-top bg-dark flex-md-nowrap p-0 shadow">
     <a
      className="navbar-brand col-sm-3 col-md-2 mr-0"
      href="http://www.dappuniversity.com/bootcamp"
      target="_blank"
      rel="noopener noreferrer"
     >
      Dapp University's Blockchain Marketplace
     </a>
     <ul className="navbar-nav px-3">
      <li className="nav-item text-nowrap d-none d-sm-none d-sm-block">
       <small className="text-white"><span id="account">{this.props.account}</span></small>
      </li>
     </ul>
    </nav>
  );
 }
}

export default Navbar;
```

Connect Frontend with Blockchain

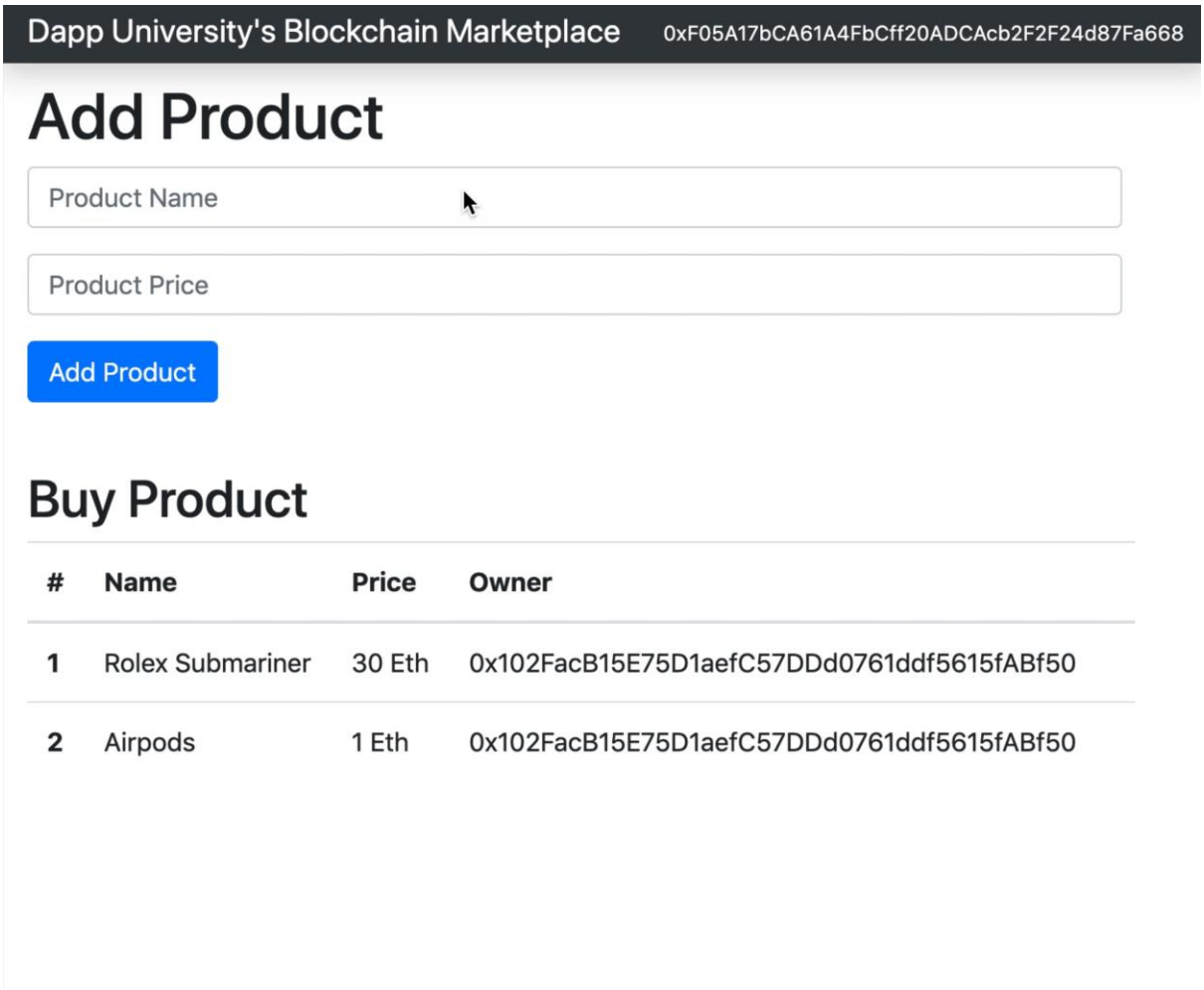
Now create a folder name *abis* inside *src* folder and create a file named *Markerplace.json* inside it



Copy the code from *product-marketplace/build/contracts/Marketplace.json* and paste it in the *starter-kit/src/abis/Marketplace.json*

Note : here product-marketplace/build/contracts/Marketplace.json is the code which is generated when we compile the smart contract . It is from the precious practical .

Dapp Demo


Dapp University's Blockchain Marketplace
0xF05A17bCA61A4FbCff20ADCAcb2F2F24d87Fa668
Add Product
Product Name
Product Price
Add Product
Buy Product
Name Price Owner
1 Rolex Submariner 30 Eth 0x102FacB15E75D1aefC57DDd0761ddf5615fABf50
2 Airpods 1 Eth 0x102FacB15E75D1aefC57DDd0761ddf5615fABf50

Practical 10

Aim :

Install and configure the Hyperledger Fabric on Linux platform. Create dApp for insurance application over hyperledger fabric.

Solution:

Prerequisites

We find that Blockchain can be finicky when it comes to installing Node. We want to share this StackOverflow response - because many times the errors you see with Compose are derived in having installed either the wrong Node version or took an approach that is not supported by Compose:

- IBM Cloud account
- Docker - latest
- Docker Compose - latest
- NPM - latest
- nvm - latest
- Node.js - Node v8.9.x
- Git client - latest
- **Python - 2.7.x**
- React - 15.6.1

Steps

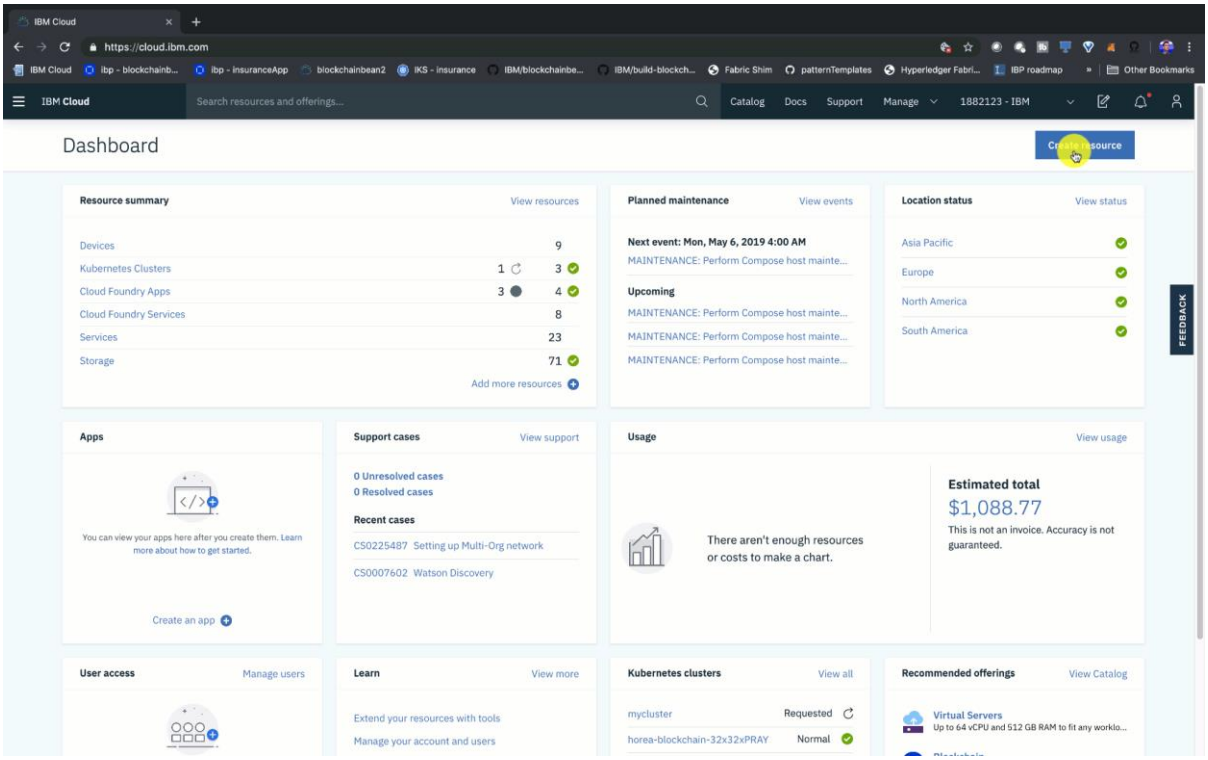
1. **Create IBM Cloud services**
2. **Build a network - Certificate Authority**
3. **Build a network - Create MSP Definitions**
4. **Build a network - Create Peers**
5. **Build a network - Create Orderer**
6. **Build a network - Create and Join Channel**
7. **Deploy Insurance Smart Contract on the network**
8. **Connect application to the network**
9. **Enroll App Admin Identities**
10. **Run the application**

Important Note: This pattern is more advanced because it uses four organizations. For this reason, you will likely have to get a paid kubernetes cluster to run this pattern on the cloud, since a free cluster will not have the CPU/storage necessary to deploy all of the pods that we need to run this pattern. There are other patterns that leverage a free Kubernetes cluster (and only two organizations), so if you want to try that one out first, go here.

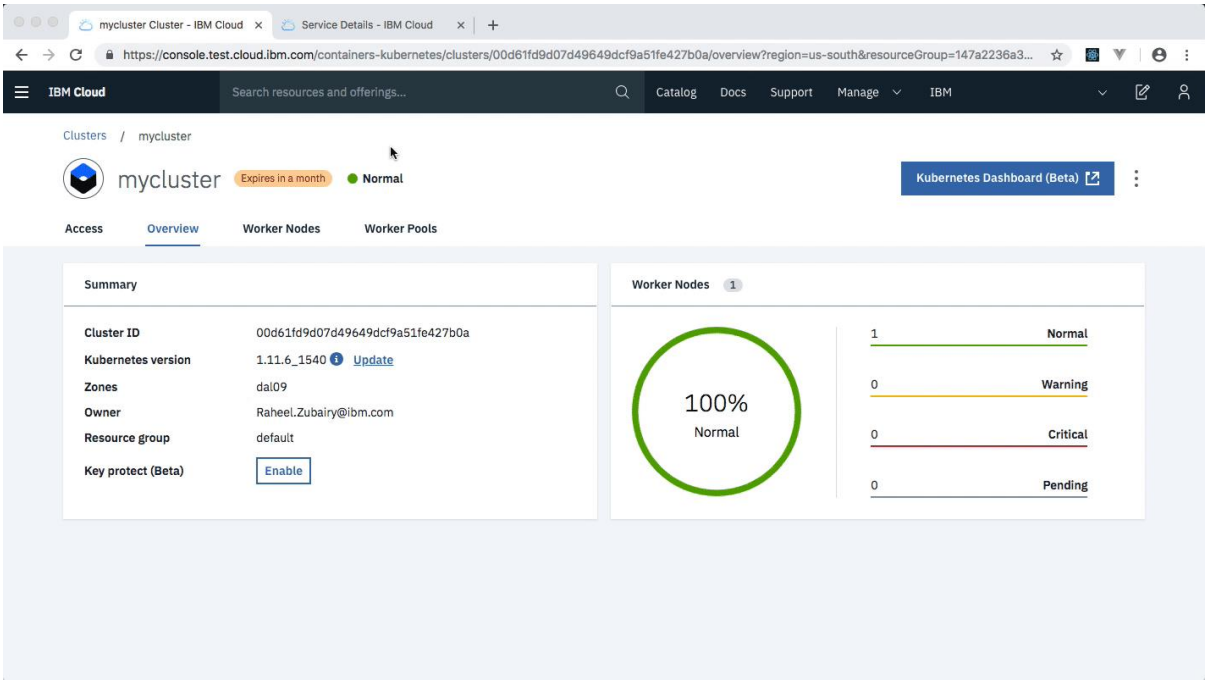
Step 1. Create IBM Cloud services

- Create the IBM Cloud Kubernetes Service. You can find the service in the Catalog. Note that **for this code pattern, we need to use the 32CPU, 32GB RAM cluster.**

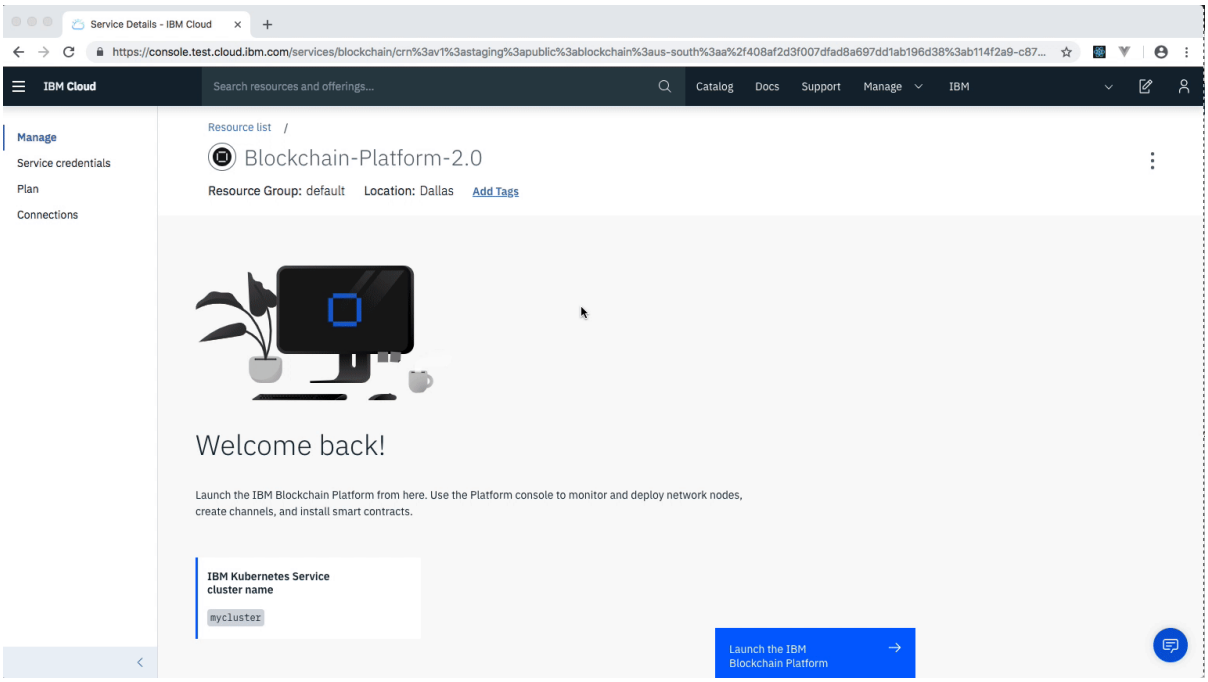
- Once you reach the **create a new cluster page** you will need to do the following:
    - Choose **standard** cluster type
    - Fill out cluster name
    - choose Geography: **North America**
    - Choose Location and availability: **Multizone**
    - Choose Metro: **Dallas**
    - Choose Worker nodes: **Dallas 10 only**
    - Choose Master service endpoint: **Both private & public endpoints**
    - Choose Default worker pool: **1.12.7 (Stable, Default)**
    - Choose Master service endpoint: **Both private & public endpoints**
    - Choose Flavor **32 Cores 32GB RAM, Ubuntu 18**
    - Choose Encrypt local disk **Yes**
    - Choose Worker nodes **3**
    - Click on **create cluster. The cluster takes around 15-20 minutes to provision, so please be patient!**



- After your kubernetes cluster is up and running, you can deploy your IBM Blockchain Platform V2 Beta on the cluster. The service walks through few steps and finds your cluster on the IBM Cloud to deploy the service on.



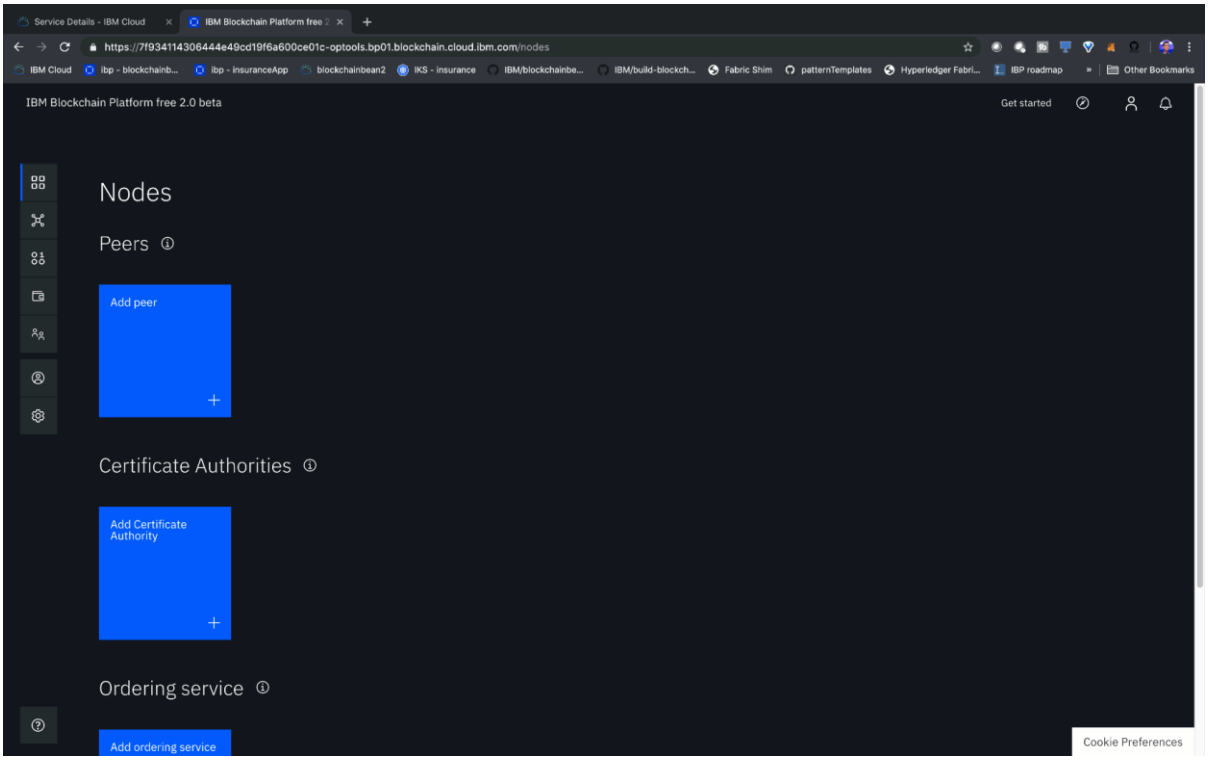
- Once the Blockchain Platform is deployed on the Kubernetes cluster, you can launch the console to start operating on your blockchain network.



Step 2. Build a network - Certificate Authority

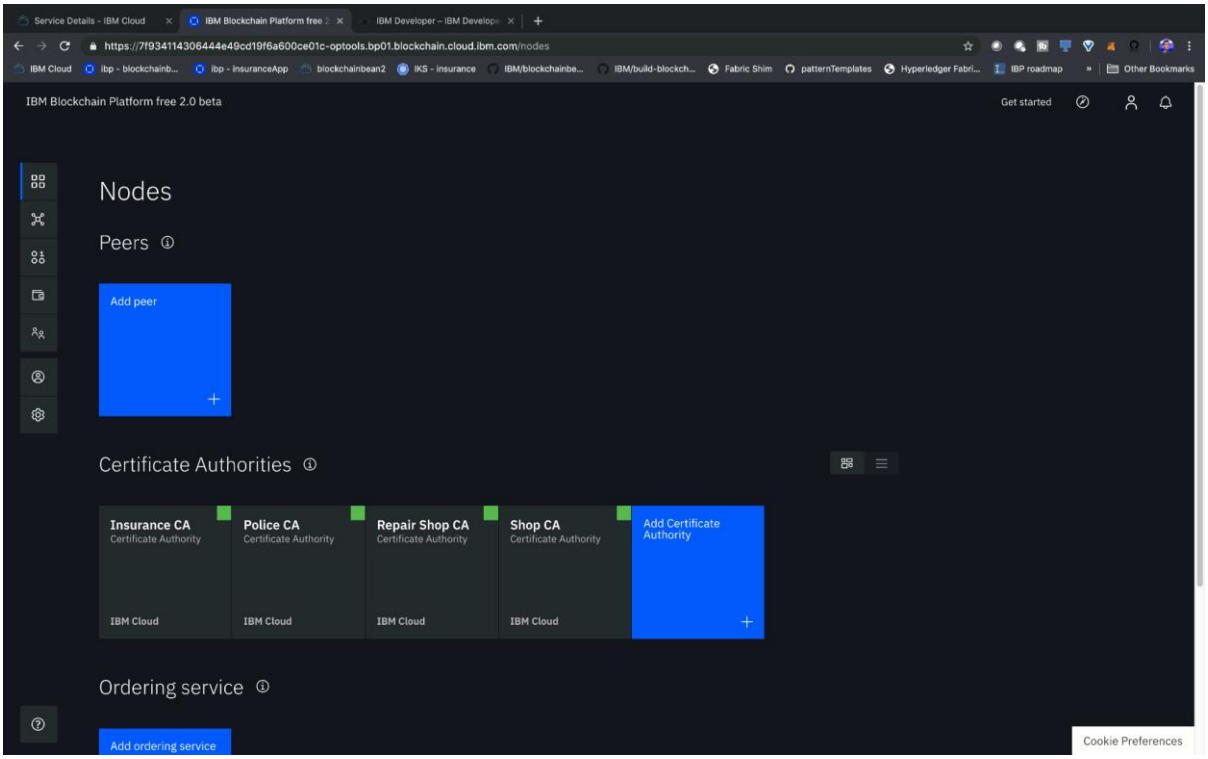
We will build a network as provided by the IBM Blockchain Platform documentation. This will include creating a channel with a single peer organization with its own MSP and CA (Certificate Authority), and an orderer organization with its own MSP and CA. We will create the respective identities to deploy peers and operate nodes.

- *Create your insurance organization CA*
    - Click **Add Certificate Authority**.
    - Click **IBM Cloud** under **Create Certificate Authority** and **Next**.
    - Give it a **Display name** of Insurance CA.
    - Specify an **Admin ID** of admin and **Admin Secret** of adminpw.



- *Create your shop organization CA (process is same as shown in gif above)*
    - Click **Add Certificate Authority**.
    - Click **IBM Cloud** under **Create Certificate Authority** and **Next**.
    - Give it a **Display name** of Shop CA.
    - Specify an **Admin ID** of admin and **Admin Secret** of adminpw.
- *Create your repair shop organization CA (process is same as shown in gif above)*
    - Click **Add Certificate Authority**.
    - Click **IBM Cloud** under **Create Certificate Authority** and **Next**.
    - Give it a **Display name** of Repair Shop CA.
    - Specify an **Admin ID** of admin and **Admin Secret** of adminpw.
- *Create your police organization CA (process is same as shown in gif above)*
    - Click **Add Certificate Authority**.
    - Click **IBM Cloud** under **Create Certificate Authority** and **Next**.
    - Give it a **Display name** of Police CA.
    - Specify an **Admin ID** of admin and **Admin Secret** of adminpw.
- *Use your CA to register insurance identities*
    - Select the **Insurance CA** Certificate Authority that we created.
    - First, we will register an admin for our Insurance Organization. Click on the **Register User** button. Give an **Enroll ID** of insuranceAdmin, and **Enroll Secret** of insuranceAdminpw. Click **Next**. Set the **Type** for this identity as client and select org1 from the affiliated organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.
    - We will repeat the process to create an identity of the peer. Click on the **Register User** button. Give an **Enroll ID** of insurancePeer, and **Enroll Secret** of insurancePeerpw. Click **Next**. Set the **Type** for this identity as peer and select org1 from the affiliated

organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.



- *Use your CA to register shop identities (process is same as shown in gif above)*
    - Select the **Shop CA** Certificate Authority that we created.
    - First, we will register an admin for our Insurance Organization. Click on the **Register User** button. Give an **Enroll ID** of shopAdmin, and **Enroll Secret** of shopAdminpw. Click **Next**. Set the **Type** for this identity as client and select org1 from the affiliated organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.
    - We will repeat the process to create an identity of the peer. Click on the **Register User** button. Give an **Enroll ID** of shopPeer, and **Enroll Secret** of shopPeerpw. Click **Next**. Set the **Type** for this identity as peer and select org1 from the affiliated organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.

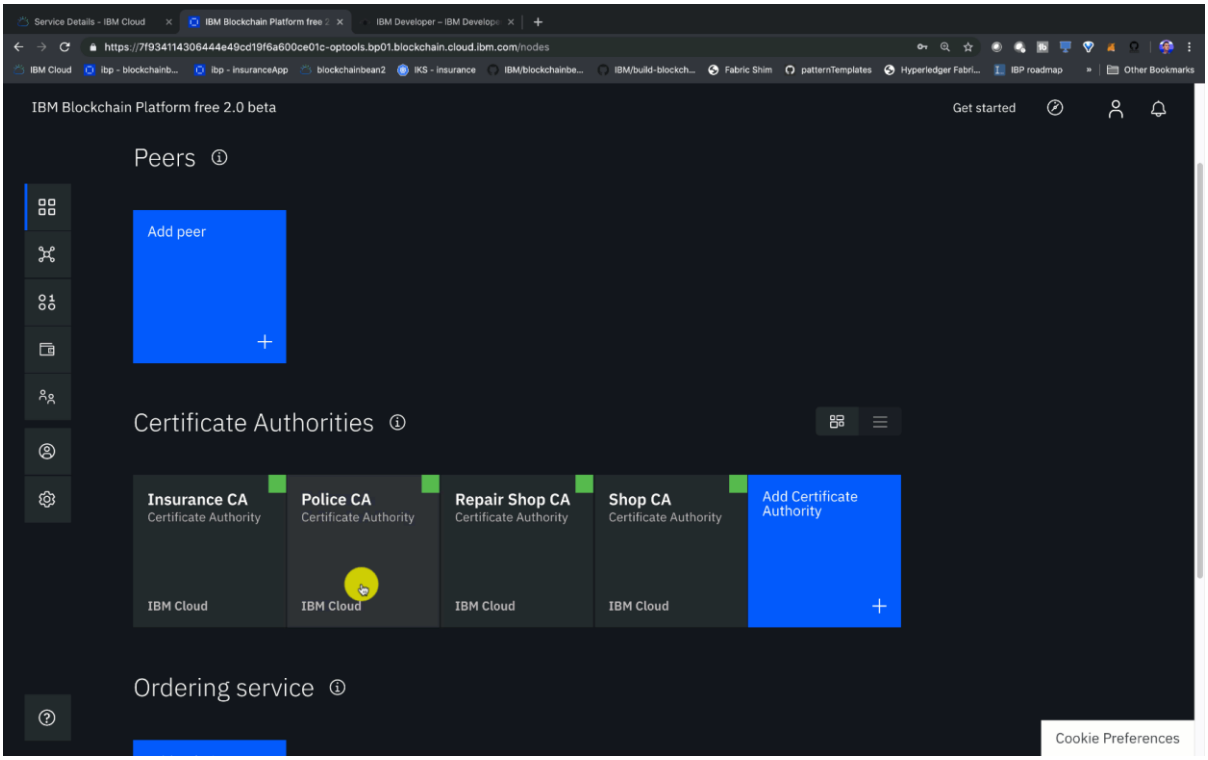
- *Use your CA to register repair shop identities (process is same as shown in gif above)*
    - Select the **Repair Shop CA** Certificate Authority that we created.
    - First, we will register an admin for our Insurance Organization. Click on the **Register User** button. Give an **Enroll ID** of repairShopAdmin, and **Enroll Secret** of repairShopAdminpw. Click **Next**. Set the **Type** for this identity as client and select org1 from the affiliated organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.
    - We will repeat the process to create an identity of the peer. Click on the **Register User** button. Give an **Enroll ID** of repairShopPeer, and **Enroll**

Secret of repairShopPeerpw. Click **Next**. Set the **Type** for this identity as peer and select org1 from the affiliated organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.

- *Use your CA to register police shop identities (process is same as shown in gif above)*
    - Select the **Police CA** Certificate Authority that we created.
    - First, we will register an admin for our Insurance Organization. Click on the **Register User** button. Give an **Enroll ID** of policeAdmin, and **Enroll Secret** of policeAdminpw. Click **Next**. Set the **Type** for this identity as client and select org1 from the affiliated organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.
    - We will repeat the process to create an identity of the peer. Click on the **Register User** button. Give an **Enroll ID** of policePeer, and **Enroll Secret** of policePeerpw. Click **Next**. Set the **Type** for this identity as peer and select org1 from the affiliated organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.

Step 3. Build a network - Create MSP Definitions

- *Create the insurance MSP definition*
    - Navigate to the **Organizations** tab in the left navigation and click **Create MSP definition**.
    - Enter the **MSP Display name** as Insurance MSP and an **MSP ID** of insurancemsp.
    - Under **Root Certificate Authority** details, specify the peer CA that we created Insurance CA as the root CA for the organization.
    - Give the **Enroll ID** and **Enroll secret** for your organization admin, insuranceAdmin and insuranceAdminpw. Then, give the Identity name, Insurance Admin.
    - Click the **Generate** button to enroll this identity as the admin of your organization and export the identity to the wallet. Click **Export** to export the admin certificates to your file system. Finally click **Create MSP definition**.



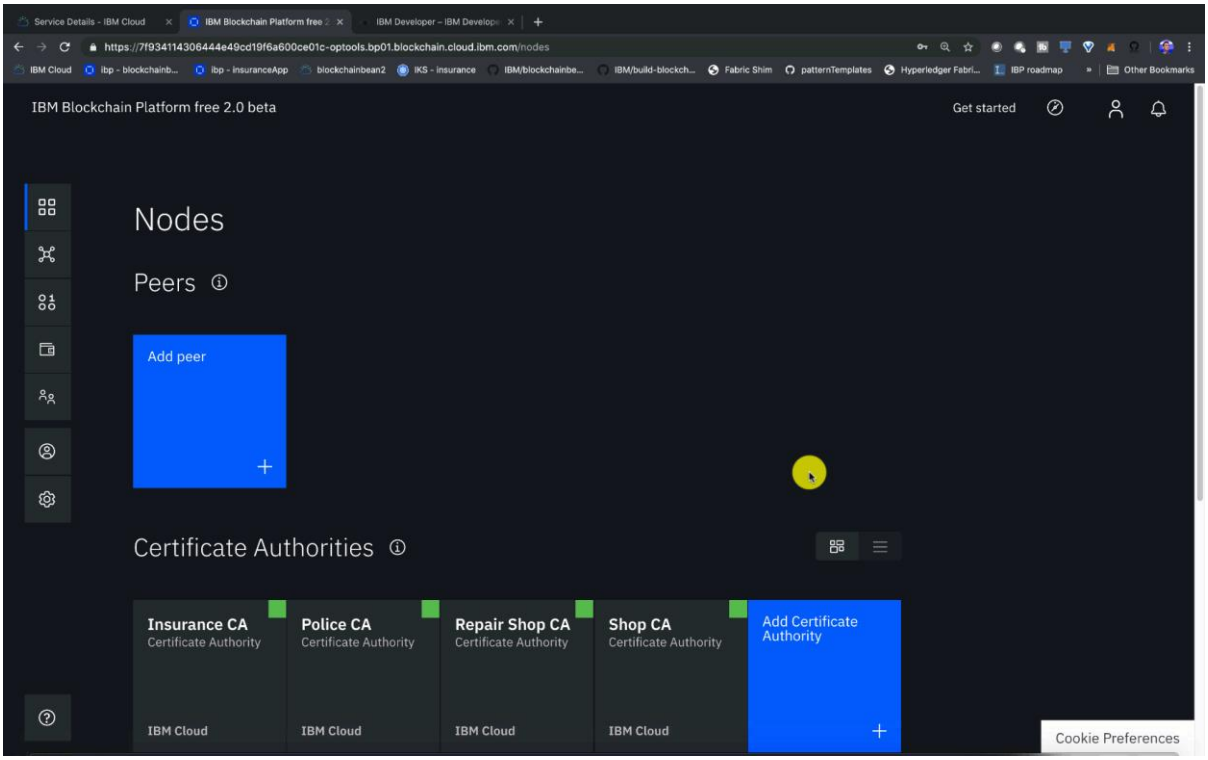
- *Create the shop MSP definition (same process as shown in gif above)*
    - Navigate to the **Organizations** tab in the left navigation and click **Create MSP definition**.
    - Enter the **MSP Display name** as Shop MSP and an **MSP ID** of shopmsp.
    - Under **Root Certificate Authority** details, specify the peer CA that we created Shop CA as the root CA for the organization.
    - Give the **Enroll ID** and **Enroll secret** for your organization admin, shopAdmin and shopAdminpw. Then, give the Identity name, Shop Admin.
    - Click the **Generate** button to enroll this identity as the admin of your organization and export the identity to the wallet. Click **Export** to export the admin certificates to your file system. Finally click **Create MSP definition**.

- *Create the repair shop MSP definition (same process as shown in gif above)*
    - Navigate to the **Organizations** tab in the left navigation and click **Create MSP definition**.
    - Enter the **MSP Display name** as Repair Shop MSP and an **MSP ID** of repairshopmsp.
    - Under **Root Certificate Authority** details, specify the peer CA that we created Repair Shop CA as the root CA for the organization.
    - Give the **Enroll ID** and **Enroll secret** for your organization admin, repairShopAdmin and repairShopAdminpw. Then, give the Identity name, Repair Shop Admin.
    - Click the **Generate** button to enroll this identity as the admin of your organization and export the identity to the wallet. Click **Export** to export the admin certificates to your file system. Finally click **Create MSP definition**.

- *Create the police MSP definition (same process as shown in gif above)*
    - Navigate to the **Organizations** tab in the left navigation and click **Create MSP definition**.
    - Enter the **MSP Display name** as Police MSP and an **MSP ID** of policemsp.
    - Under **Root Certificate Authority** details, specify the peer CA that we created Police CA as the root CA for the organization.
    - Give the **Enroll ID** and **Enroll secret** for your organization admin, policeAdmin and policeAdminpw. Then, give the Identity name, Police Admin.
    - Click the **Generate** button to enroll this identity as the admin of your organization and export the identity to the wallet. Click **Export** to export the admin certificates to your file system. Finally click **Create MSP definition**.

Step 4. Build a network - Create Peers

- Create an insurance peer
    - On the **Nodes** page, click **Add peer**.
    - Click **IBM Cloud** under Create a new peer and **Next**.
    - Give your peer a **Display name** of Insurance Peer.
    - On the next screen, select Insurance CA as your **Certificate Authority**. Then, give the **Enroll ID** and **Enroll secret** for the peer identity that you created for your peer, insurancePeer, and insurancePeerpw. Then, select the **Administrator Certificate (from MSP)**, Insurance MSP, from the drop-down list and click **Next**.
    - Give the **TLS Enroll ID**, admin, and **TLS Enroll secret**, adminpw, the same values are the Enroll ID and Enroll secret that you gave when creating the CA. Leave the **TLS CSR hostname** blank.
    - The last side panel will ask you to **Associate an identity** and make it the admin of your peer. Select your peer admin identity Insurance Admin.
    - Review the summary and click **Submit**.



- Create a shop peer (same process as shown in gif above)
  - On the **Nodes** page, click **Add peer**.
  - Click **IBM Cloud** under Create a new peer and **Next**.
  - Give your peer a **Display name** of Shop Peer.
  - On the next screen, select Shop CA as your **Certificate Authority**. Then, give the **Enroll ID** and **Enroll secret** for the peer identity that you created for your peer, shopPeer, and shopPeerpw. Then, select the **Administrator Certificate (from MSP)**, Shop MSP, from the drop-down list and click **Next**.
  - Give the **TLS Enroll ID**, admin, and **TLS Enroll secret**, adminpw, the same values are the Enroll ID and Enroll secret that you gave when creating the CA. Leave the **TLS CSR hostname** blank.
  - The last side panel will ask you to **Associate an identity** and make it the admin of your peer. Select your peer admin identity Shop Admin.
  - Review the summary and click **Submit**.

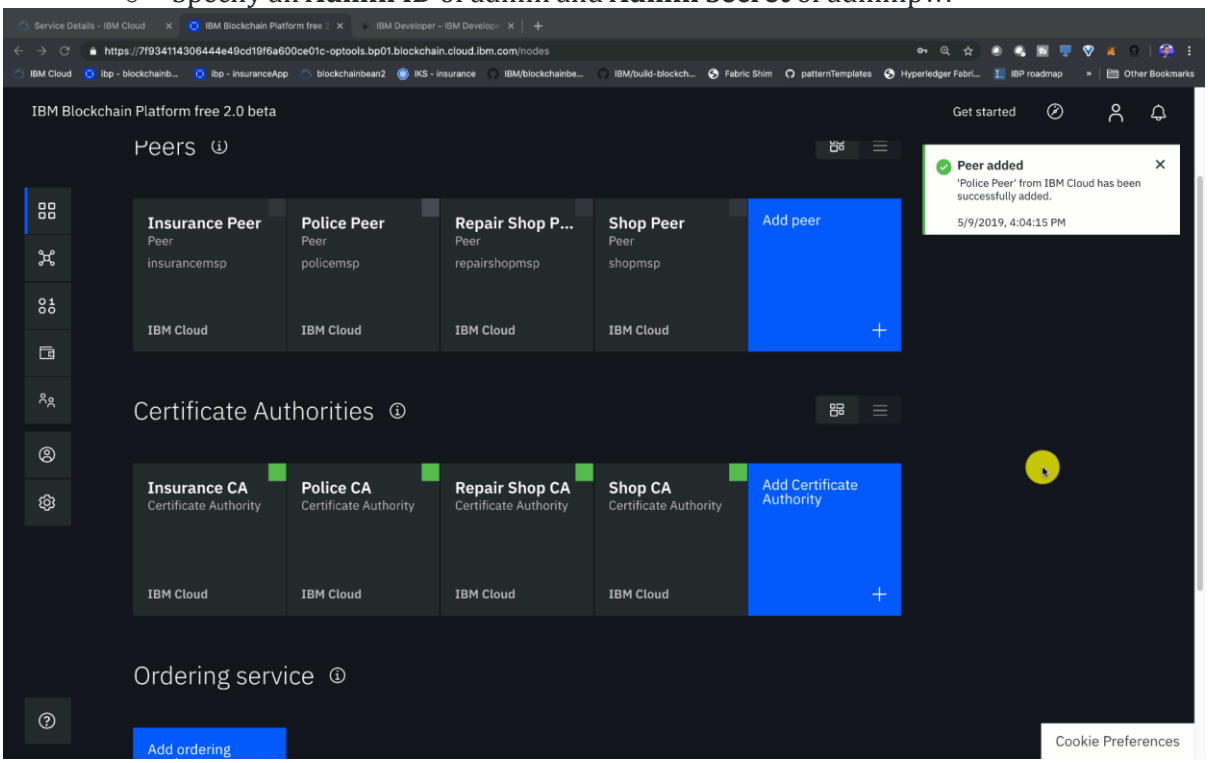
- Create a repair shop peer (same process as shown in gif above)
  - On the **Nodes** page, click **Add peer**.
  - Click **IBM Cloud** under Create a new peer and **Next**.
  - Give your peer a **Display name** of Repair Shop Peer.
  - On the next screen, select Repair Shop CA as your **Certificate Authority**. Then, give the **Enroll ID** and **Enroll secret** for the peer identity that you created for your peer, repairShopPeer, and repairShopPeerpw. Then, select the **Administrator Certificate (from MSP)**, Repair Shop MSP, from the drop-down list and click **Next**.
  - Give the **TLS Enroll ID**, admin, and **TLS Enroll secret**, adminpw, the same values are the Enroll ID and Enroll secret that you gave when creating the CA. Leave the **TLS CSR hostname** blank.

    - The last side panel will ask you to **Associate an identity** and make it the admin of your peer. Select your peer admin identity Repair Shop Admin.
    - Review the summary and click **Submit**.

- Create a police peer (same process as shown in gif above)
    - On the **Nodes** page, click **Add peer**.
    - Click **IBM Cloud** under Create a new peer and **Next**.
    - Give your peer a **Display name** of Police Peer.
    - On the next screen, select Police CA as your **Certificate Authority**. Then, give the **Enroll ID** and **Enroll secret** for the peer identity that you created for your peer, policePeer, and policePeerpw. Then, select the **Administrator Certificate (from MSP)**, Police MSP, from the drop-down list and click **Next**.
    - Give the **TLS Enroll ID**, admin, and **TLS Enroll secret**, adminpw, the same values are the Enroll ID and Enroll secret that you gave when creating the CA. Leave the **TLS CSR hostname** blank.
    - The last side panel will ask you to **Associate an identity** and make it the admin of your peer. Select your peer admin identity Police Admin.
    - Review the summary and click **Submit**.

Step 5. Build a network - Create Orderer

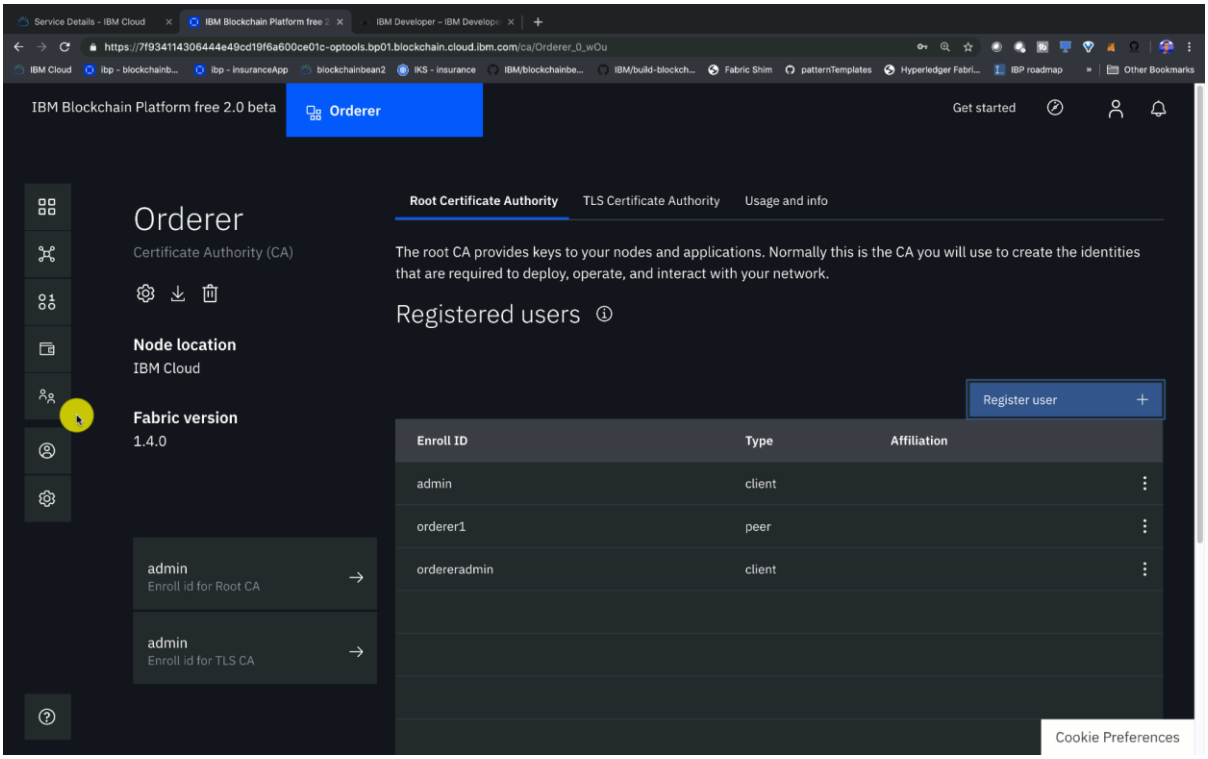
- *Create your orderer organization CA*
  - Click **Add Certificate Authority**.
  - Click **IBM Cloud** under **Create Certificate Authority** and **Next**.
  - Give it a unique **Display name** of Orderer CA.
  - Specify an **Admin ID** of admin and **Admin Secret** of adminpw.



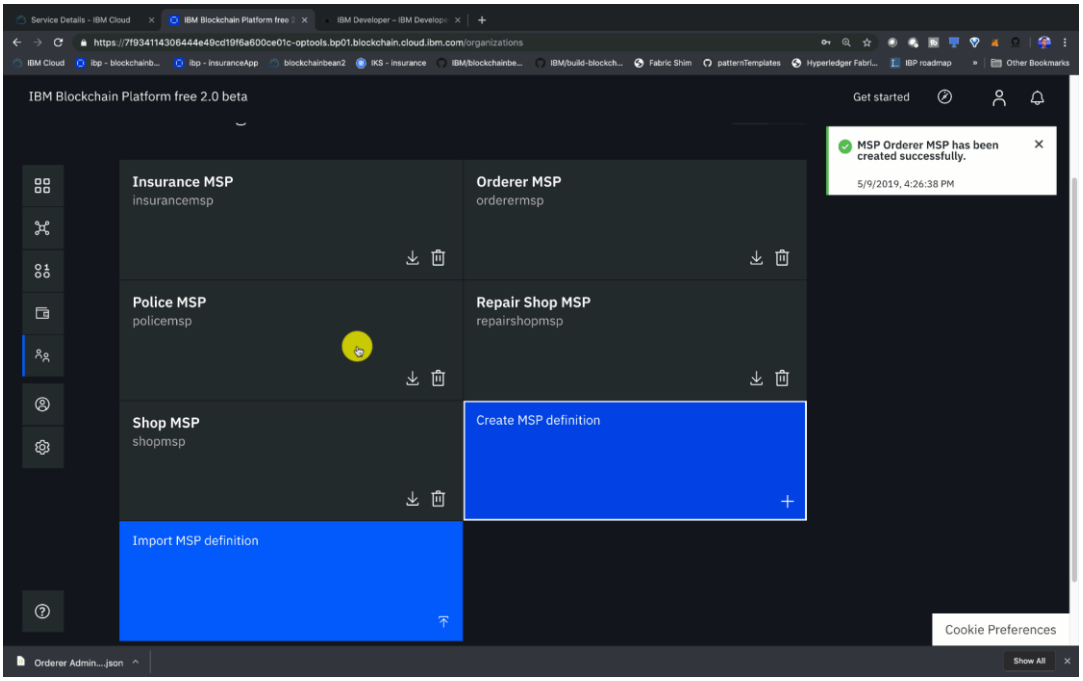
- *Use your CA to register orderer and orderer admin identities (shown in gif above)*
  - In the **Nodes** tab, select the **Orderer CA** Certificate Authority that we created.
  - First, we will register an admin for our organization. Click on the **Register User** button. Give an **Enroll ID** of ordereradmin, and **Enroll Secret** of ordereradminpw. Click **Next**. Set the **Type** for this identity as client and select org1 from the affiliated organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.
  - We will repeat the process to create an identity of the orderer. Click on the **Register User** button. Give an **Enroll ID** of orderer1, and **Enroll Secret** of orderer1pw. Click **Next**. Set the **Type** for this identity as peer and select org1 from the affiliated organizations drop-down list. We will leave the **Maximum enrollments** and **Add Attributes** fields blank.

- *Create the orderer organization MSP definition*
  - Navigate to the **Organizations** tab in the left navigation and click **Create MSP definition**.
  - Enter the **MSP Display name** as Orderer MSP and an **MSP ID** of ordererMSP.
  - Under **Root Certificate Authority** details, specify the peer CA that we created Orderer CA as the root CA for the organization.

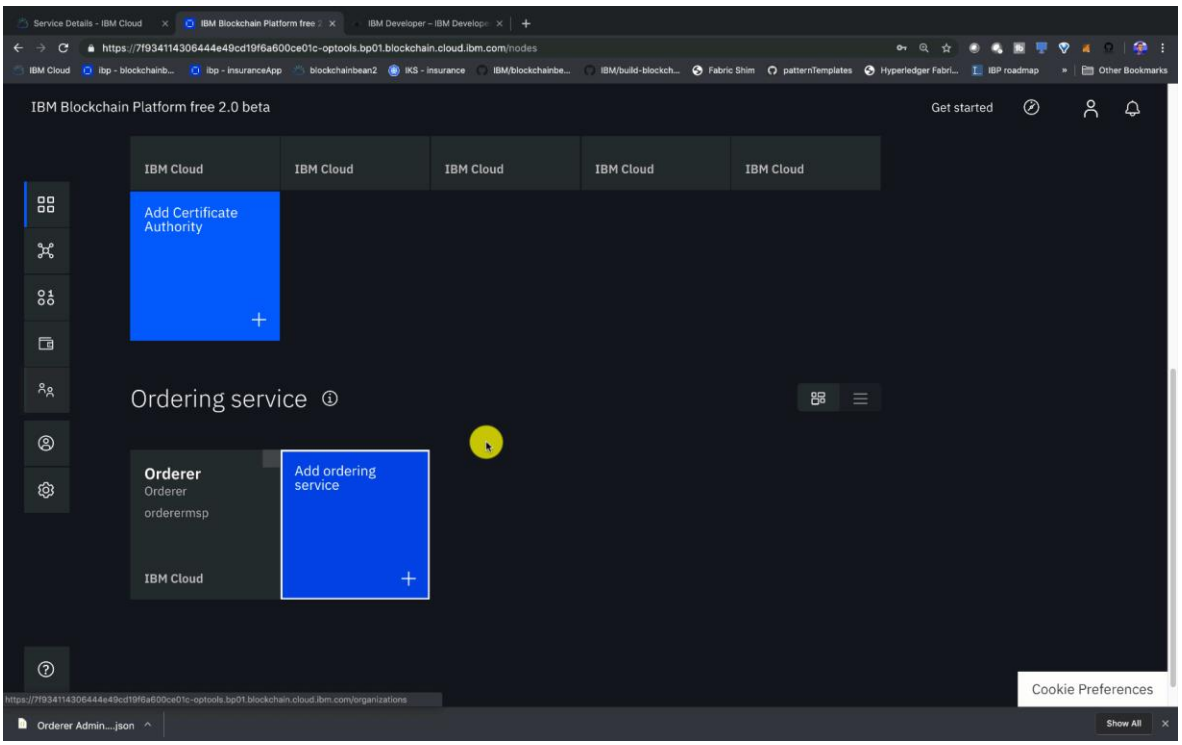
- Give the **Enroll ID** and **Enroll secret** for your organization admin, ordereradmin and ordereradminpw. Then, give the **Identity name**, Orderer Admin.
- Click the **Generate** button to enroll this identity as the admin of your organization and export the identity to the wallet. Click **Export** to export the admin certificates to your file system. Finally click **Create MSP definition**.



- *Create an orderer*
  - On the **Nodes** page, click **Add orderer**.
  - Click **IBM Cloud** and proceed with **Next**.
  - Give your peer a **Display name** of Orderer.
  - On the next screen, select Orderer CA as your **Certificate Authority**. Then, give the **Enroll ID** and **Enroll secret** for the peer identity that you created for your orderer, orderer1, and orderer1pw. Then, select the **Administrator Certificate (from MSP)**, Orderer MSP, from the drop-down list and click **Next**.
  - Give the **TLS Enroll ID**, admin, and **TLS Enroll secret**, adminpw, the same values are the Enroll ID and Enroll secret that you gave when creating the CA. Leave the **TLS CSR hostname** blank.
  - The last side panel will ask to **Associate an identity** and make it the admin of your peer. Select your peer admin identity Orderer Admin.
  - Review the summary and click **Submit**.

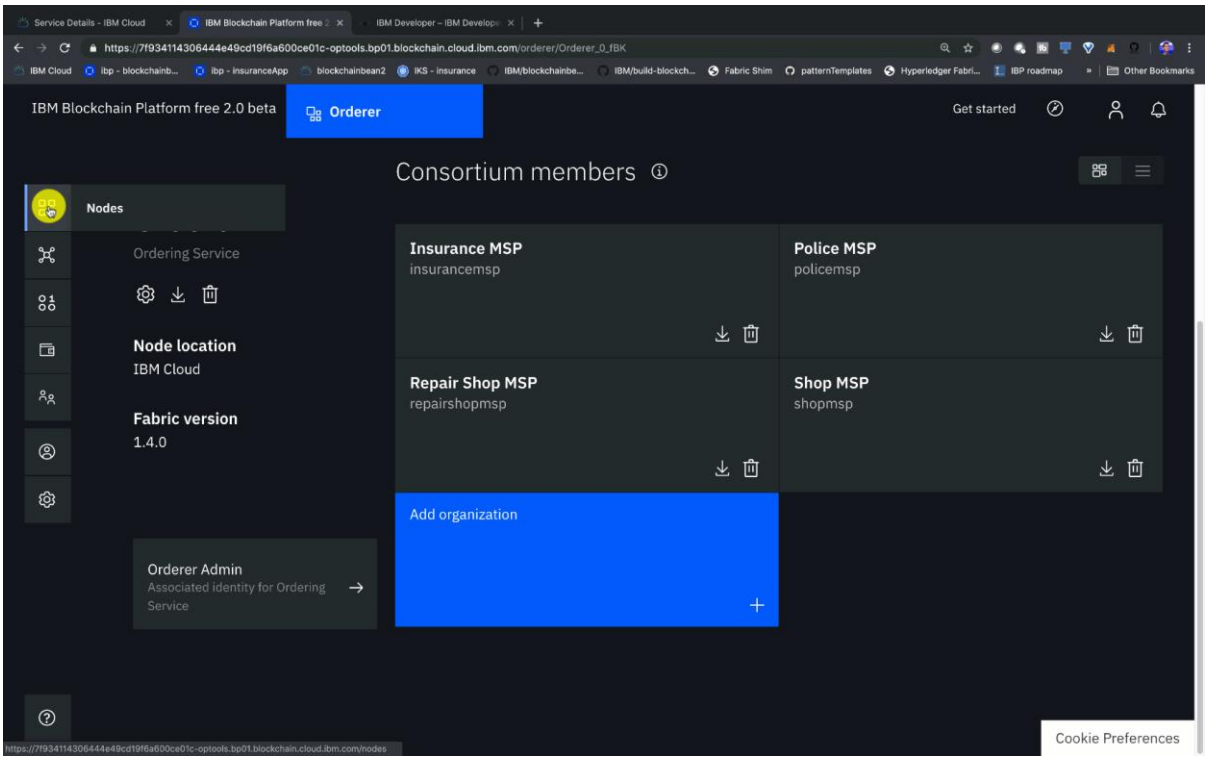


- *Add organizations as Consortium Member on the orderer to transact*
    - Navigate to the **Nodes** tab, and click on the **Orderer** that we created.
    - Under **Consortium Members**, click **Add organization**.
    - From the drop-down list, select Insurance MSP.
    - Click **Submit**.
    - Repeat the same steps, but add Shop MSP, Repair Shop MSP, and Police MSP as well.



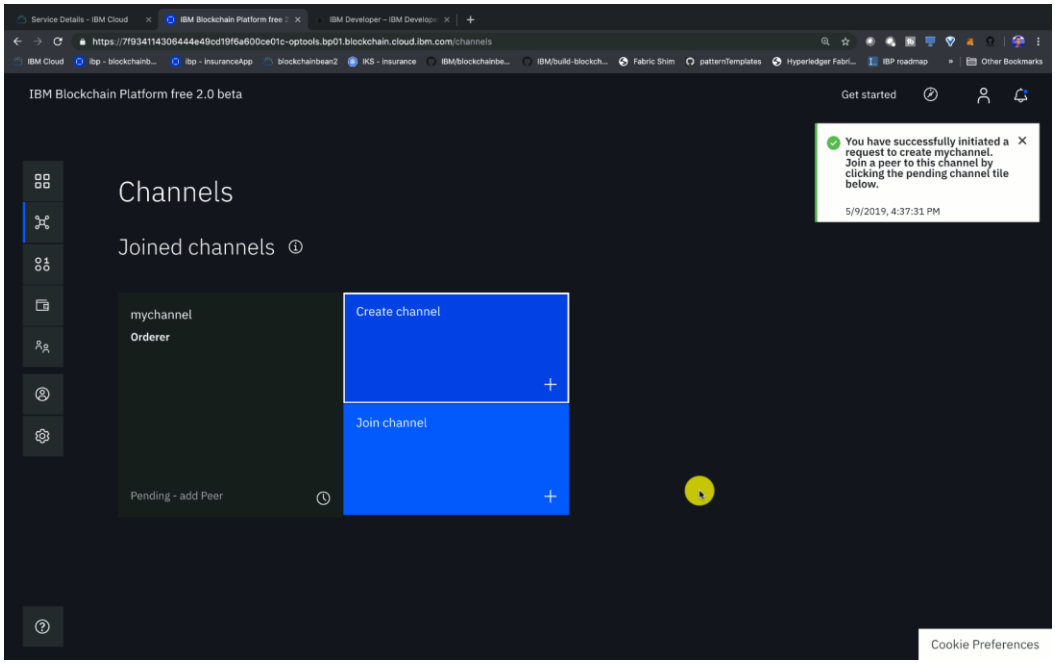
Step 6. Build a network - Create and Join Channel

- *Create the channel*
  - Navigate to the **Channels** tab in the left navigation.
  - Click **Create channel**.
  - Give the channel a name, mychannel.
  - Select the orderer you created, Orderer from the orderers drop-down list.
  - Select the MSP identifying the organization of the channel creator from the drop-down list. This should be Insurance MSP (insurancemsp).
  - Associate available identity as Insurance Admin.
  - Click **Add** next to the insurance organization. Make the insurance organization an **Operator**.
  - Click **Add** next to the shop organization. Make the shop organization an **Operator**.
  - Click **Add** next to the repair shop organization. Make the repair shop organization an **Operator**.
  - Click **Add** next to the police organization. Make the insurance organpoliceization an **Operator**.
  - Click **Create**.

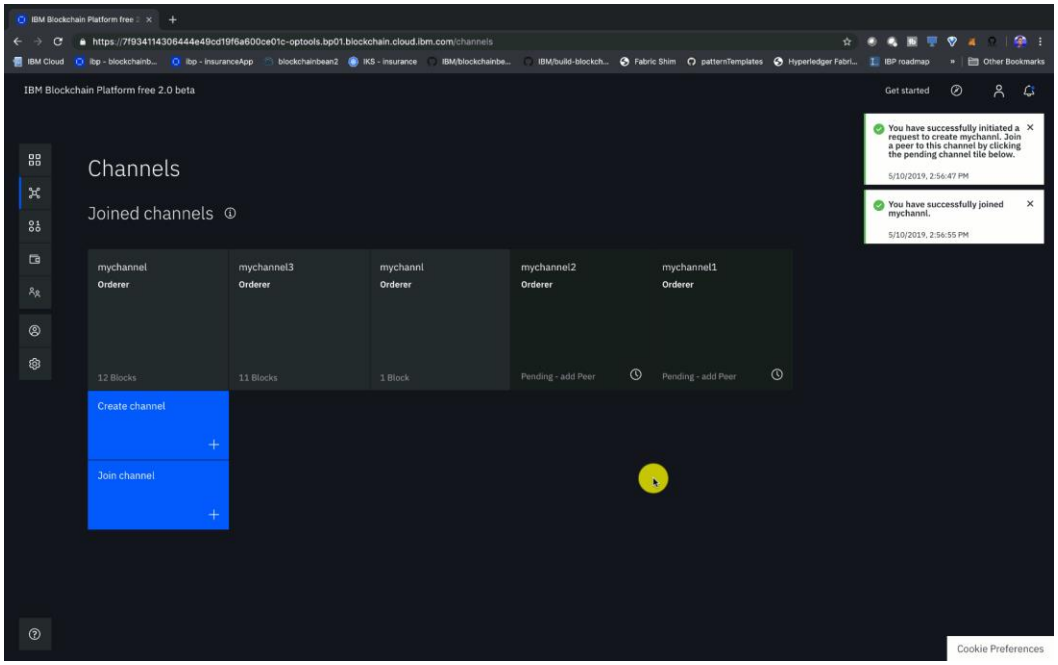


- *Join your peer to the channel*
  - Click **Join channel** to launch the side panels.
  - Select your Orderer and click **Next**.
  - Enter the name of the channel you just created. mychannel and click **Next**.

- Select which peers you want to join the channel, click Insurance Peer, Shop Peer, Repair Shop Peer, and Police Peer.
- Click **Submit**.

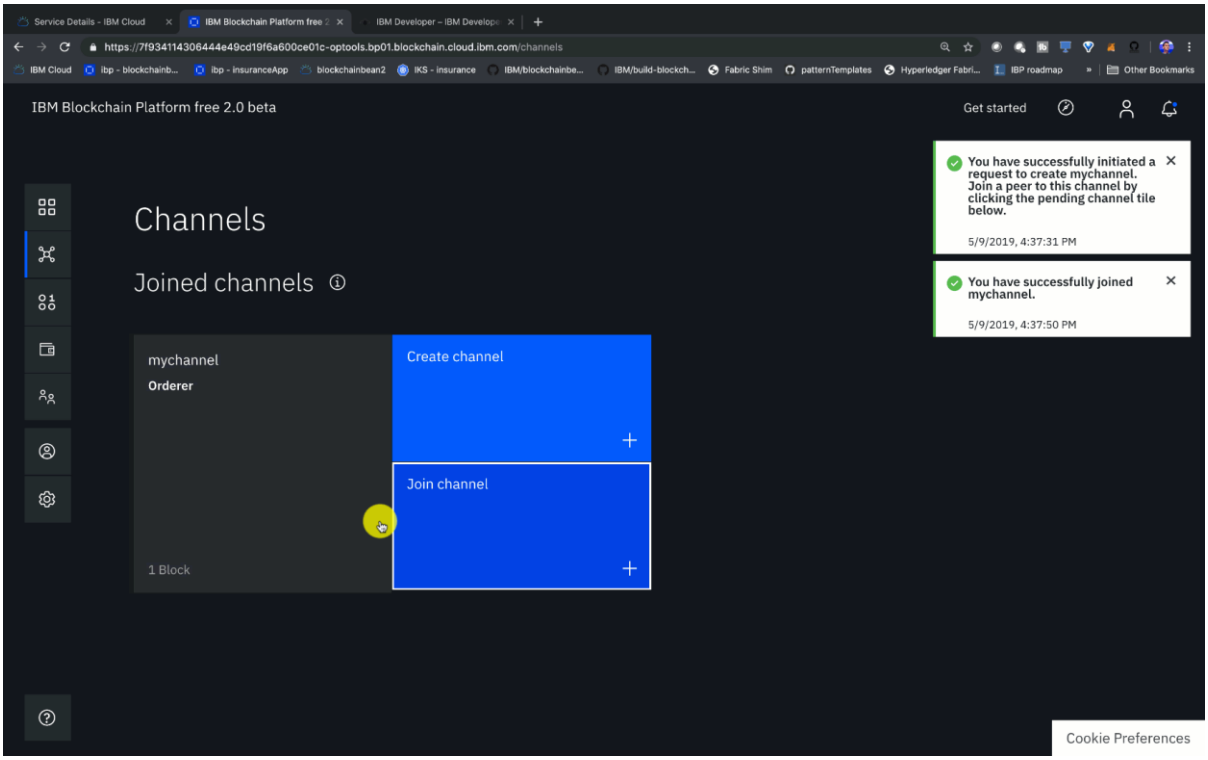


- *Add anchor peers to the channel*
  - In order to communicate between organizations, we need to enroll anchor peers.
  - From the channels tab, click on the channel you have created, mychannel.
  - From the channel overview page, click on channel details. Scroll all the way down until you see Anchor peers.
  - Click Add anchor peer and add the Insurance, Police, Repair Shop, and Shop peers.
  - Select which peers you want to join the channel, click Insurance Peer, Shop Peer, Repair Shop Peer, and Police Peer.
  - Click **Add anchor peer**.
  - If all went well, your channel Anchor peers should look like below:



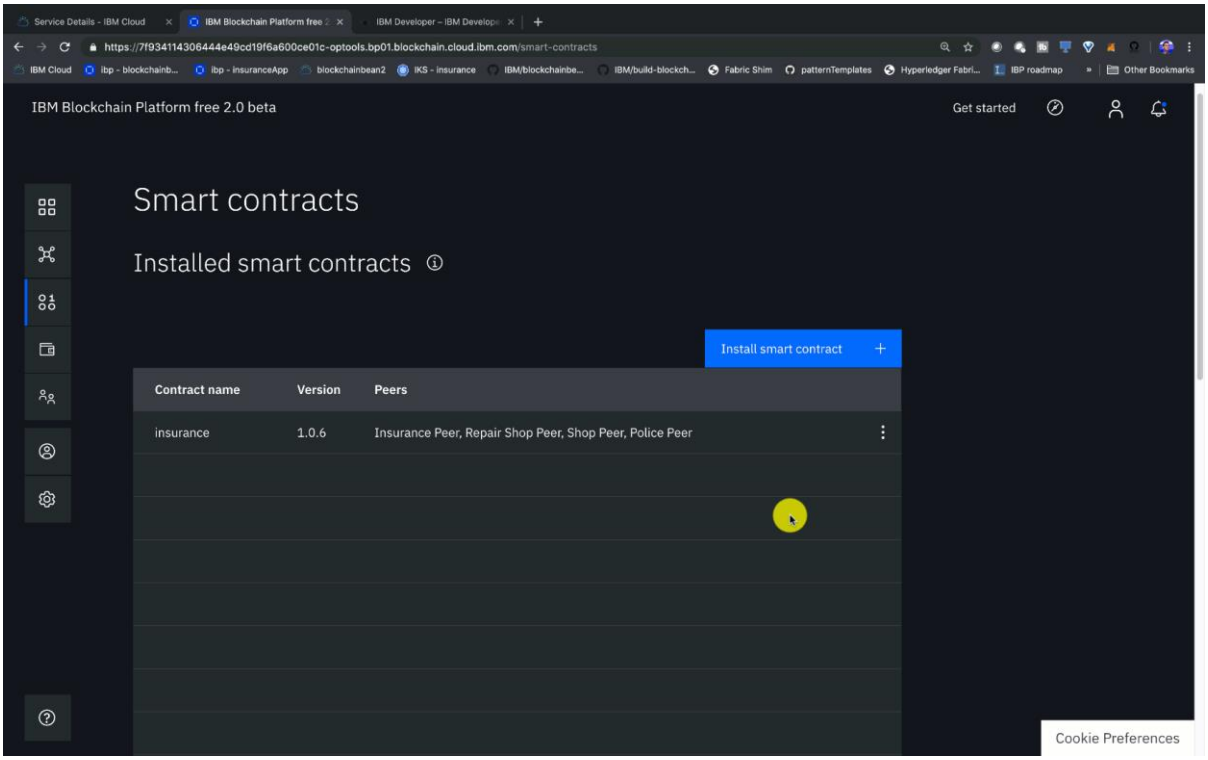
Step 7. Deploy Insurance Smart Contract on the network

- *Install a smart contract*
- Clone the repository:
  git clone https://github.com/IBM/build-blockchain-insurance-app
  - Click the **Smart contracts** tab to install the smart contract.
  - Click **Install smart contract** to upload the insurance smart contract package file.
  - Click on **Add file** and find your packaged smart contract. It is the file in the build-blockchain-insurance-app/chaincodePackage directory.
  - Select all peers - we need to install the contract on each peer.
  - Once the contract is uploaded, click **Install**.



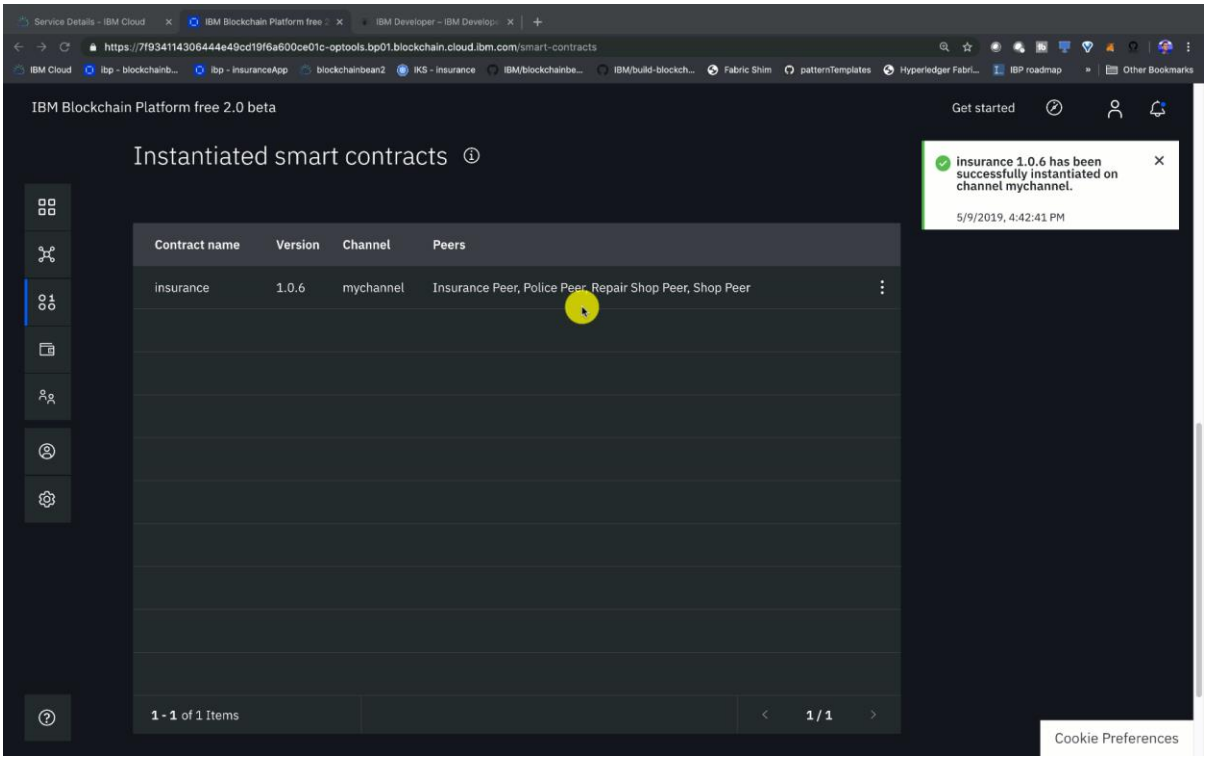
Instantiate smart contract

- On the smart contracts tab, find the smart contract from the list installed on your peers and click **Instantiate** from the overflow menu on the right side of the row.
- On the side panel that opens, select the channel, mychannel to instantiate the smart contract on. Click **Next**.
- Select the organization members to be included in the policy, insurancemsp, shopmsp, repairshopmsp, policemsp. Click **Next**.
- Give **Function name** of Init and leave **Arguments** blank.
- Click **Instantiate**.



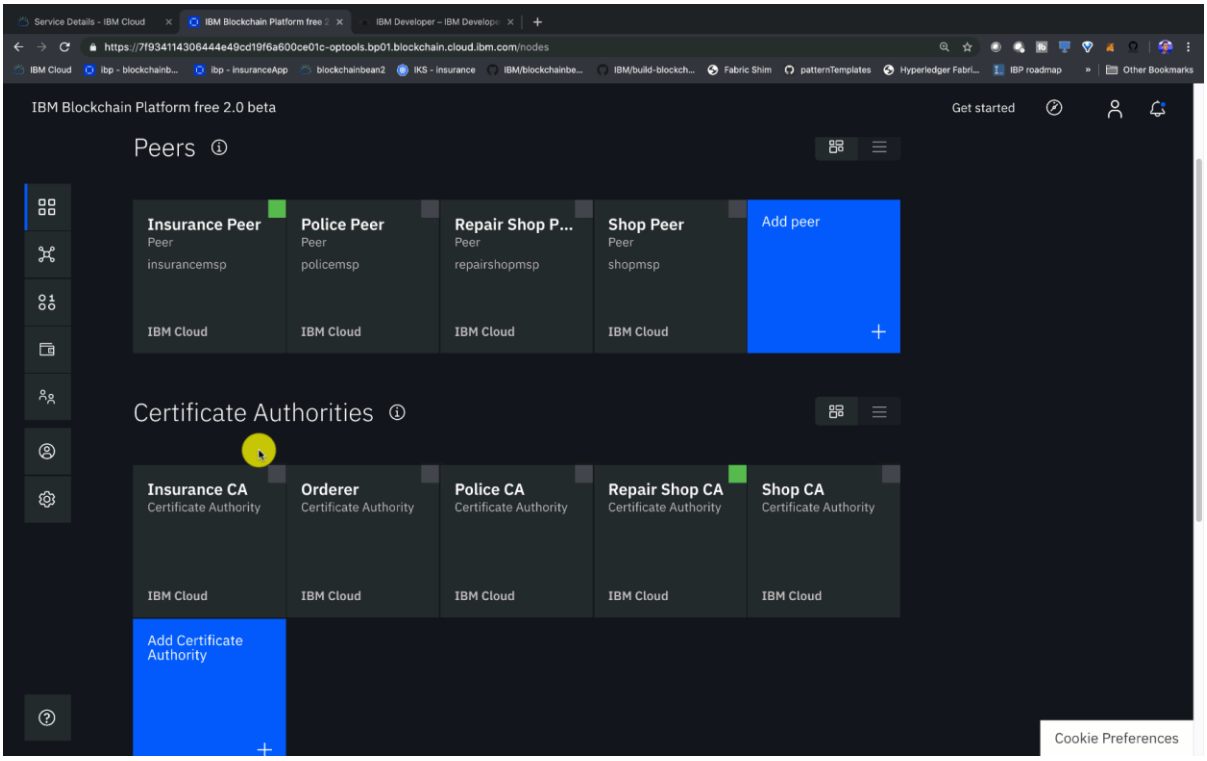
Step 8. Connect application to the network

- *Connect with sdk through connection profile*
    - Under the Instantiated Smart Contract, click on Connect with SDK from the overflow menu on the right side of the row.
    - Choose from the dropdown for **MSP for connection**, insurancemsp.
    - Choose from **Certificate Authority** dropdown, Insurance CA.
    - Download the connection profile by scrolling down and clicking **Download Connection Profile**. This will download the connection json which we will use soon to establish connection.
    - You can click **Close** once the download completes.



- *Create insurance application admin*

    - Go to the **Nodes** tab on the left bar, and under **Certificate Authorities**, choose your **Insurance CA**.
    - Click on **Register user**.
    - Give an **Enroll ID** and **Enroll Secret** to administer your application users, insuranceApp-admin and insuranceApp-adminpw.
    - Choose client as **Type**.
    - You can leave the **Use root affiliation** box checked.
    - You can leave the **Maximum enrollments** blank.
    - Under **Attributes**, click on **Add attribute**. Give attribute as hf.Registrar.Roles = *. This will allow this identity to act as registrar and issues identities for our app. Click **Add-attribute**.
    - Click **Register**.



- *Create shop application admin (same process as shown above in the gif)*
    - Go to the **Nodes** tab on the left bar, and under **Certificate Authorities**, choose your **Shop CA**.
    - Click on **Register user**.
    - Give an **Enroll ID** and **Enroll Secret** to administer your application users, shopApp-admin and shopApp-adminpw.
    - Choose client as **Type**.
    - You can leave the **Use root affiliation** box checked.
    - You can leave the **Maximum enrollments** blank.
    - Under **Attributes**, click on **Add attribute**. Give attribute as hf.Registrar.Roles = *. This will allow this identity to act as registrar and issues identities for our app. Click **Add-attribute**.
    - Click **Register**.

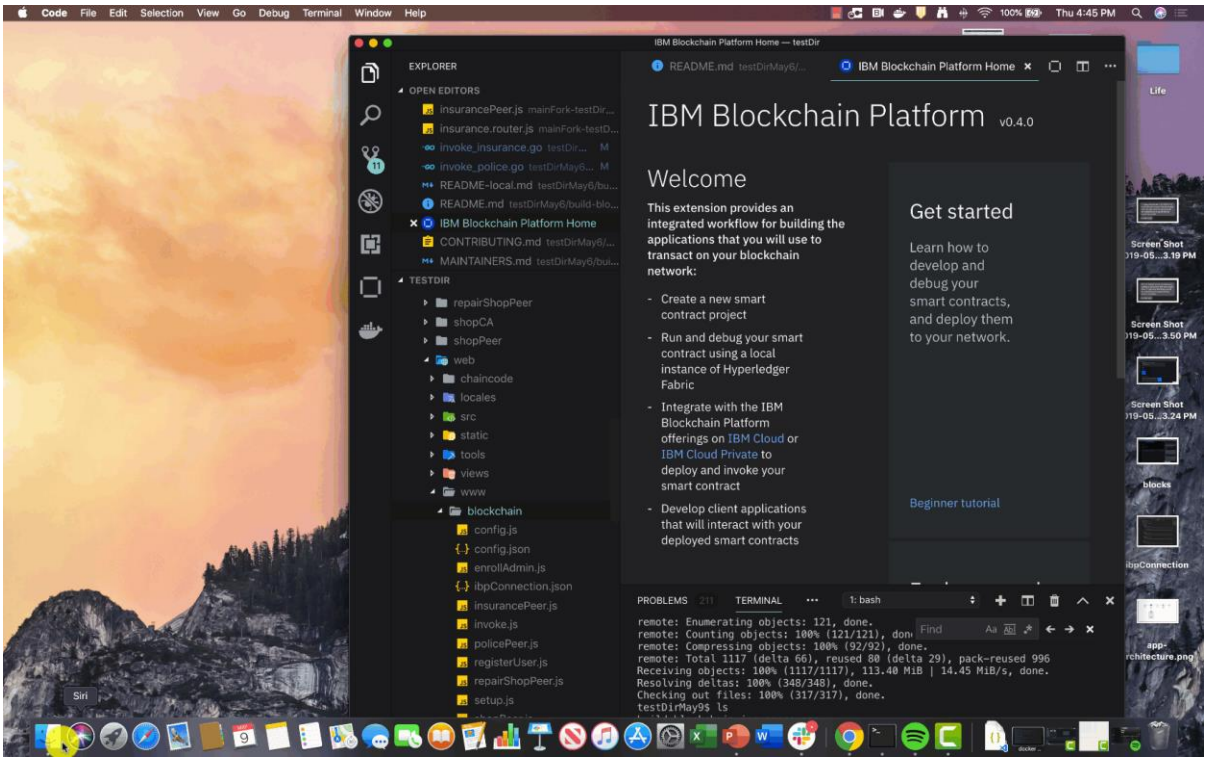
- *Create repair shop application admin (same process as shown above in the gif)*
    - Go to the **Nodes** tab on the left bar, and under **Certificate Authorities**, choose your **Repair Shop CA**.
    - Click on **Register user**.
    - Give an **Enroll ID** and **Enroll Secret** to administer your application users, repairShopApp-admin and repairShopApp-adminpw.
    - Choose client as **Type**.
    - You can leave the **Use root affiliation** box checked.
    - You can leave the **Maximum enrollments** blank.

- Under **Attributes**, click on **Add attribute**. Give attribute as hf.Registrar.Roles = *. This will allow this identity to act as registrar and issues identities for our app. Click **Add-attribute**.
- Click **Register**.

- *Create police application admin (same process as shown above in the gif)*
  - Go to the **Nodes** tab on the left bar, and under **Certificate Authorities**, choose your **Police CA**.
  - Click on **Register user**.
  - Give an **Enroll ID** and **Enroll Secret** to administer your application users, policeApp-admin and policeApp-adminpw.
  - Choose client as **Type**.
  - You can leave the **Use root affiliation** box checked.
  - You can leave the **Maximum enrollments** blank.
  - Under **Attributes**, click on **Add attribute**. Give attribute as hf.Registrar.Roles = *. This will allow this identity to act as registrar and issues identities for our app. Click **Add-attribute**.
  - Click **Register**.

Update application connection

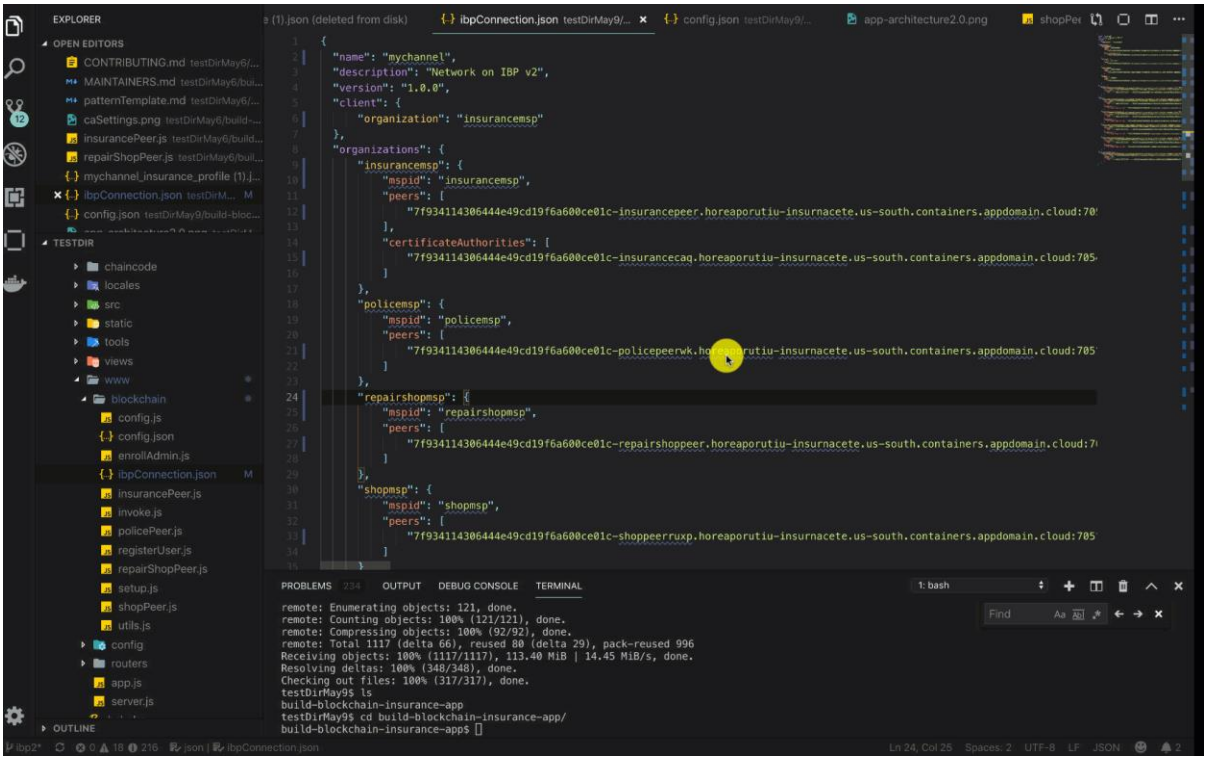
- Copy the connection profile you downloaded into the web/www/blockchain directory.
- Copy and paste everything in the connection profile, and overwrite the **ibpConnection.json**.



Step 9. Enroll App Admin Identities

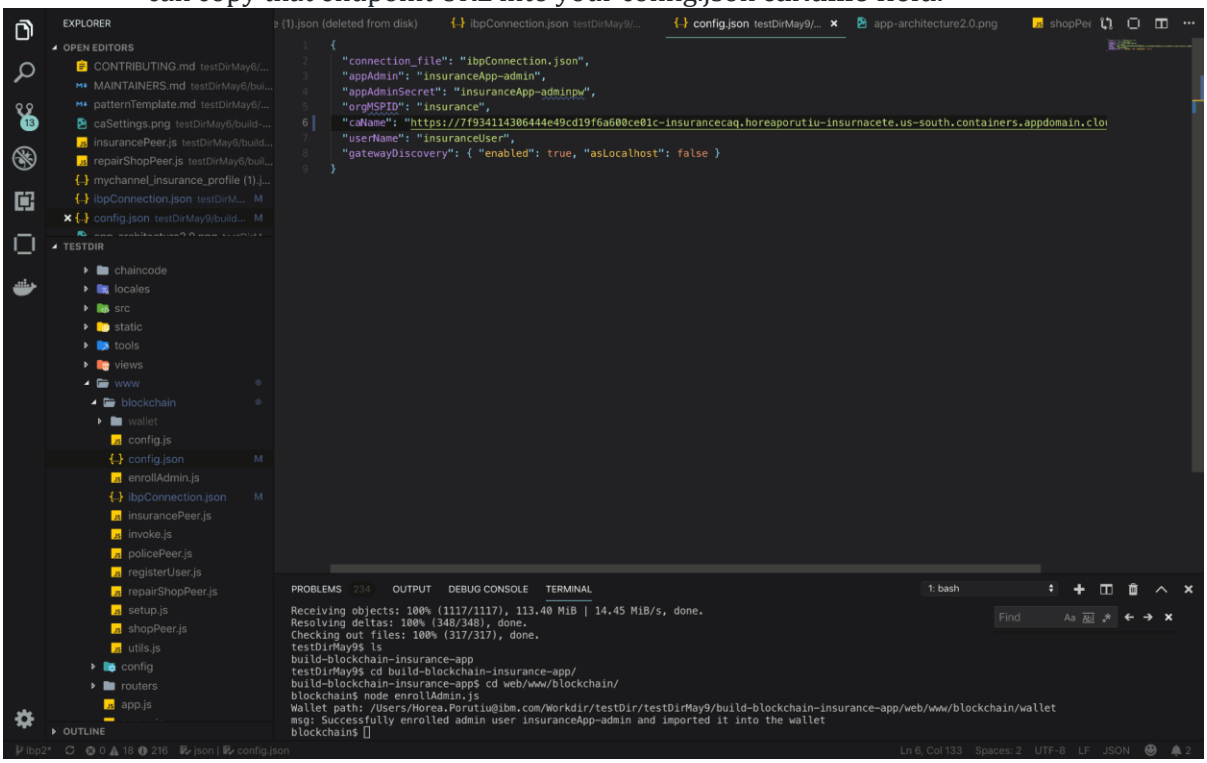
- *Enroll insurnaceApp-admin*
    - First, navigate to the web/www/blockchain directory.
      cd web/www/blockchain/
    - Open the config.json file, and update the caName with the URL of the **insurance** certificate authority from your ibpConnection.json file. Save the file.
    - Run the enrollAdmin.js script
      node enrollAdmin.js

    - You should see the following in the terminal:
      msg: Successfully enrolled admin user insuranceApp-admin and imported it into the wallet



- Enroll shopApp-admin
    - First, change the appAdmin, appAdminSecret, and caName properties in your config.json file, so that it looks something like this (your caName should be different than mine):
    - {
    - "connection_file": "ibpConnection.json",
    - "appAdmin": "shopApp-admin",
    - "appAdminSecret": "shopApp-adminpw",
    - "orgMSPID": "shopmsp",
    - "caName": "https://fa707c454921423c80ec3c3c38d7545c-caf2e287.horeainsurancetest.us-south.containers.appdomain.cloud:7054",
    - "userName": "shopUser",

- "gatewayDiscovery": { "enabled": true, "asLocalhost": false }
  }
- To find the other CA urls, you will need to click on the Nodes tab in IBM Blockchain Platform, then on the Shop CA, and on the settings cog icon at the top of the page. That will take you to the certificate authority settings, as shown in the picture below, and you can copy that endpoint URL into your config.json **caName** field.



- Run the enrollAdmin.js script
  node enrollAdmin.js

- You should see the following in the terminal:

  msg: Successfully enrolled admin user shopApp-admin and imported it into the wallet

- *Enroll repairShopApp-admin (same process as shown in gif above)*

  - First, change the appAdmin, appAdminSecret, and caName properties in your config.json file, so that it looks something like this (your caName should be different than mine):
  - {
  - "connection_file": "ibpConnection.json",
  - "appAdmin": "repairShopApp-admin",
  - "appAdminSecret": "repairShopApp-adminpw",
  - "orgMSPID": "repairshopmsp",
  - "caName": "https://fa707c454921423c80ec3c3c38d7545c-caf2e287.horeainsurancetest.us-south.containers.appdomain.cloud:7054",
  - "userName": "repairUser",
  - "gatewayDiscovery": { "enabled": true, "asLocalhost": false }

}
- Run the enrollAdmin.js script
  node enrollAdmin.js

- You should see the following in the terminal:

  msg: Successfully enrolled admin user repairShopApp-admin and imported it into the wallet

- *Enroll policeApp-admin (same process as shown in gif above)*

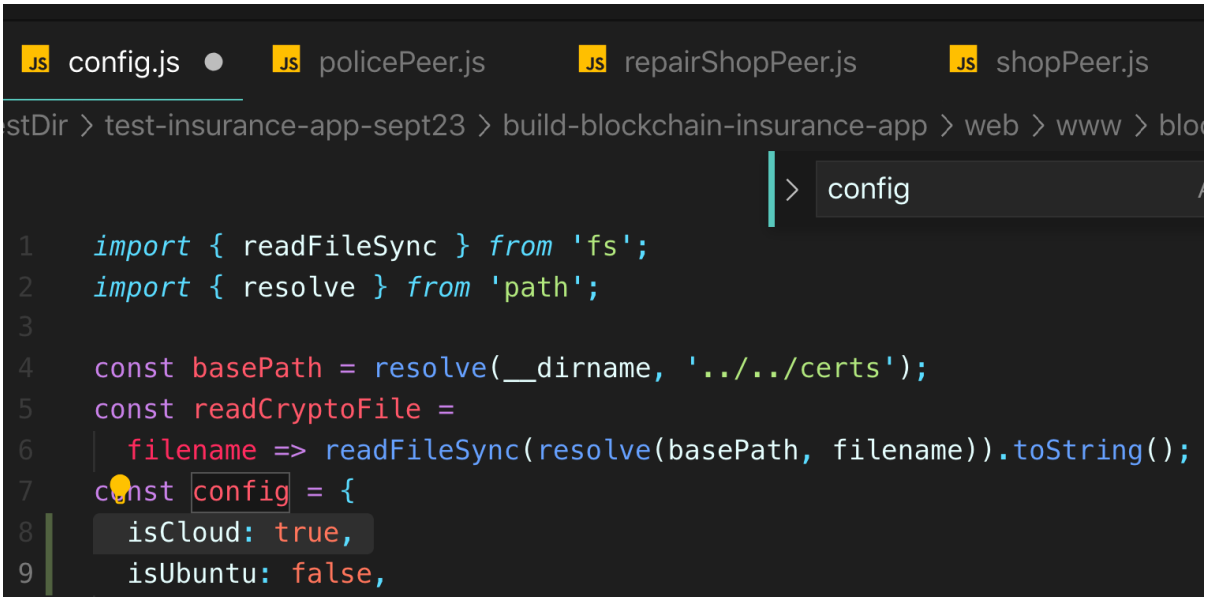
  - First, change the appAdmin, appAdminSecret, and caName properties in your config.json file, so that it looks something like this (your caName should be different than mine):
  - {
  - "connection_file": "ibpConnection.json",
  - "appAdmin": "policeApp-admin",
  - "appAdminSecret": "policeApp-adminpw",
  - "orgMSPID": "policemsp",
  - "caName": "https://fa707c454921423c80ec3c3c38d7545c-caf2e287.horeainsurancetest.us-south.containers.appdomain.cloud:7054",
  - "userName": "policeUser",
  - "gatewayDiscovery": { "enabled": true, "asLocalhost": false }
    }
  - Run the enrollAdmin.js script
    node enrollAdmin.js

  - You should see the following in the terminal:

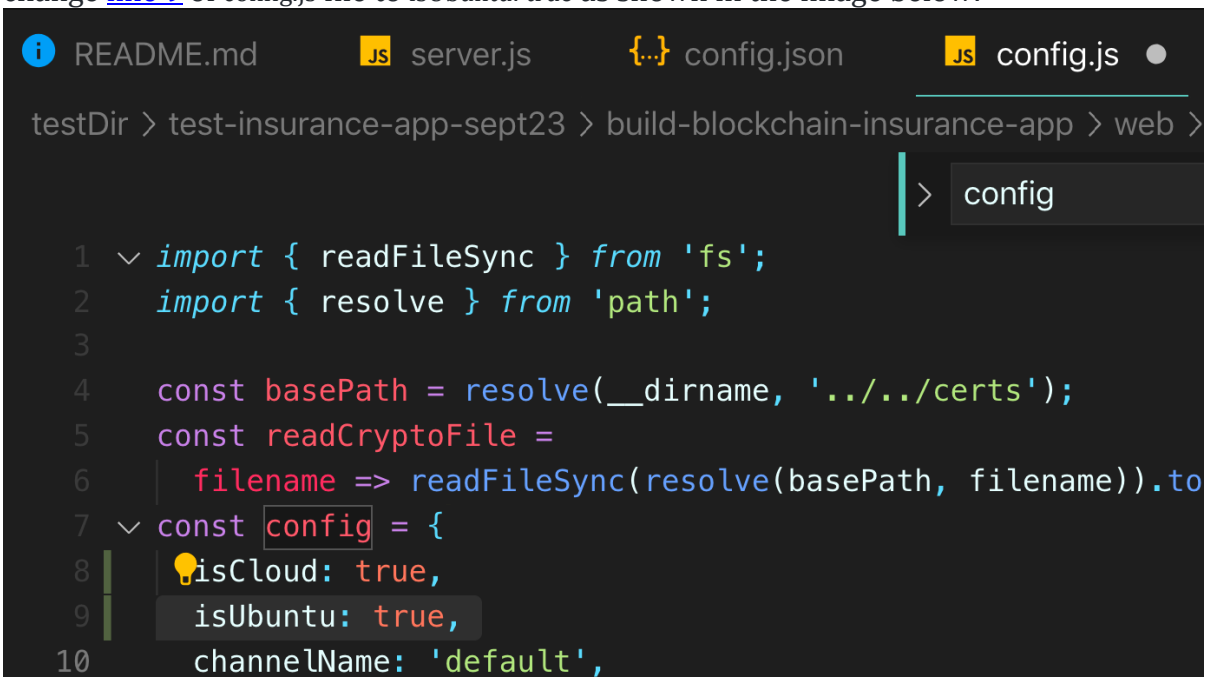
    msg: Successfully enrolled admin user policeApp-admin and imported it into the wallet

Step 10. Run the application

Navigate to the directory blockchain directory which contains the config.js file:
cd build-blockchain-insurance-app/web/www/blockchain/
In the editor of choice, change line 8 of the config.js file to isCloud: true as shown in the image below:



If you are using Mac, save the changes. Otherwise, if you are using an Ubuntu system, change line 9 of config.js file to isUbuntu: true as shown in the image below:



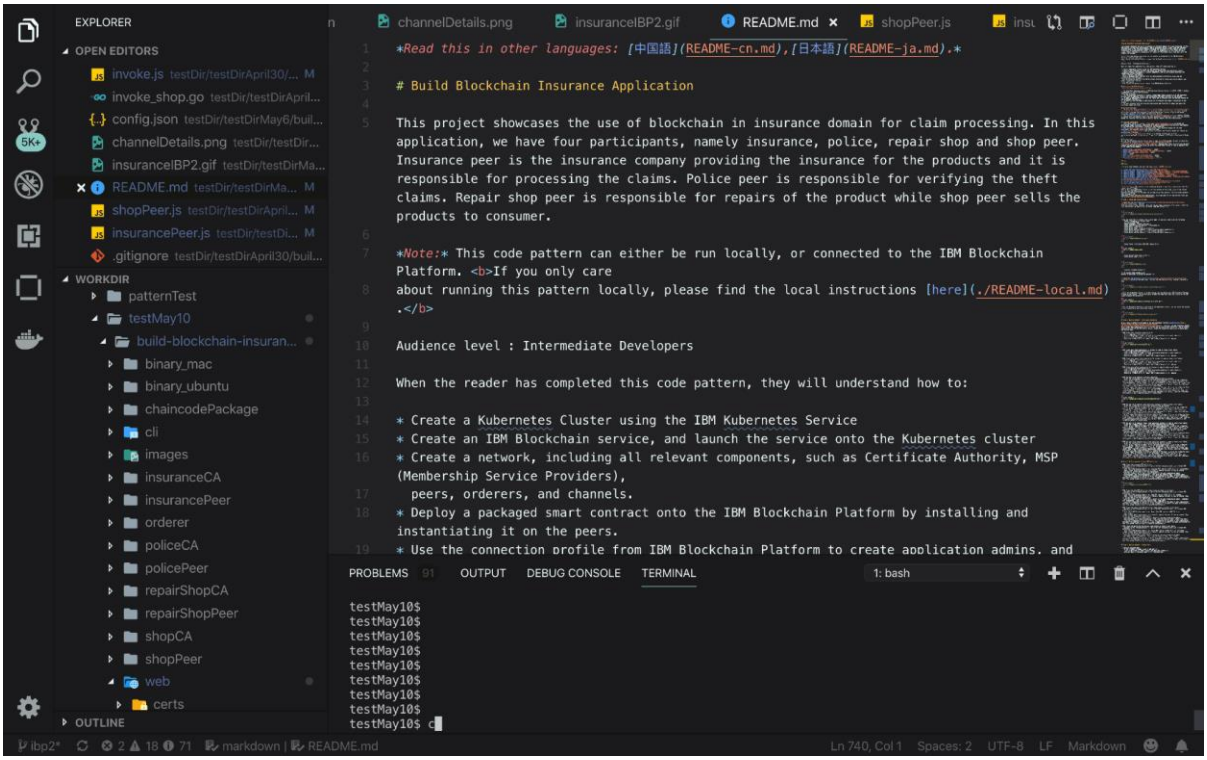
Next, from the blockchain directory navigate to the root project directory:
blockchain$ cd ../../../
build-blockchain-insurance-app$
Login using your docker hub credentials.

docker login
Run the build script to download and create docker images for the orderer, insurance-peer, police-peer, shop-peer, repairshop-peer, web application and certificate authorities for each peer. This will run for a few minutes.

For Mac user:

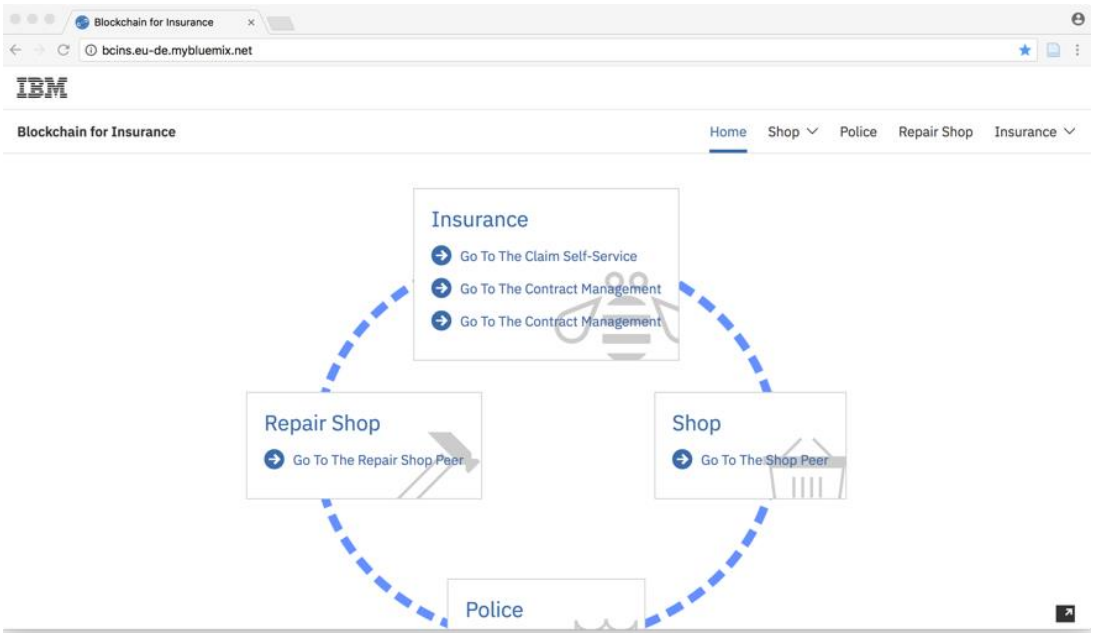
cd build-blockchain-insurance-app
./build_mac.sh
For Ubuntu user **Make sure isUbuntu:true is saved in the line 9 of config.js**:
cd build-blockchain-insurance-app
./build_ubuntu.sh
You should see the following output on console:

Creating repairshop-ca ...
Creating insurance-ca ...
Creating shop-ca ...
Creating police-ca ...
Creating orderer0 ...
Creating repairshop-ca
Creating insurance-ca
Creating police-ca
Creating shop-ca
Creating orderer0 ... done
Creating insurance-peer ...
Creating insurance-peer ... done
Creating shop-peer ...
Creating shop-peer ... done
Creating repairshop-peer ...
Creating repairshop-peer ... done
Creating web ...
Creating police-peer ...
Creating web
Creating police-peer ... done

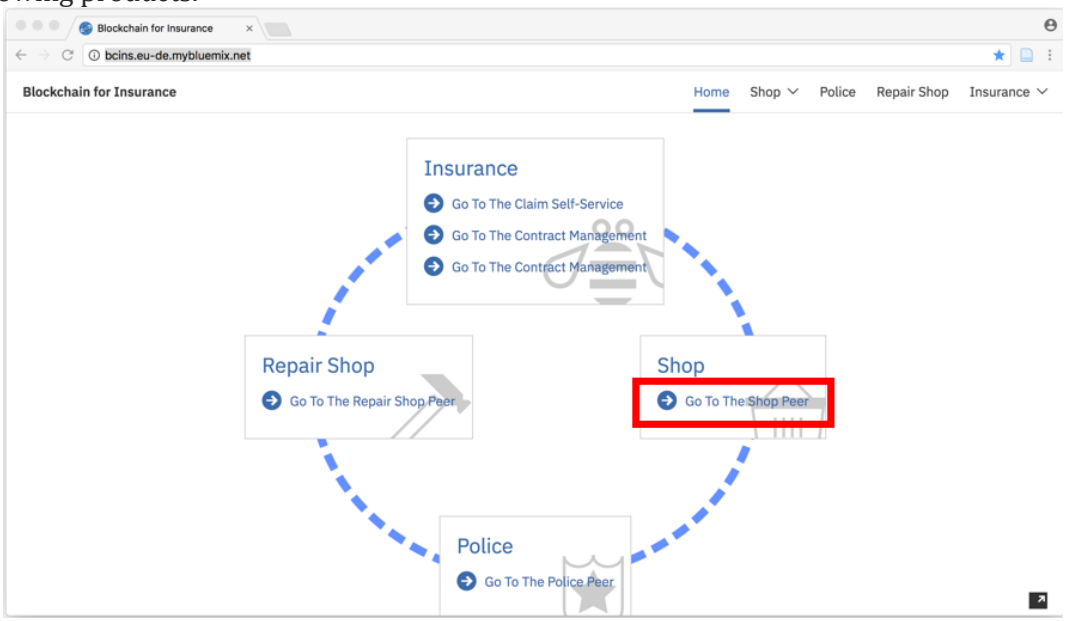


Wait for few minutes for application to install and instantiate the chaincode on network
Check the status of installation using command:
docker logs web
On completion, you should see the following output on console:
> blockchain-for-insurance@2.1.0 serve /app
> cross-env NODE_ENV=production&&node ./bin/server

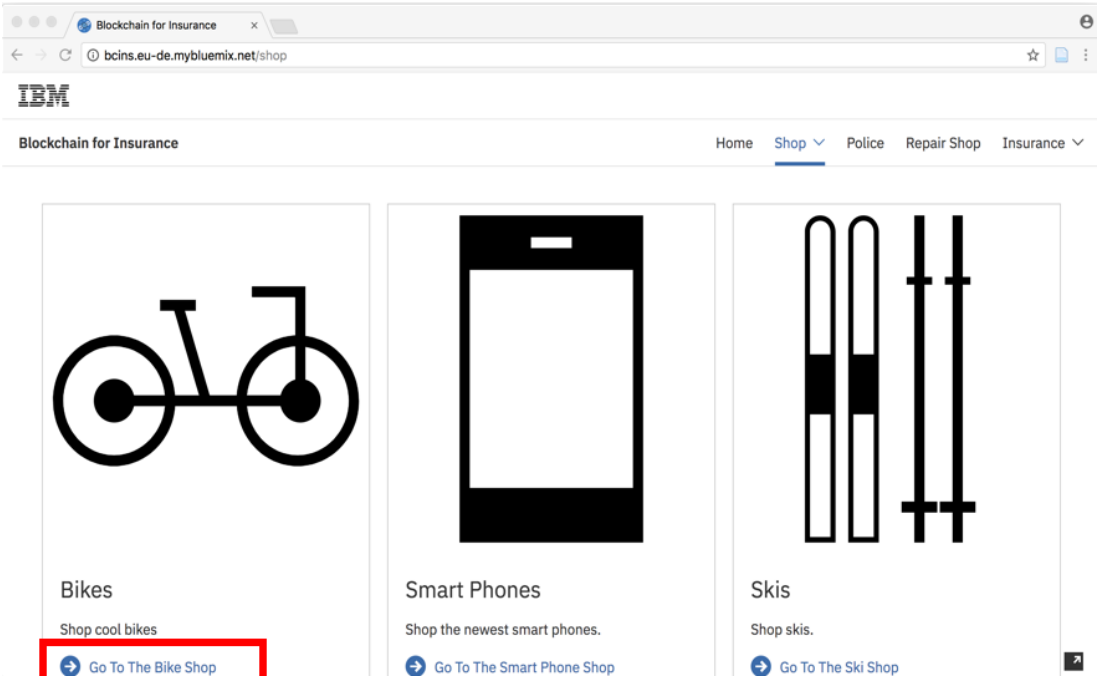
/app/app/static/js
Server running on port: 3000
Default channel not found, attempting creation...
Successfully created a new default channel.
Joining peers to the default channel.
Chaincode is not installed, attempting installation...
Base container image present.
info: [packager/Golang.js]: packaging GOLANG from bcins
info: [packager/Golang.js]: packaging GOLANG from bcins
info: [packager/Golang.js]: packaging GOLANG from bcins
info: [packager/Golang.js]: packaging GOLANG from bcins
Successfully installed chaincode on the default channel.
Successfully instantiated chaincode on all peers.
Use the link http://localhost:3000 to load the web application in browser.
The home page shows the participants (Peers) in the network. You can see that there is an Insurance, Repair Shop, Police and Shop Peer implemented. They are the participants of the network.



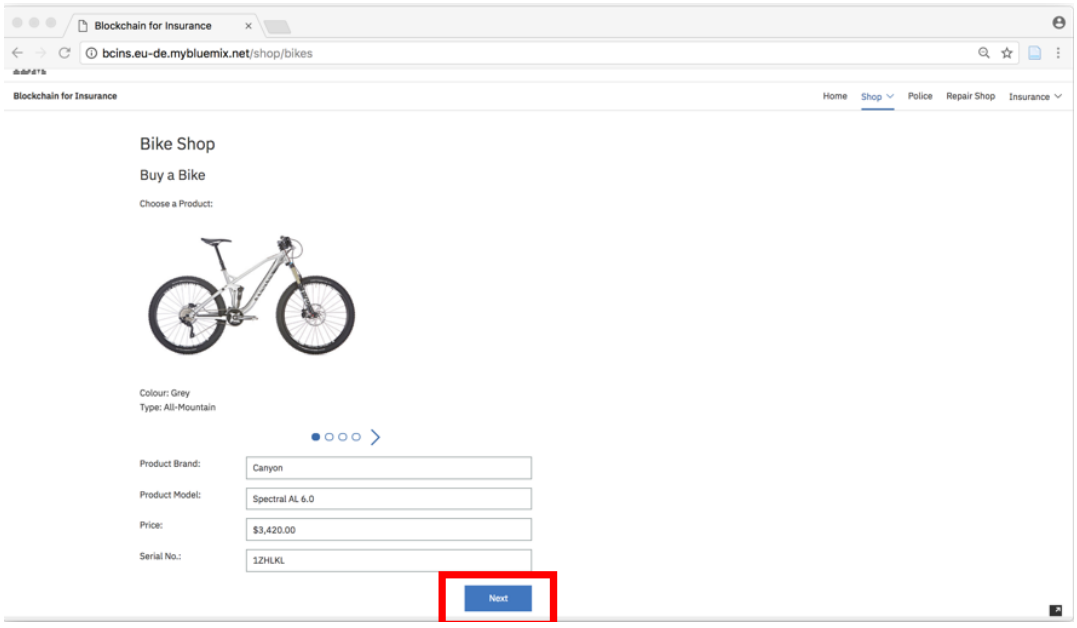
Imagine being a consumer (hereinafter called "Biker") that wants to buy a phone, bike or Ski. By clicking on the "Go to the shop" section, you will be redirected to the shop (shop peer) that offers you the following products.



You can see the three products offered by the shop(s) now. In addition, you have insurance contracts available for them. In our scenario, you are an outdoor sport enthusiast who wants to buy a new Bike. Therefore, you'll click on the Bike Shop section.

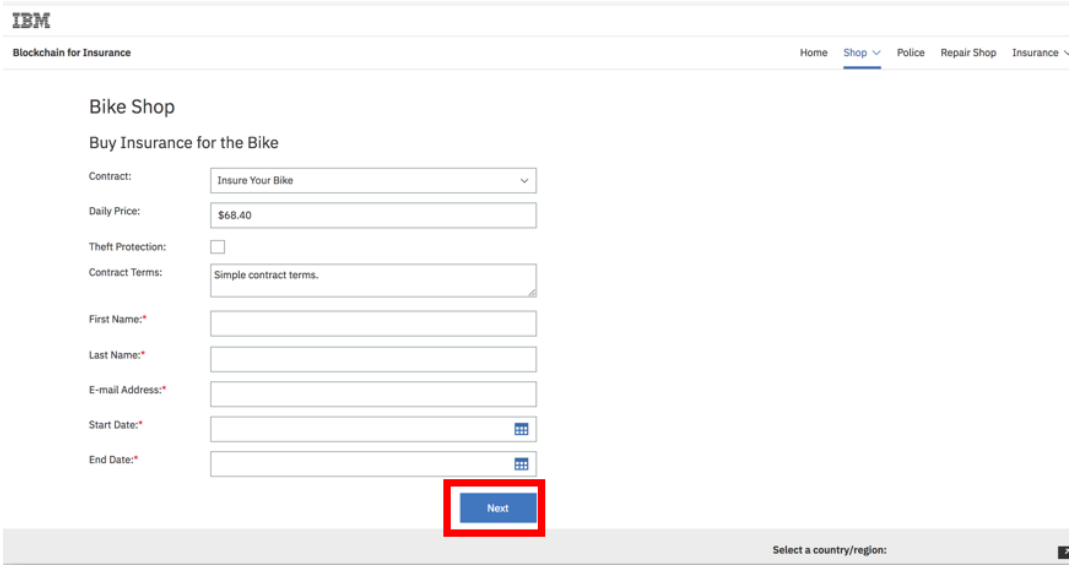


In this section, you are viewing the different bikes available in the store. You can select within four different Bikes. By clicking on next you'll be forwarded to the next page which will ask for the customer's personal data.

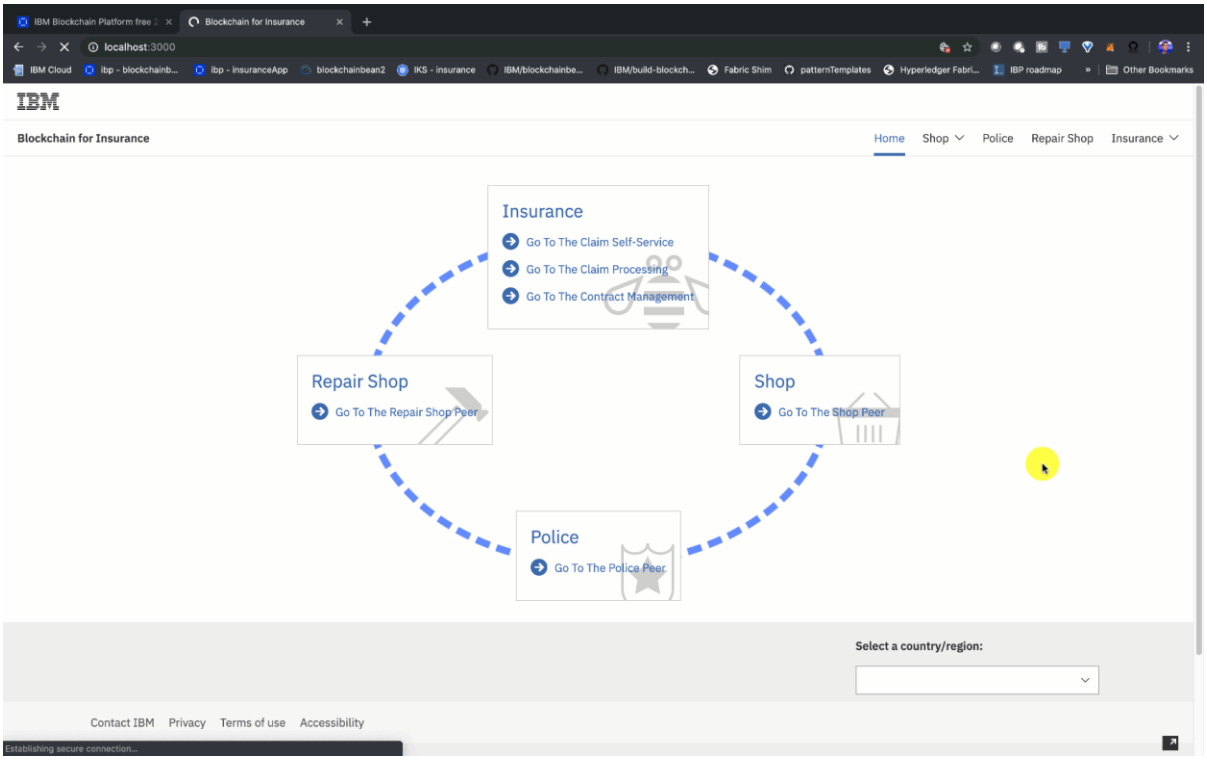


You have the choice between different insurance contracts that feature different coverage as well as terms and conditions. You are required to type-in your personal data and select a start and end date of the contract. Since there is a trend of short-term or event-driven contracts in the insurance industry you have the chance to select the duration of the contract on a daily basis. The daily price of the insurance contract is being calculated by a formula that had been defined in the chaincode. By

clicking on next you will be forwarded to a screen that summarizes your purchase and shows you the total sum.



The application will show you the total sum of your purchase. By clicking on “order” you agree to the terms and conditions and close the deal (signing of the contract). In addition, you’ll receive a unique username and password. The login credentials will be used once you file a claim. A block is being written to the Blockchain.



Congratulations! You've successfully connection your React app to the IBM Blockchain Platform! Now each time you submit transactions with the UI, they will be logged by the blockchain service.