Web3 Technologies Practical

R.D.&S.H NATIONAL COLLEGE & S.WA. SCIENCE COLLEGE

Bandra, Mumbai - 400050

DEPARTMENT OF COMPUTER SCIENCE

M.Sc. Computer Science – Semester III

Web3 Technologies Practical

JOURNAL 2025-2026

Roll No. CS24002

Seat No. 1293021

R.D. & S.H. NATIONAL COLLEGE

& S. W.A. SCIENCE COLLEGE,

Bandra, Mumbai – 400050.

Department of Computer Science

CERTIFICATE

This is to certify that **Mr. Mohd Bilal Shakir Hasan** of **M.Sc Part II (Sem III)** class has satisfactorily completed **VIII** Practicals in the subject of **Web3 Technologies Practical** as a part of M.Sc. Degree Course in Computer Science during the academic year 2025 – 2026.

Date of Submission: 26/01/2026

Faculty Incharge

Co-ordinator,

Department of Computer Science

Signature of External Examiner

INDEX

Practical No. 1

Aim - Install and understand Docker container, Node.js, Java and Hyperledger Fabric, Ethereum and perform necessary software installation on local machine/create instance on Cloud to run.

Theory – This experiment establishes the development environment required for both permissioned (enterprise) and permissionless (public) blockchain applications.

- Docker: A containerization platform used to run applications in isolated environments. In blockchain, it is critical for running multiple distinct network nodes (Peers, Orderers) on a single local machine to mimic a distributed network.

- Node.js: A JavaScript runtime environment. It is essential for running client-side SDKs (like the Hyperledger Fabric SDK or Ethereum's web3.js) that allow external applications to communicate with the blockchain ledger.

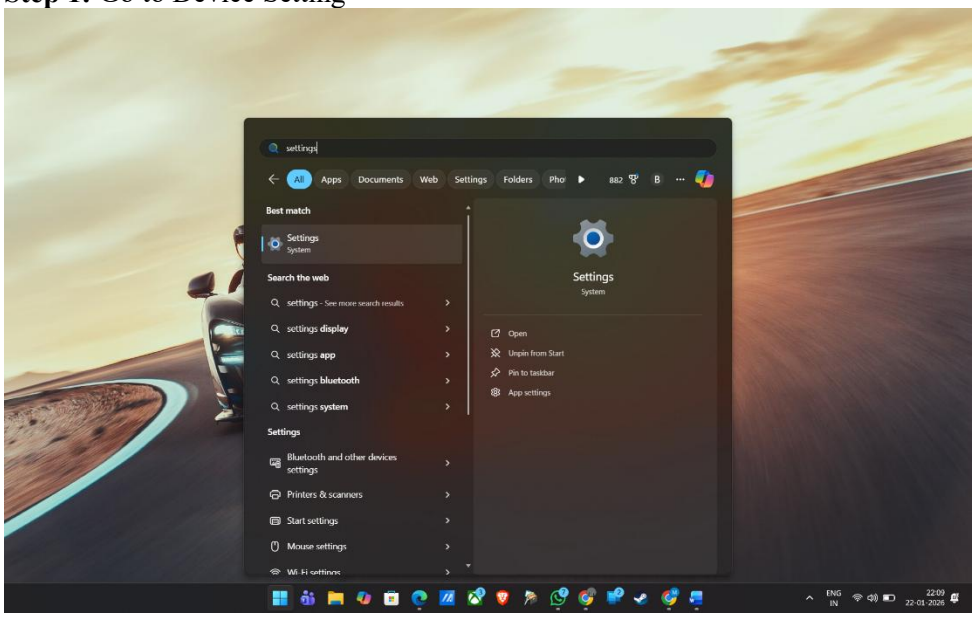
- Java: A robust, object-oriented language. It is a primary language used for writing Chaincode (smart contracts) in Hyperledger Fabric due to its stability and strong typing.

- Hyperledger Fabric: A permissioned, enterprise-grade blockchain framework. It uses a modular architecture where components like consensus and membership services run as separate Docker containers.

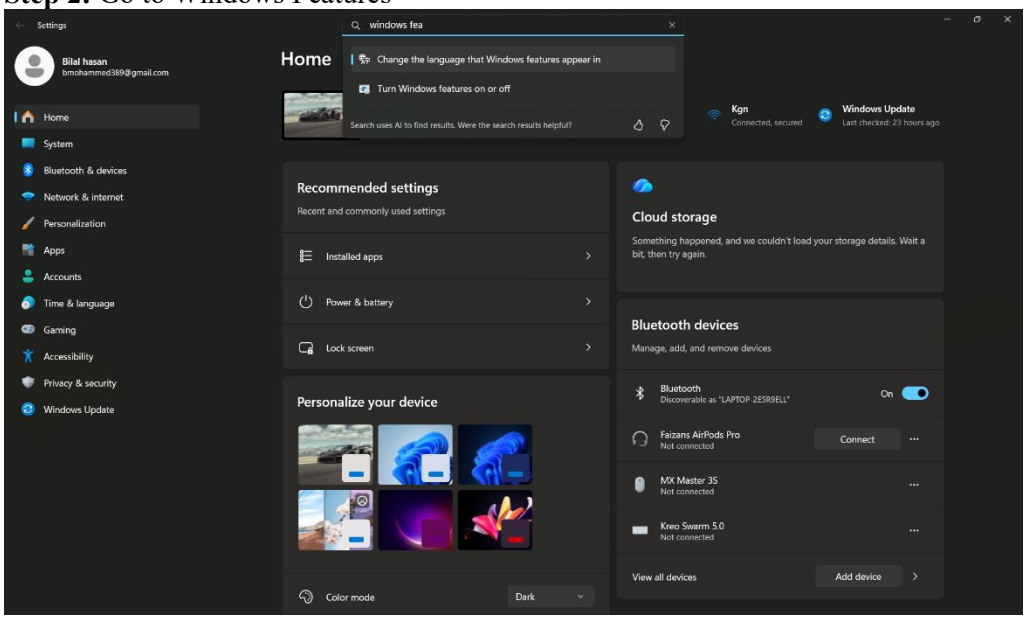
- Ethereum: A public blockchain platform featuring the Ethereum Virtual Machine (EVM). Development involves simulating the EVM locally to test smart contracts before deploying them to the live network.

Steps:

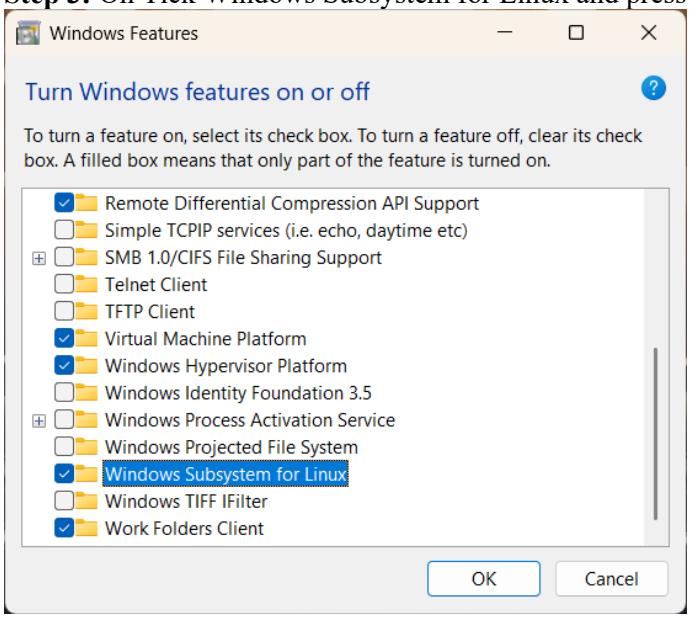
Step 1: Go to Device Setting



Step 2: Go to Windows Features



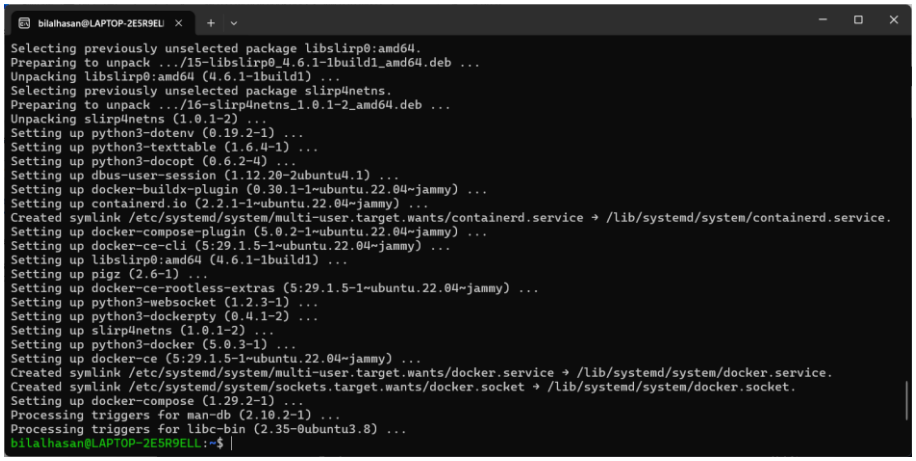
Step 3: On Tick Windows Subsystem for Linux and press OK.



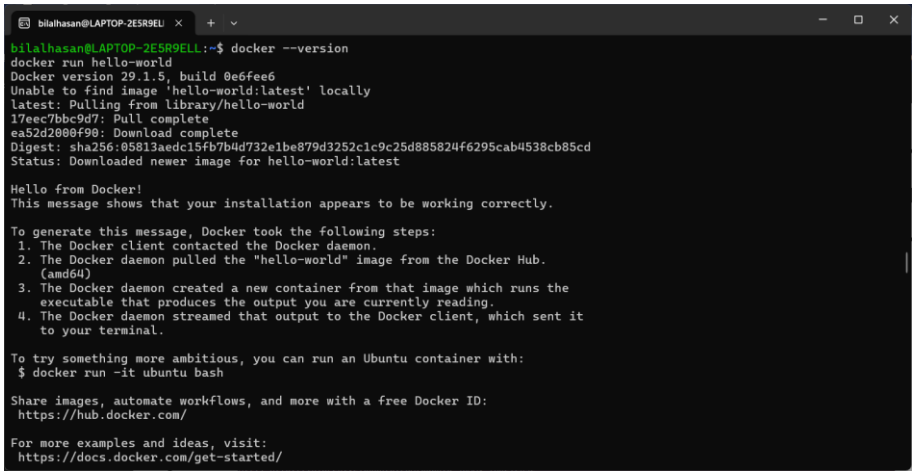
Step 4: Install following things. (using below command)

```
bilalhasan@LAPTOP-2E5R9ELL:~$ # Update system
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

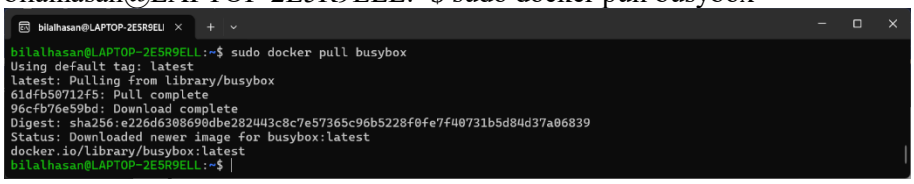
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin docker-compose
sudo usermod -aG docker $USER
newgrp docker



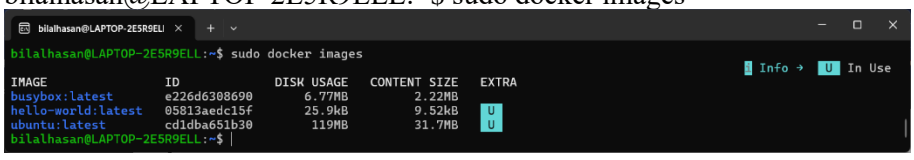
Step 5: Verifying the Installed Version and Program.
bilalhasan@LAPTOP-2E5R9ELL:~$ docker --version
docker run hello-world



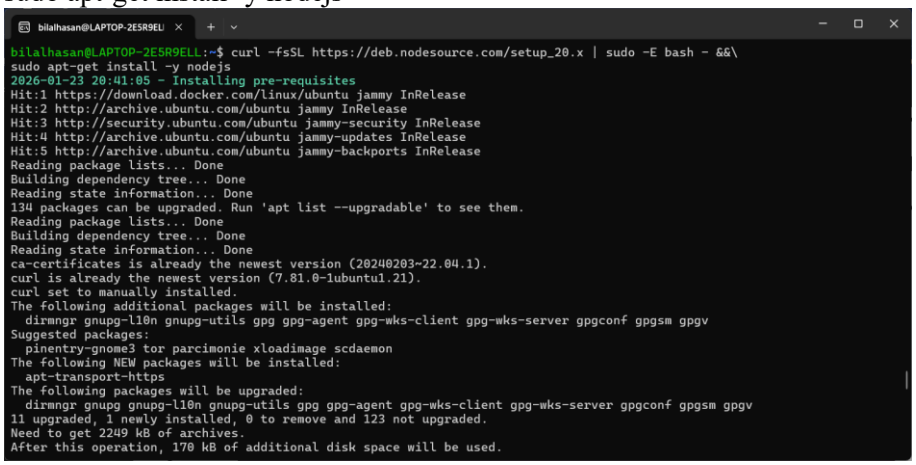
Step 6: Verify if docker is working
bilalhasan@LAPTOP-2E5R9ELL:~$ sudo docker pull busybox



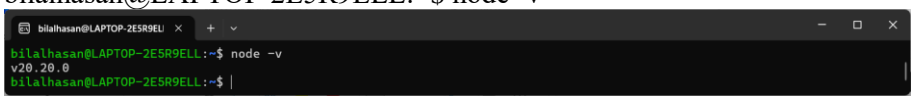
bilalhasan@LAPTOP-2E5R9ELL:~$ sudo docker images



Step 7(a): Installation of Node.js
bilalhasan@LAPTOP-2E5R9ELL:~$ curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash - &&\
sudo apt-get install -y nodejs



Verifying the version of node.js
bilalhasan@LAPTOP-2E5R9ELL:~$ node -v



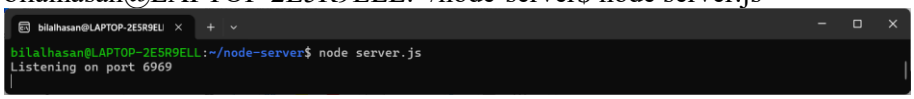
Step 7(b): Create Your Project Folder
bilalhasan@LAPTOP-2E5R9ELL:~$ mkdir ~/node-server
bilalhasan@LAPTOP-2E5R9ELL:~$ cd ~/node-server

Step 7(c): Create server.js
bilalhasan@LAPTOP-2E5R9ELL:~/node-server$ nano server.js
Paste this code:

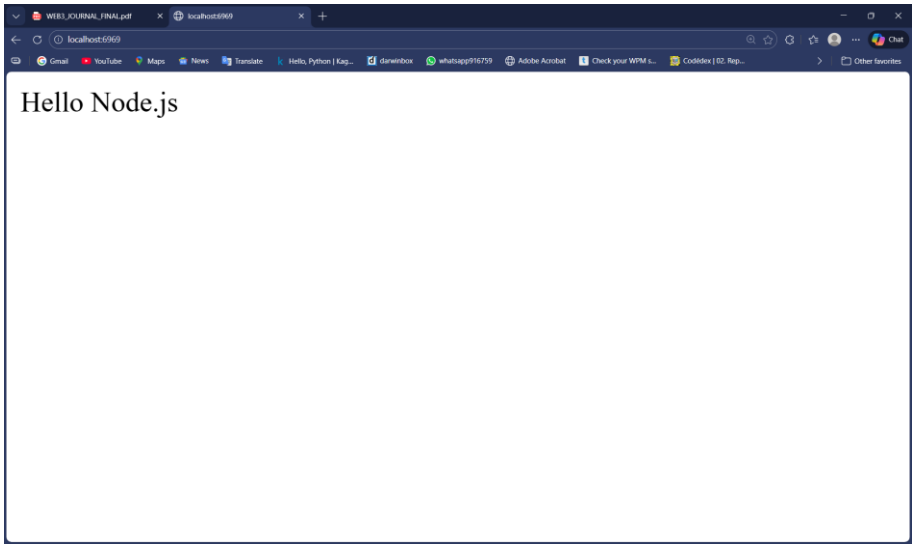
```
const http = require("http")

http.createServer((req, res) => {
   res.writeHead(200, {"content-type": "text/html"});
   res.end("Hello Node.js")
}).listen(6969, () => console.log("Listening on port 6969"))
```

Save and exit (Ctrl+O, Enter, Ctrl+X in nano).

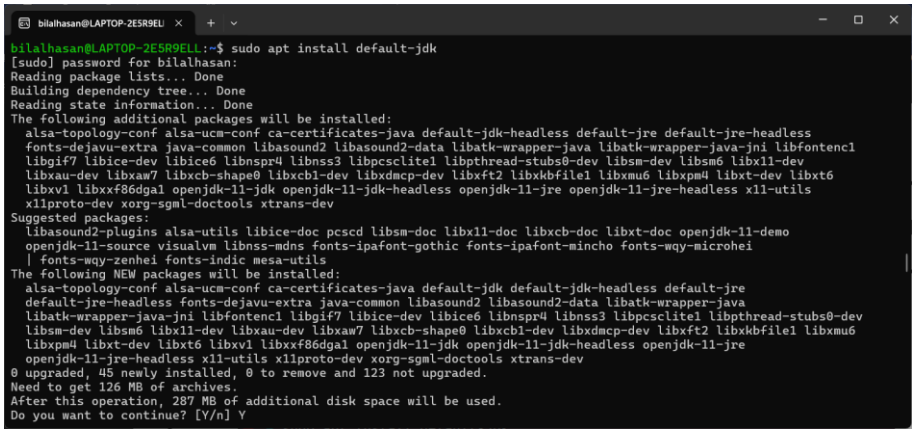
Step 7(d): Run the Server
bilalhasan@LAPTOP-2E5R9ELL:~/node-server$ node server.js



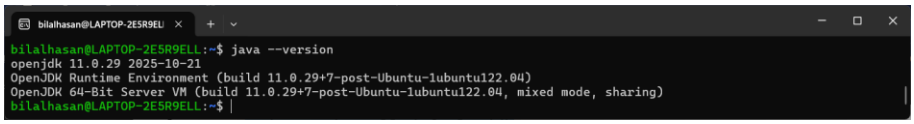
Step 7(e): View in Browser
Open your Windows browser and go to:
http://localhost:6969



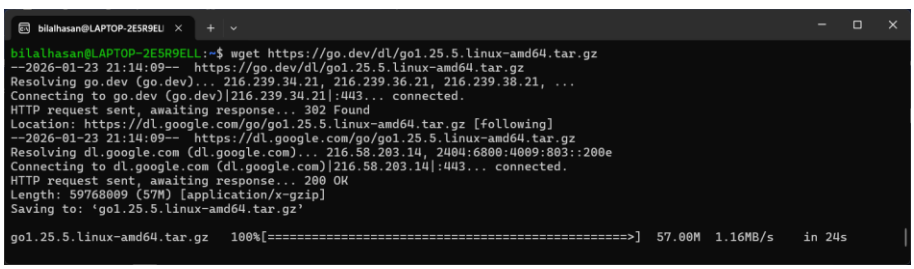
Step 8(a): Installing Java
bilalhasan@LAPTOP-2E5R9ELL:~$ sudo apt install default-jdk



Step 8(b): Verifying Java Installation.



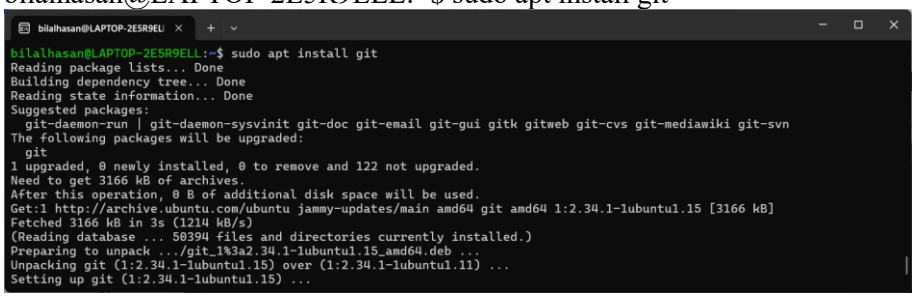
Step 9(a): Installing GoLang
bilalhasan@LAPTOP-2E5R9ELL:~$ wget https://go.dev/dl/go1.25.5.linux-amd64.tar.gz



bilalhasan@LAPTOP-2E5R9ELL:~$ sudo rm -rf /usr/local/go
bilalhasan@LAPTOP-2E5R9ELL:~$ sudo tar -C /usr/local -xzf go1.25.5.linux-amd64.tar.gz
bilalhasan@LAPTOP-2E5R9ELL:~$ echo 'export PATH=$PATH:/usr/local/go/bin' >> ~/.bashrc
source ~/.bashrc

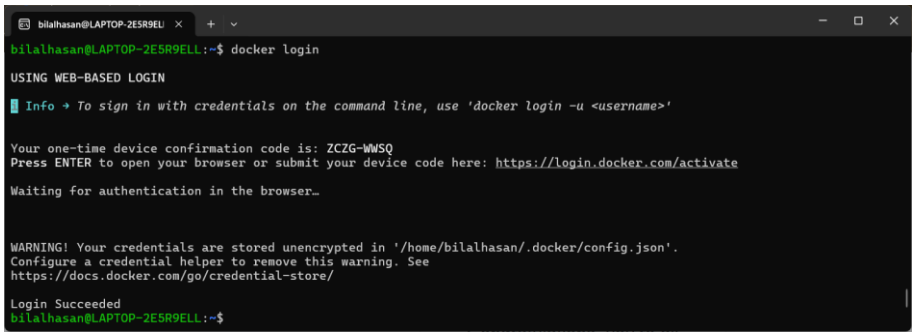
Step 9(b): Verifying go version
bilalhasan@LAPTOP-2E5R9ELL:~$ go version

Step 10(a): Installing Git
bilalhasan@LAPTOP-2E5R9ELL:~$ sudo apt install git

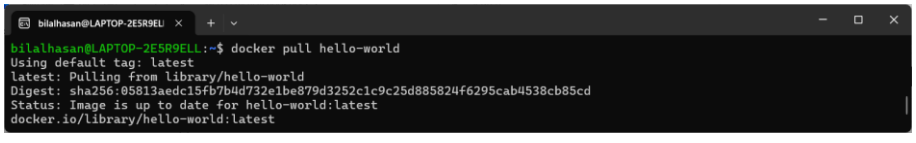


Step 10(b): Verifying Git version
bilalhasan@LAPTOP-2E5R9ELL:~$ git --version

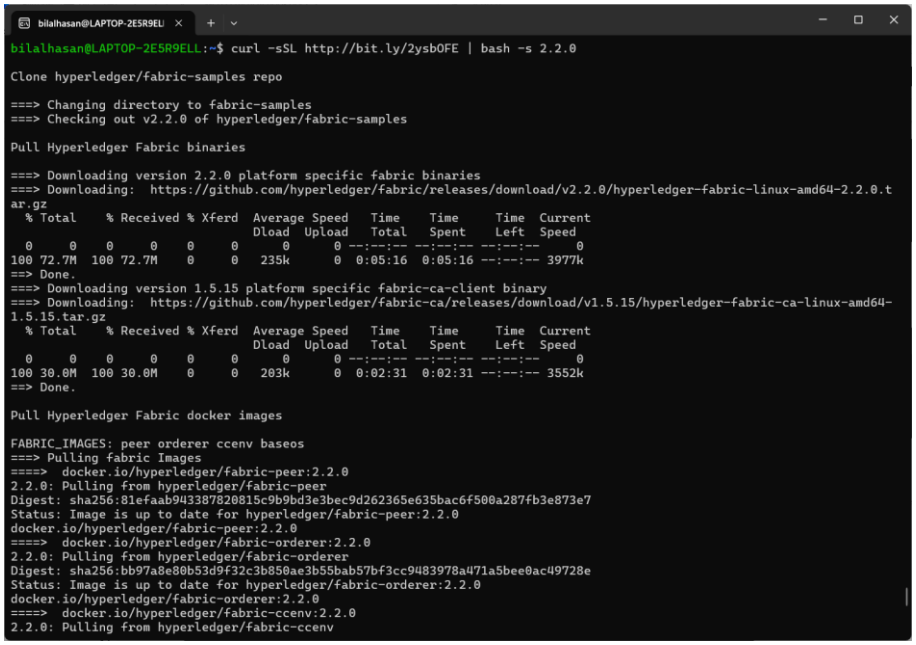
Step 10(c): Installing Hyperledger Fabric
Before installing hyper ledger make sure docker is up and running (sudo systemctl status docker)
Login into the Docker
bilalhasan@LAPTOP-2E5R9ELL:~$ docker login

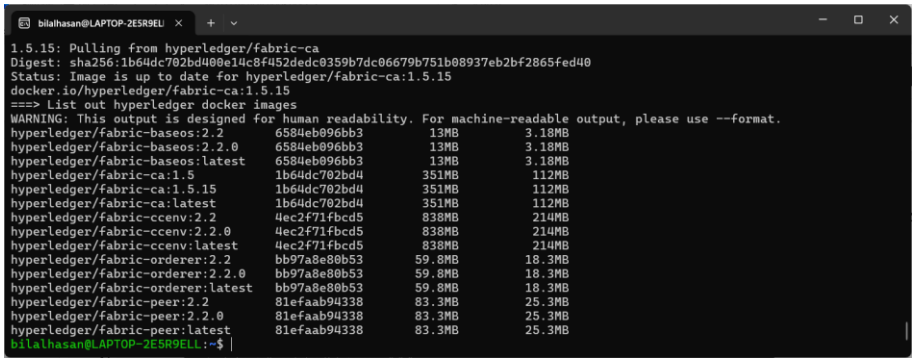


Verify Docker Pull Request
bilalhasan@LAPTOP-2E5R9ELL:~$ docker pull hello-world

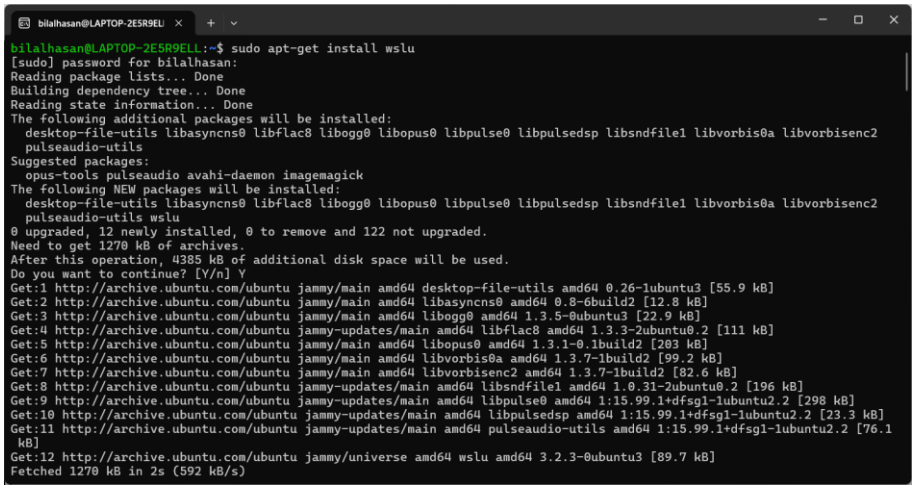


bilalhasan@LAPTOP-2E5R9ELL:~$ curl -sSL http://bit.ly/2ysbOFE | bash -s 2.2.0

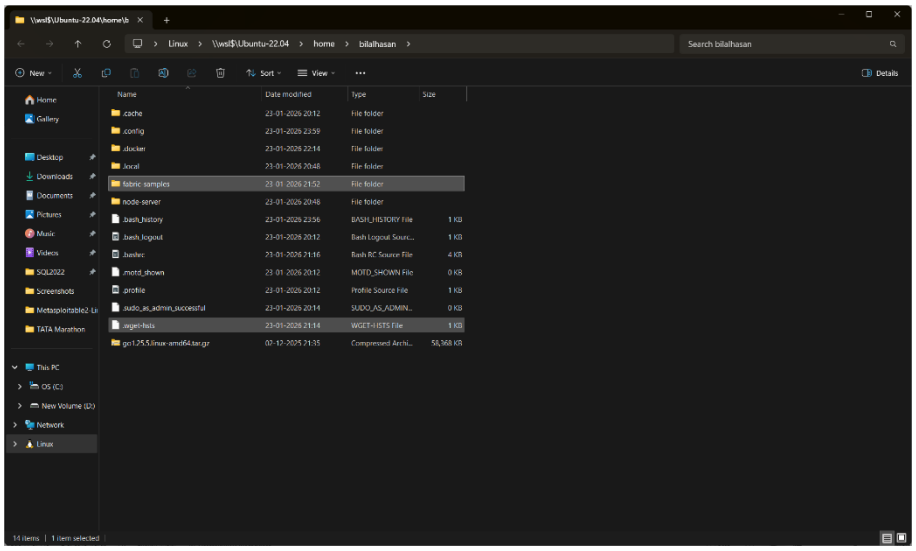




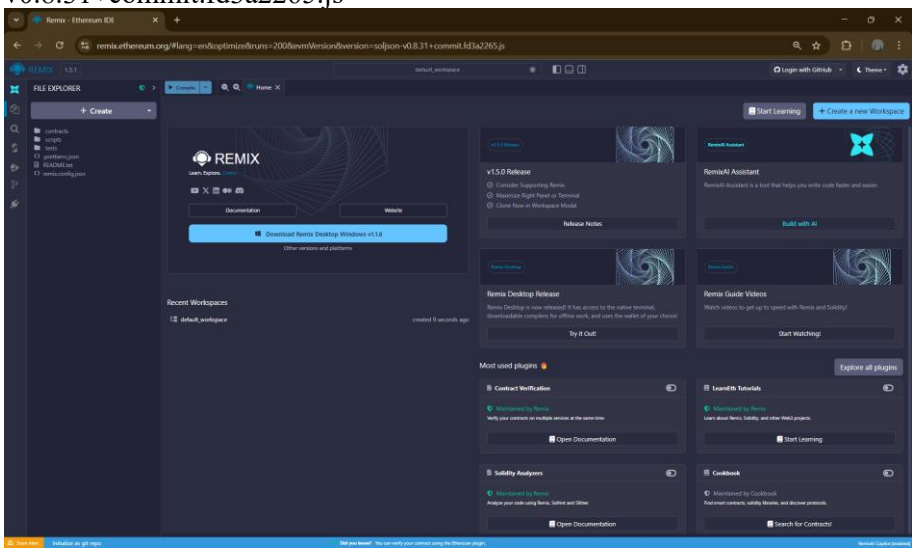
Step 10(d): Install wslu for viewing the liabrary where Hyperledger installed.
bilalhasan@LAPTOP-2E5R9ELL:~$ sudo apt-get install wslu



For Viewing the Liabrary:
bilalhasan@LAPTOP-2E5R9ELL:~$ wslview .



Step 12: Ethereum:
Running Ethereum IDE online (Remix IDE)
link: https://remix.ethereum.org/#lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js



Practical No. 2

Aim - Create and deploy a block chain network using Hyperledger Fabric SDK for Java.

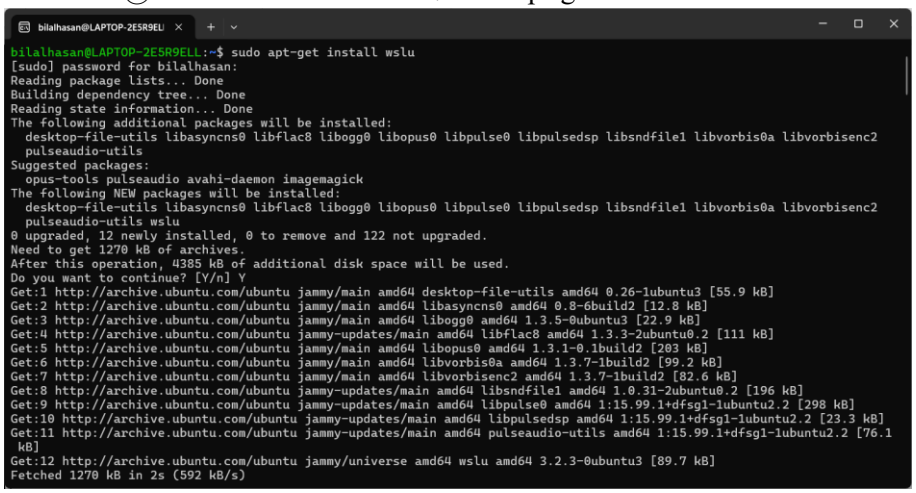
Theory: This experiment focuses on the application layer, demonstrating how a Java client application interacts with a running Hyperledger Fabric blockchain.

- Hyperledger Fabric SDK for Java: This is a library that enables Java applications to communicate with the Fabric network. It abstracts the complex low-level gRPC protocols, allowing developers to easily manage the network and invoke smart contracts.
- Connection Profile: The SDK connects to the blockchain using a Connection Profile (usually connection.yaml or json). This file describes the network topology, specifying the addresses and ports of the Peers, Orderers, and Certificate Authorities.
- Identity Management (Wallet): Fabric is a permissioned network. The SDK uses a Wallet to store digital identities (X.509 certificates and private keys). The Java application retrieves these credentials from the wallet to sign transactions, proving the user's identity.
- The Gateway Class: This is the primary entry point for the application. It acts as a bridge, managing the transaction lifecycle—from proposal and endorsement to submission—simplifying the code required to talk to the network.
- Transaction Types:
  - Submit Transaction: Used to write data to the ledger. This process involves the consensus mechanism and updating the world state.
  - Evaluate Transaction: Used to read data from the ledger. This is a quick query that executes on a peer but does not change the state or require consensus.

Steps:

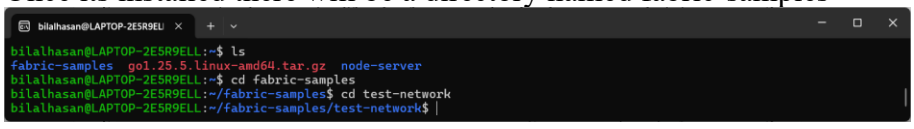
Step 1: Install wslu for viewing the liabrary where Hyperledger installed.

bilalhasan@LAPTOP-2E5R9ELL:~$ sudo apt-get install wslu

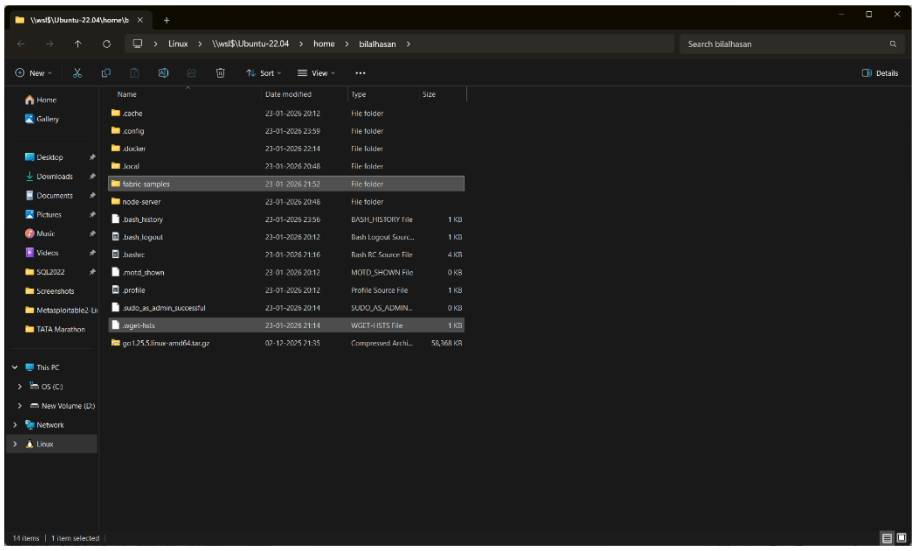


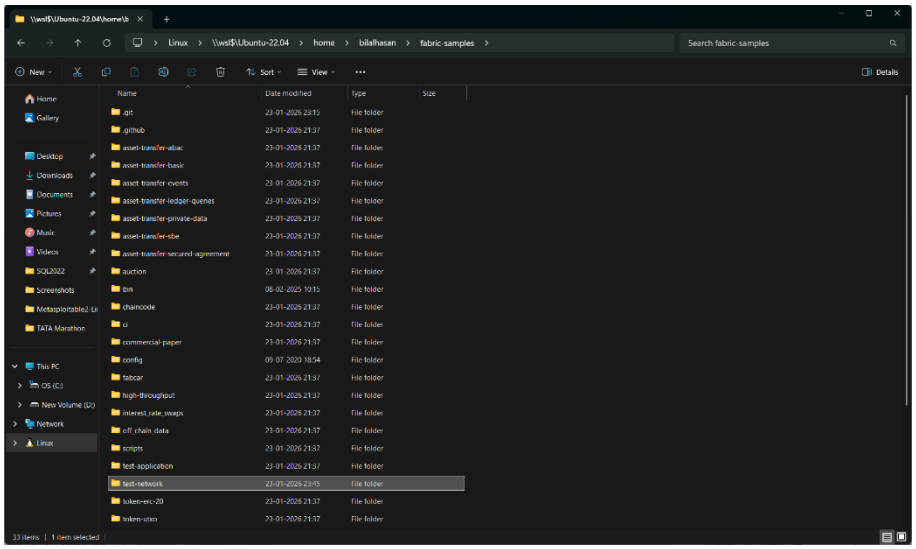
Step 2: Install Hyperledger (If not already done)

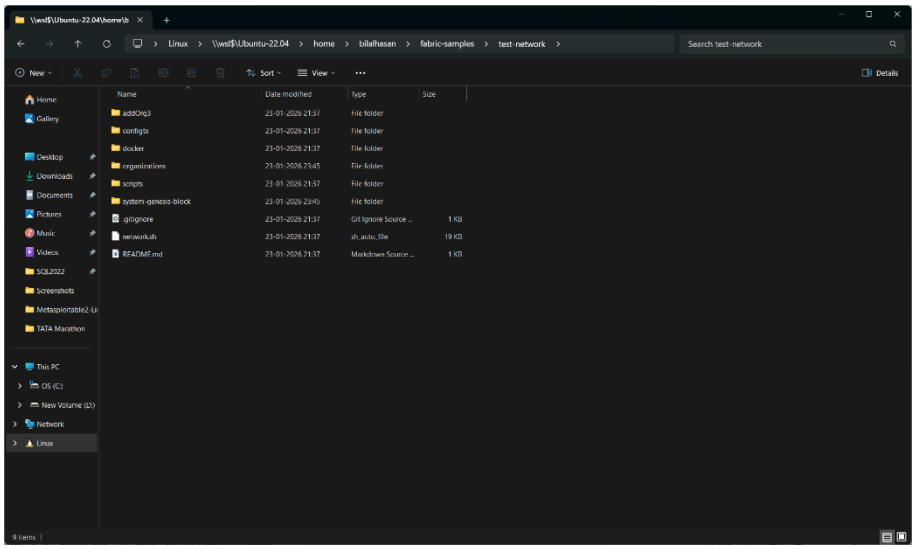
Once its installed there will be a directory named fabric-samples



For Viewing the Liabrary:
bilalhasan@LAPTOP-2E5R9ELL:~$ wslview .

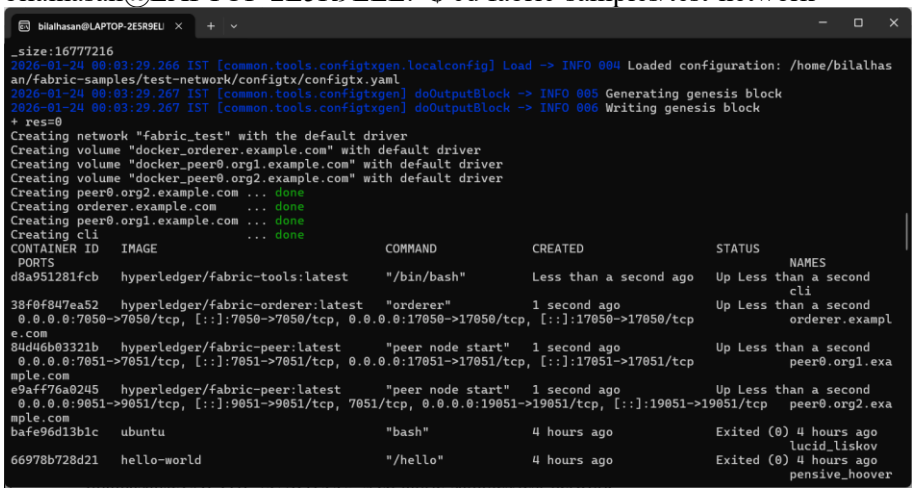






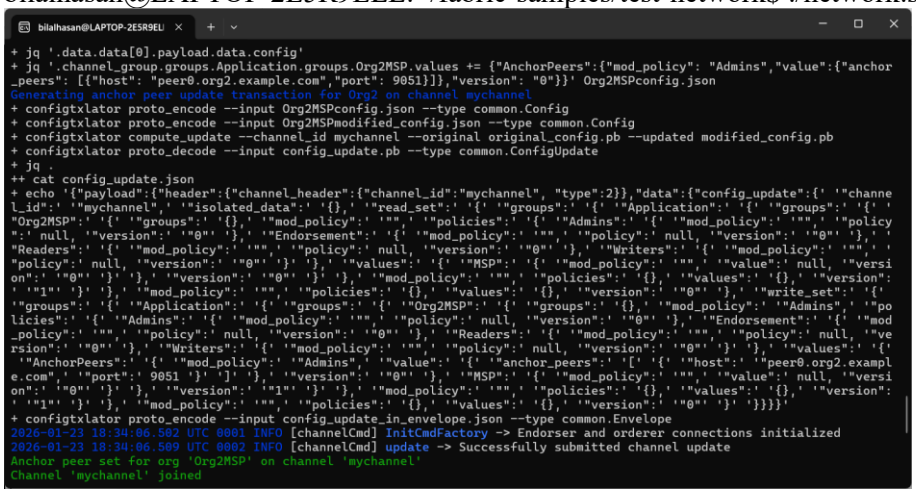
Step 3: Test the network.sh

bilalhasan@LAPTOP-2E5R9ELL:~$ cd fabric-samples/test-network



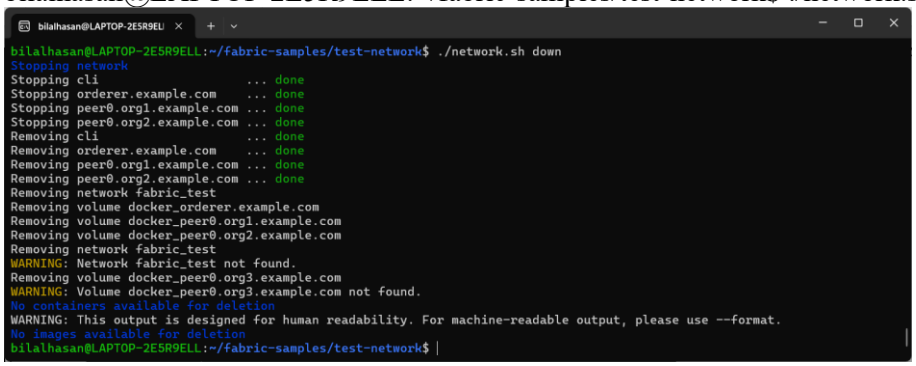
Step 4: Create Channel on network.sh

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ ./network.sh createChannel



Step 5: Down network.sh

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ ./network.sh down



Practical No. 3

Aim - Interact with a block chain network. Execute transactions and requests against a block chain network by creating an app to test the network and its rules.

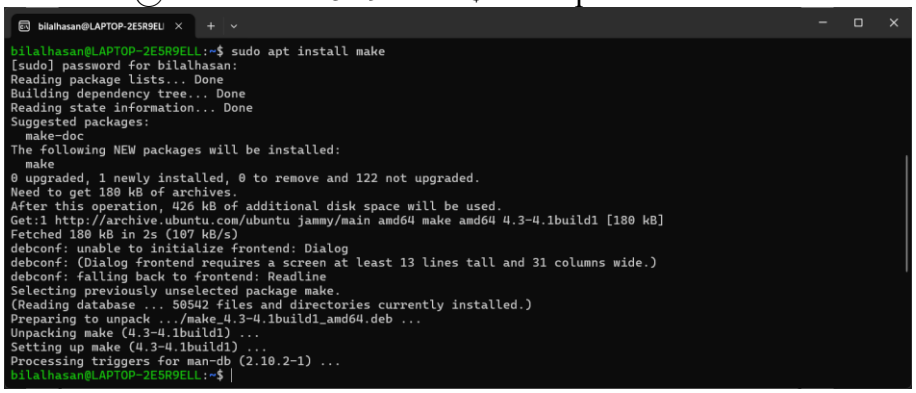
Theory:

To interact with a permissioned blockchain like Hyperledger Fabric, a Client Application is used. This application acts as a bridge between the end-user and the blockchain network, utilizing the Fabric SDK (Software Development Kit) to invoke smart contracts.

1. The Application Gateway: The application connects to the network using a "Gateway." The Gateway manages the complexity of the network interactions, such as service discovery (finding available peers) and transaction submission.
2. Identity Management (Wallet): All interactions are authenticated. The application uses a Wallet to store digital identities (X.509 certificates). Every transaction request is cryptographically signed by the user's private key to prove identity and authority.
3. Transaction Types:
   a. Submit Transaction (Invoke): Used to modify the ledger (e.g., creating or transferring an asset). This process is complex: the transaction is proposed to peers for endorsement, sent to the Orderer for packaging into a block, and finally validated and committed by all peers.
   b. Evaluate Transaction (Query): Used to read data from the ledger. This is lightweight; the client queries a peer directly, and the peer returns the result immediately without involving the ordering service or consensus.
4. Testing Network Rules: The application tests the "business rules" defined in the Chaincode. For example, if the logic states "Asset ownership cannot be negative," the app sends a request violating this to verify that the network correctly rejects it.

Steps:

Step 1: Installing dependencies
bilalhasan@LAPTOP-2E5R9ELL:~$ sudo apt install make



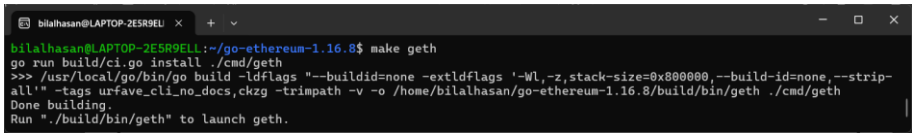
(Use this command if Golang is not installed in the PC)
sudo apt update
sudo apt install golang-go
Step 2: Install GEth (Go Ethereum)

Geth is a CLI with some resources to connect you to the Ethereum network. It will be used to start our private network in the local environment.

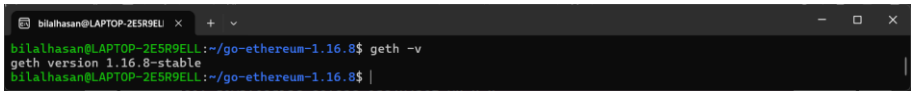
To install Geth in Ubuntu/Debian follow the following steps:

bilalhasan@LAPTOP-2E5R9ELL:~$ git clone -b release/1.11 --depth 1 https://github.com/ethereum/go-ethereum.git

bilalhasan@LAPTOP-2E5R9ELL:~/go-ethereum-1.16.8$ make geth



bilalhasan@LAPTOP-2E5R9ELL:~/go-ethereum-1.16.8$ geth -v



The Genesis Block

All blockchains start with the genesis block.This block defines the initial configuration to a blockchain. The configuration to genesis block is defined in genesis.json file.

So, let's create a folder inside the web3 folder to start the private network and create the genesis.json file.

Start Database (The Bootnode)

The bootnode is the first node created when the blockchain is started. To start the database to first node, execute:

The folder node1 will be created with the database to bootnode.

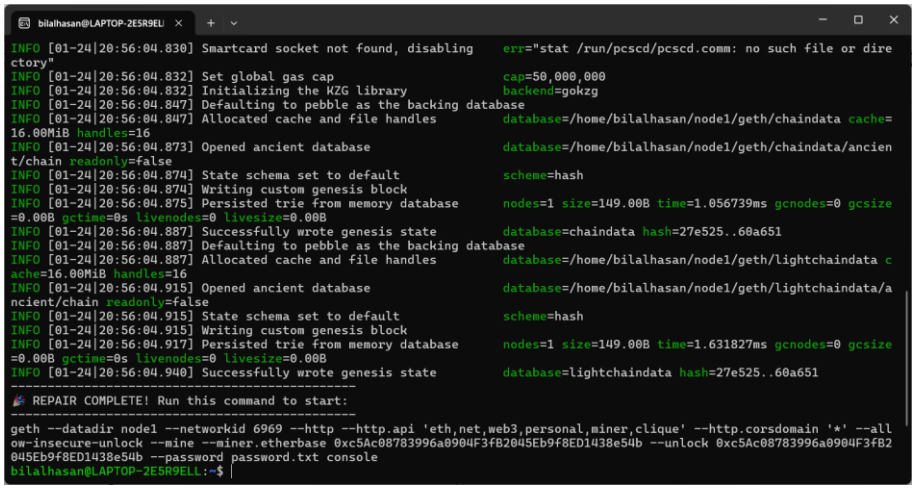
The terminal result must be something like this:

bilalhasan@LAPTOP-2E5R9ELL:~$ cd blockchain

Config genesis.json

```
# 1. Clean up old broken data
pkill -f geth
rm -rf node1 genesis.json start.sh
# 2. Create password
echo "bilal2000" > password.txt
# 3. Create NEW account
echo "Generating new account..."
ADDRESS=$(geth account new --datadir node1 --password password.txt | grep "Public address" |
awk '{print $6}')
```

```
# 4. Generate Genesis (Safe Mode: No Cancun)
python3 -c "
import json
address = '$ADDRESS'[2:]
extra_data = '0x' + ('0' * 64) + address.lower() + ('0' * 130)
genesis = {
  'config': { 'chainId': 6969, 'homesteadBlock': 0, 'eip150Block': 0, 'eip155Block': 0, 'eip158Block': 0,
'byzantiumBlock': 0, 'constantinopleBlock': 0, 'petersburgBlock': 0, 'istanbulBlock': 0, 'berlinBlock': 0,
'londonBlock': 0, 'shanghaiTime': 0, 'clique': { 'period': 5, 'epoch': 30000 } },
  'difficulty': '1', 'gasLimit': '30000000', 'alloc': { '$ADDRESS': { 'balance':
'100000000000000000000' } }, 'extraData': extra_data
}
with open('genesis.json', 'w') as f: json.dump(genesis, f, indent=2)
"
# 5. Initialize
geth init --datadir node1 genesis.json
# 6. Create the Shortcut Script (start.sh)
echo "geth --datadir node1 --networkid 6969 --http --http.api 'eth,net,web3,personal,miner,clique' --
http.corsdomain '*' --allow-insecure-unlock --mine --miner.etherbase $ADDRESS --unlock
$ADDRESS --password password.txt console" > start.sh
chmod +x start.sh
echo "------------------------------------------------------"
echo " ✅ SETUP COMPLETE! Your address is: $ADDRESS"
echo "------------------------------------------------------"
```

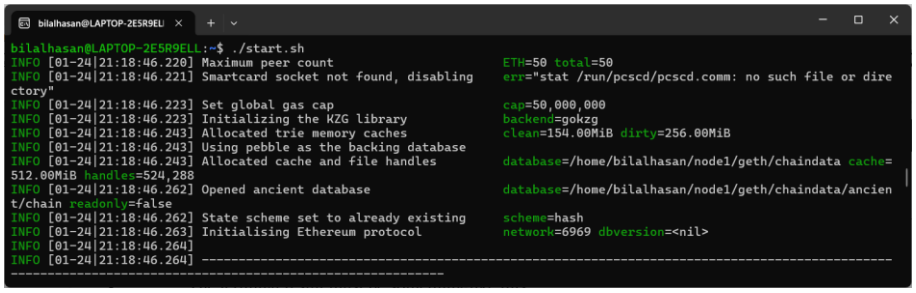


Start Node

- This command will start the node in the private network with id 6969.
- The flag --http is used to enable Web App Access, we will use this in next post to connect Metamask with that private network.
- The flag --allow-insecure-unlock is used to permit execute transfers without a web application, we will use this in tests to this network.
- The flag --nodiscover is used to prevent node from trying to connect to others automatically.

The terminal result must be something like this:

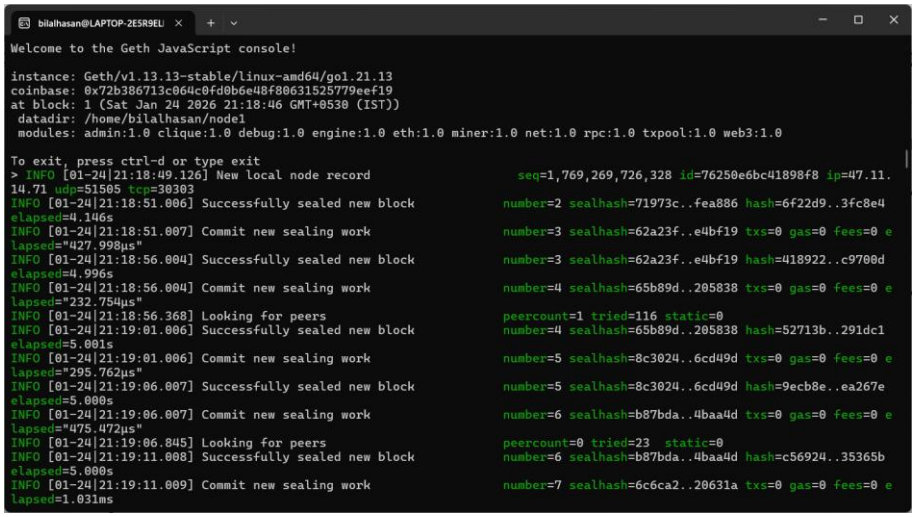
bilalhasan@LAPTOP-2E5R9ELL:~$ ./start.sh



Done!! The blockchain in the private network is running.

Let's go do some tests to verify this. Open another terminal window in the same folder my blockchain and execute the following command:

This opens an interactive javascript terminal to execute some tasks to this node. The terminal result should be something like this:



Adding member peers

Let's start the second node to our blockchain. To do this execute the following commands in new terminal inside the same directory:

Make directory Node2 for adding member in peer

mkdir node2

create geth for node2

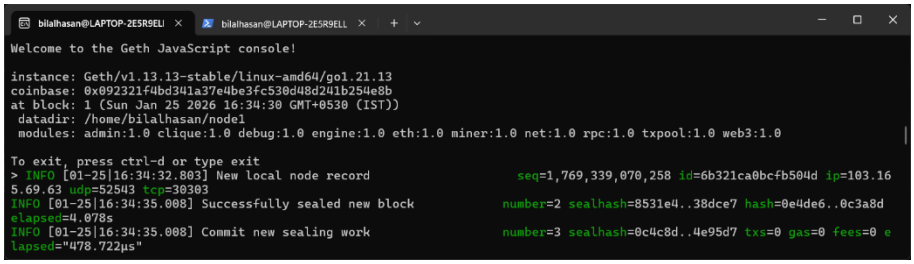
geth init --datadir node2 genesis.json

make start2.sh command for starting the second peer.

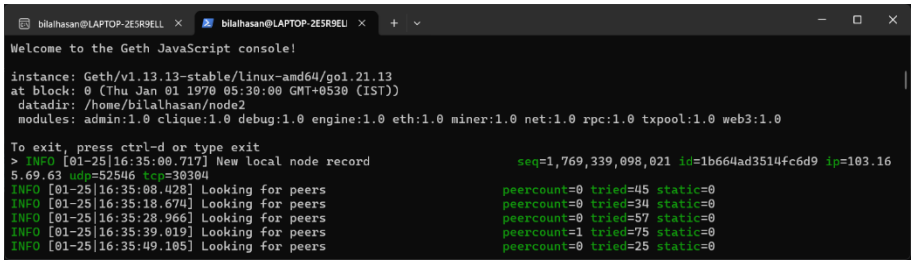
echo "geth --datadir node2 --networkid 6969 --port 30304 --http --http.addr 127.0.0.1 --http.port 8546 --http.api 'eth,net,web3,personal,miner,clique' --http.corsdomain '*' --authrpc.port 8552 console" > start2.sh

bilalhasan@LAPTOP-2E5R9ELL:~$ ./start2.sh

Node1



Node2



Connecting Peers:

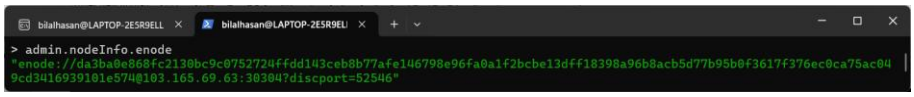
Now, let's create the peer-to-peer connection between the two nodes in blockchain.

Copy the enode from the nodeInfo of the second node, then execute the following command in javascript terminal of the first node

For Connecting both node:

In Node2 terminal get the enode using below command:

> admin.nodeInfo.enode

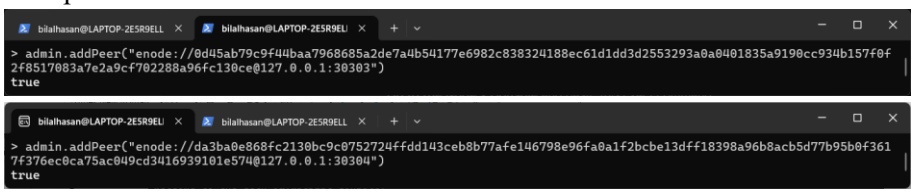


Copy the enode from the nodeInfo of the first node, then execute the following command in javascript terminal of the second node:

In Node1 terminal put the below command to link both node:

>admin.addPeer("enode://0d45ab79c9f44baa7968685a2de7a4b54177e6982c838324188ec61d1dd3d2553293a0a0401835a9190cc934b157f0f2f8517083a7e2a9cf702288a96fc130ce@127.0.0.1:30303")

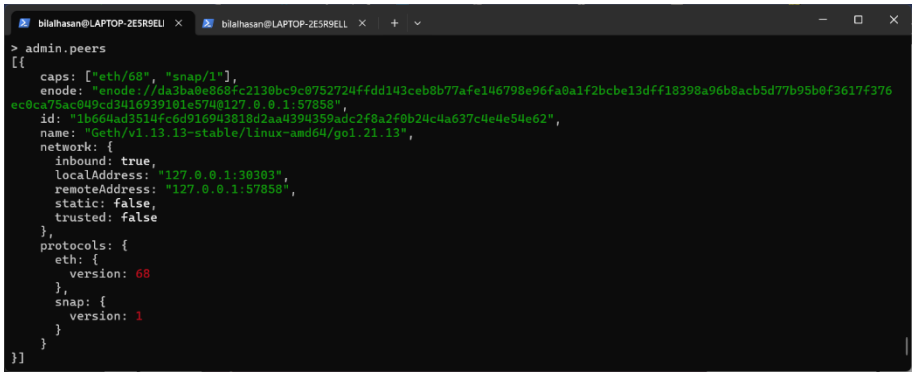
>net.peerCount



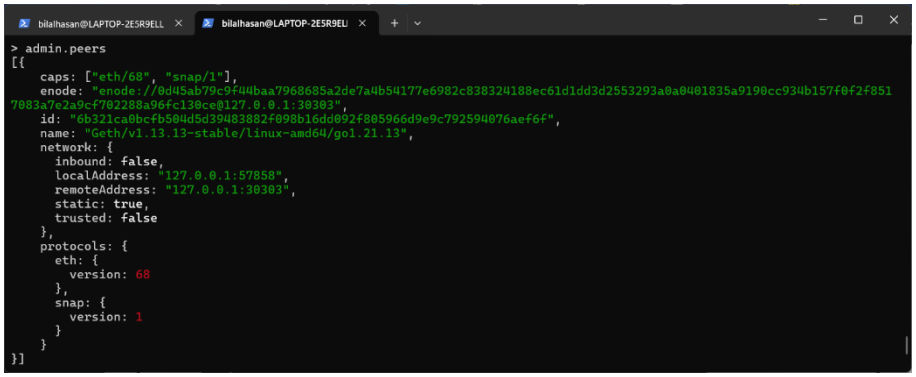
For Verifying the both connected on the same block chain the Output get from the below command should be 1.

net.peerCount

Node1:



Node2:

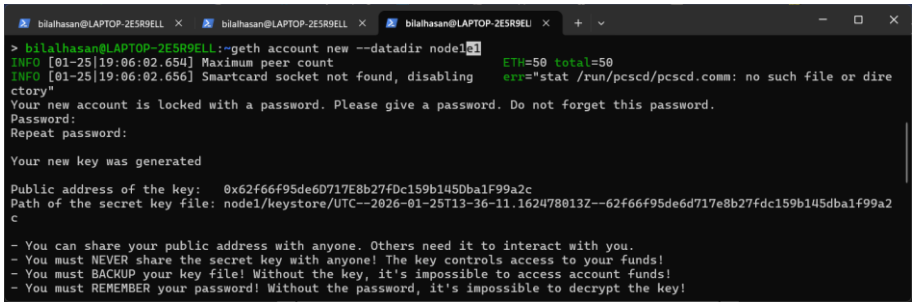


Done!! Your blockchain is created with multiple nodes running and connected.
Let's create accounts and start mining to see data updated in both nodes.

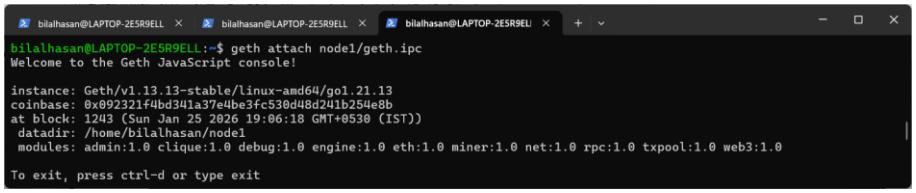
Mining

To mine blocks it's necessary to have a base account. Let's do this. In the javascript terminal of the first node, execute the command to create a new account and define the password required to this account, the public address from the created account should be presented.

Open New Terminal for WSL
> bilalhasan@LAPTOP-2E5R9ELL:~geth account new --datadir node1e1



bilalhasan@LAPTOP-2E5R9ELL:~$ geth attach node1/geth.ipc



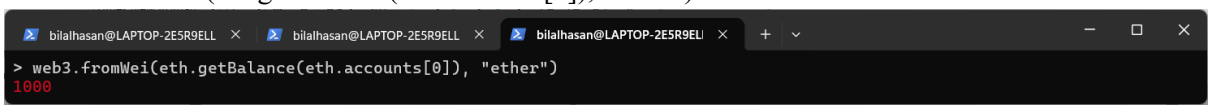
In the “>” function type this command:
> miner.setEtherbase(eth.accounts[0])

> miner.start()

>miner.stop()

Eth.blockNumber

> web3.fromWei(eth.getBalance(eth.accounts[0]), "ether")



Practical No. 4

Aim - Deploy an asset-transfer app using block chain. Learn app development within a Hyperledger Fabric network.

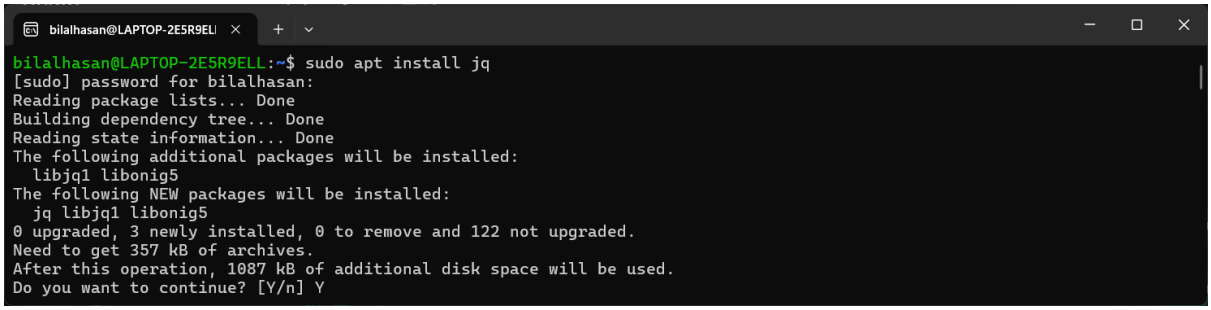
Objective: To understand the fundamental operations of a ledger (Create, Read, Update, Delete) within a Hyperledger Fabric network.

Theory:

- Hyperledger Fabric Architecture: Unlike public blockchains (like Bitcoin), Fabric is a permissioned network. All participants are known and authenticated via an MSP (Membership Service Provider).
- Asset Transfer Concept: In blockchain, an "asset" can be tangible (a house, car) or intangible (intellectual property, stocks). The "Transfer" logic is handled by Chaincode (Smart Contracts).
- The Ledger: The Fabric ledger consists of two parts:
  1. World State: A database (typically CouchDB or LevelDB) that holds the current value of an asset.
  2. Blockchain (Transaction Log): An immutable history of all transactions that resulted in the current world state.
- Smart Contract Role: The application logic defines the asset (e.g., AssetID, Owner, Color, Size) and the rules for changing ownership.

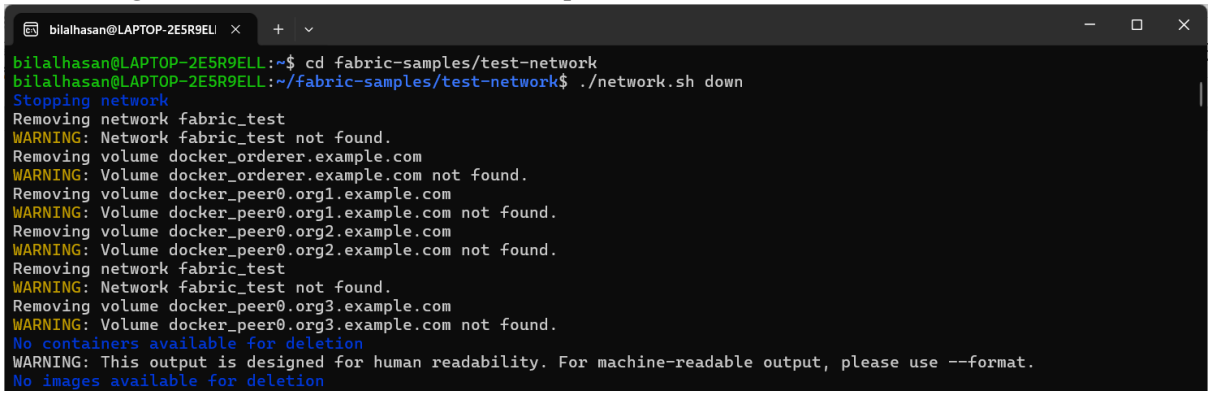
Steps:

Installing dependencies
bilalhasan@LAPTOP-2E5R9ELL:~$ sudo apt install jq

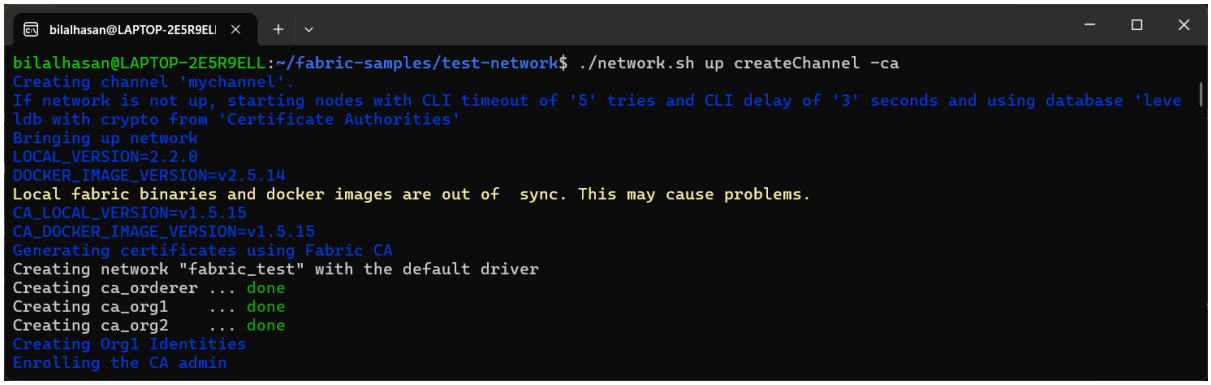


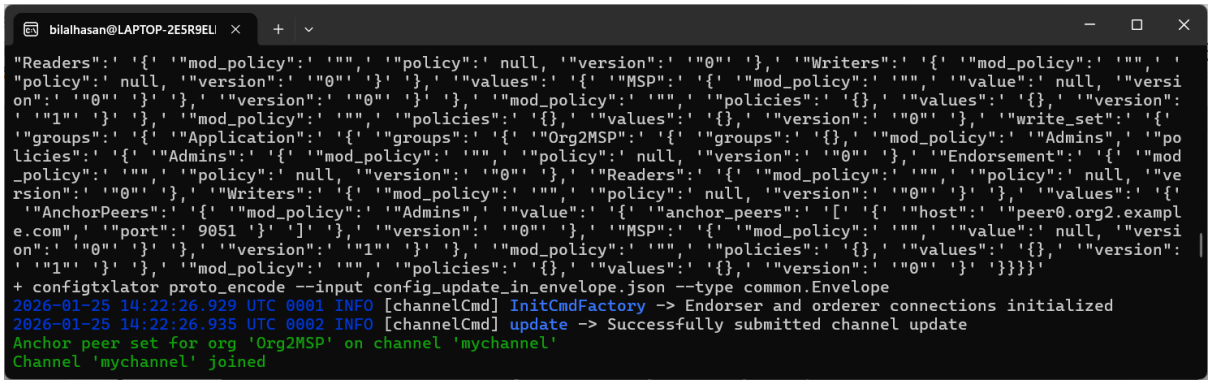
Start the network and deploy the smart contract

bilalhasan@LAPTOP-2E5R9ELL:~$ cd fabric-samples/test-network
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ ./network.sh down



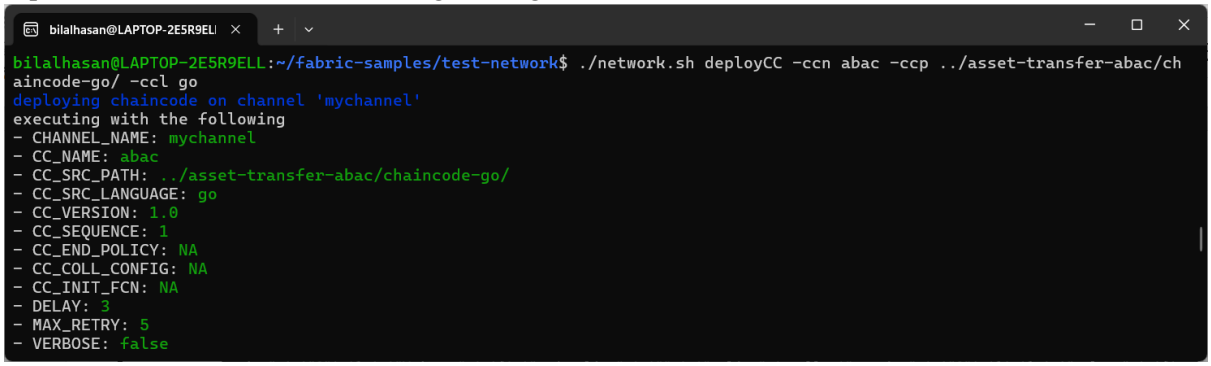
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ ./network.sh up createChannel -ca





You can then use the test network script to deploy the asset-transfer-abac smart contract to a channel on the network:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ ./network.sh deployCC -ccn abac -ccp ../asset-transfer-abac/chaincode-go/ -ccl go



Register identities with attributes

We can use the one of the test network Certificate Authorities to register and enroll identities with the attribute of abac.creator=true. First, we need to set the following environment variables in order to use the Fabric CA client.

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$

export PATH=${PWD}/../bin:${PWD}:$PATH

export FABRIC_CFG_PATH=$PWD/../config/

We will create the identities using the Org1 CA. Set the Fabric CA client home to the MSP of the Org1 CA admin:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ export FABRIC_CA_CLIENT_HOME=${PWD}/organizations/peerOrganizations/org1.example.com/

```
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ export PATH=${PWD}/../bin:${PWD}:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ export FABRIC_CA_CLIENT_HOME=${PWD}/organizations/peerOrganizations/org1.example.com/
```

There are two ways to generate certificates with attributes added. We will use both methods and create two identities in the process. The first method is to specify that the attribute be added to the certificate by default when the identity is registered. The following command will register an identity named creator1 with the attribute of abac.creator=true.

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ fabric-ca-client register --id.name creator1 --id.secret creator1pw --id.type client --id.affiliation org1 --id.attrs 'abac.creator=true:ecert' --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ fabric-ca-client enroll -u https://creator1:creator1pw@localhost:7054 --caname ca-org1 -M "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp" --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ cp "${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.yaml" "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/config.yaml"

```
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ fabric-ca-client register --id.name creator1 --id.secret creator1pw --id.type client --id.affiliation org1 --id.attrs 'abac.creator=true:ecert' --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
2026/01/25 19:54:04 [INFO] Configuration file location: /home/bilalhasan/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2026/01/25 19:54:04 [INFO] TLS Enabled
2026/01/25 19:54:04 [INFO] TLS Enabled
Password: creator1pw
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ fabric-ca-client enroll -u https://creator1:creator1pw@localhost:7054 --caname ca-org1 -M "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp" --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
2026/01/25 19:54:41 [INFO] TLS Enabled
2026/01/25 19:54:41 [INFO] generating key: &{A:ecdsa S:256}
2026/01/25 19:54:41 [INFO] encoded CSR
2026/01/25 19:54:41 [INFO] Stored client certificate at /home/bilalhasan/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/signcerts/cert.pem
2026/01/25 19:54:41 [INFO] Stored root CA certificate at /home/bilalhasan/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/cacerts/localhost-7054-ca-org1.pem
2026/01/25 19:54:41 [INFO] Stored Issuer public key at /home/bilalhasan/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/IssuerPublicKey
2026/01/25 19:54:41 [INFO] Stored Issuer revocation public key at /home/bilalhasan/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/IssuerRevocationPublicKey
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ cp "${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.yaml" "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/config.yaml"
```

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ fabric-ca-client register --id.name creator2 --id.secret creator2pw --id.type client --id.affiliation org1 --id.attrs 'abac.creator=true:' --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ fabric-ca-client enroll -u https://creator2:creator2pw@localhost:7054 --caname ca-org1 --enrollment.attrs "abac.creator" -M "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp" --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"

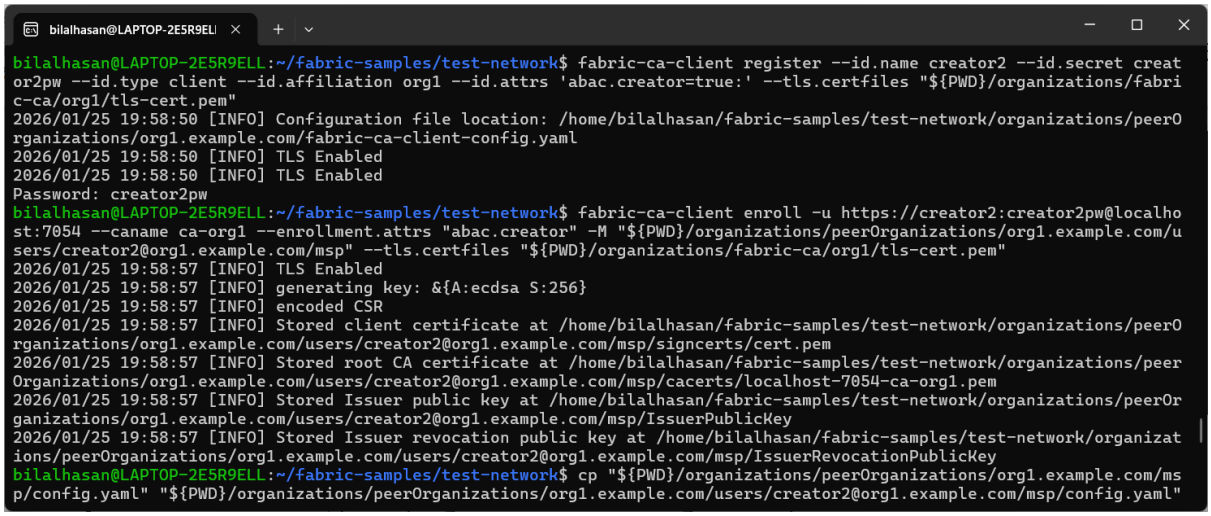
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ cp "${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.yaml" "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/config.yaml"

The ecert suffix adds the attribute to the certificate automatically when the identity is enrolled. As a result, the following enroll command will contain the attribute that was provided in the registration command.

The second method is to request that the attribute be added upon enrollment. The following command will register an identity named creator2 with the same abac.creator attribute.

The following enroll command will add the attribute to the certificate:

Run the command below to copy the Node OU configuration file into the creator2 MSP folder.



Create an asset:

You can use either identity with the abac.creator=true attribute to create an asset using the asset-transfer-abac smart contract. We will set the following environment variables to use the first identity that was generated, creator1:
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_ADDRESS=localhost:7051
export TARGET_TLS_OPTIONS=(-o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt")

```
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_ADDRESS=localhost:7051
export TARGET_TLS_OPTIONS=(-o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt")
```

Run the following command to create Asset1:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function":"CreateAsset","Args":["Asset1","blue","20","100"]}'

The result will list the creator1 identity as the asset owner.

```
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function":"CreateAsset","Args":["Asset1","blue","20","100"]}'
2026-01-25 19:59:28.367 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
```

Transfer the asset:

As the owner of Asset1, the creator1 identity has the ability to transfer the asset to another owner. In order to transfer the asset, the owner needs to provide the name and issuer of the new owner to the TransferAsset function. The asset-transfer-abac smart contract has a GetSubmittingClientIdentity function that allows users to retrieve their certificate information and provide it to the asset owner out of band (we omit this step). Issue the command below to transfer Asset1 to the user1 identity from Org1 that was created when the test network was deployed:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ export RECIPIENT="x509::CN=user1,OU=client,O=Hyperledger,ST=North Carolina,C=US::CN=ca.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US"

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function":"TransferAsset","Args":["Asset1","""$RECIPIENT"""]}'

```
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ export RECIPIENT="x509::CN=user1,OU=client,O=Hyperledger,ST=North Carolina,C=US::CN=ca.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US"
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function":"TransferAsset","Args":["Asset1","'"$RECIPIENT"'"]}'
2026-01-25 20:01:14.808 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
```

Query the ledger to verify that the asset has a new owner:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1"]}'

We can see that Asset1 with is now owned by User1:

```
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1"]}'
{"ID":"Asset1","color":"blue","size":20,"owner":"x509::CN=user1,OU=client,O=Hyperledger,ST=North Carolina,C=US::CN=ca.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US","appraisedValue":100}
```

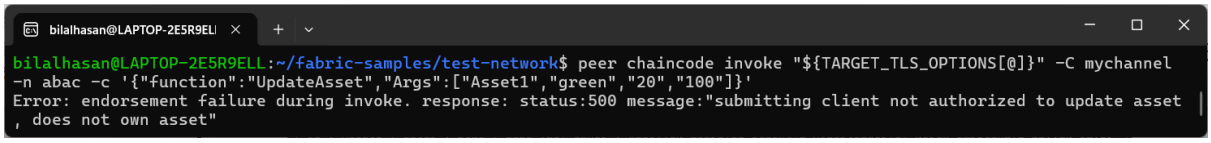
Update the asset:

Now that the asset has been transferred, the new owner can update the asset properties. The smart contract uses the GetID() API to ensure that the update is being submitted by the asset owner. To demonstrate the difference between identity and attribute based access control, lets try to update the asset using the creator1 identity first:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function":"UpdateAsset","Args":["Asset1","green","20","100"]}'

Even though creator1 can create new assets, the smart contract detects that the transaction was not submitted by the identity that owns the asset, user1. The command returns the following error:

Error: endorsement failure during invoke. response: status:500 message:"submitting client not authorized to update asset, does not own asset"

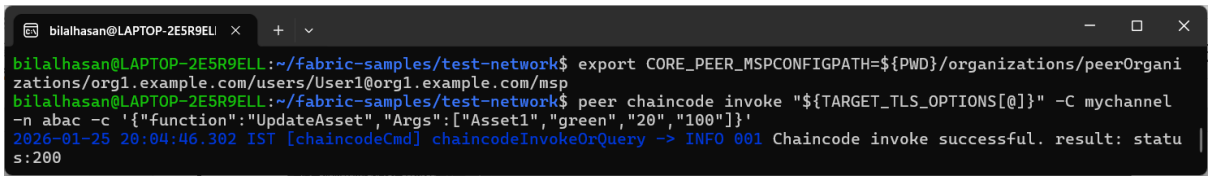


Run the following command to operate as the asset owner by setting the MSP path to User1:

We can now update the asset. Run the following command to change the asset color from blue to green. All other aspects of the asset will remain unchanged.

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp

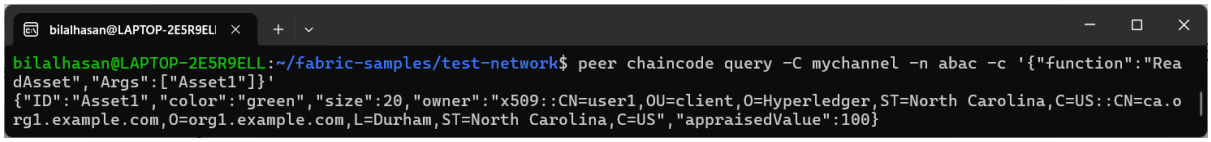
bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function":"UpdateAsset","Args":["Asset1","green","20","100"]}'



Run the query command again to verify that the asset has changed color:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1"]}'

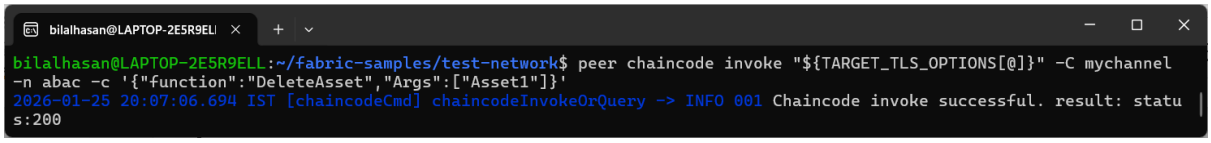
The result will display that Asset1 is now green:



Delete the asset:

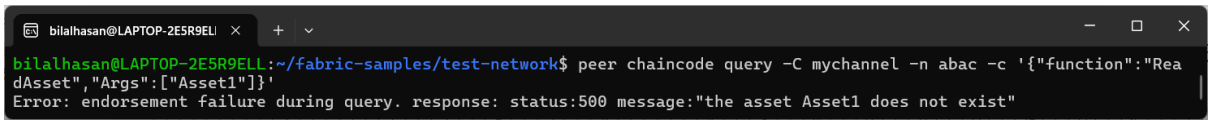
The owner also has the ability to delete the asset. Run the following command to remove Asset1 from the ledger:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function":"DeleteAsset","Args":["Asset1"]}'



If you query the ledger once more, you will see that Asset1 no longer exists:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1"]}'

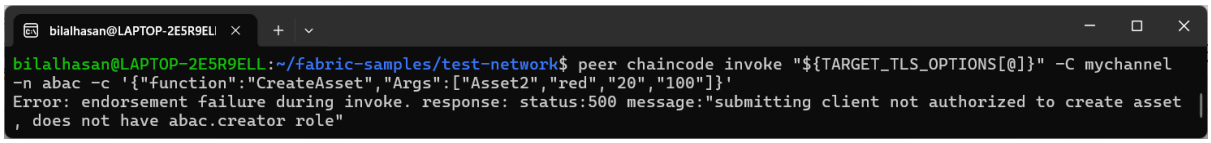


While we are operating as User1, we can demonstrate attribute based access control by trying to create an asset using an identity without the abac.creator=true attribute. Run the following command to try to create Asset1 as User1:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function":"CreateAsset","Args":["Asset2","red","20","100"]}'

The smart contract will return the following error:

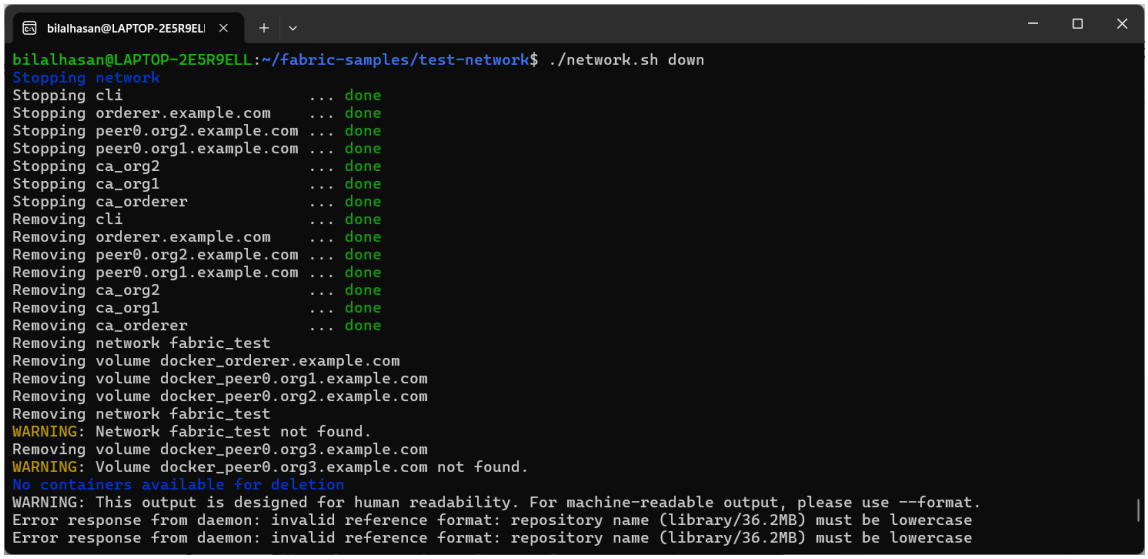
Error: endorsement failure during invoke. response: status:500 message:"submitting client not authorized to create asset, does not have abac.creator role"



Clean up

When you are finished, you can run the following command to bring down the test network:

bilalhasan@LAPTOP-2E5R9ELL:~/fabric-samples/test-network$ ./network.sh down



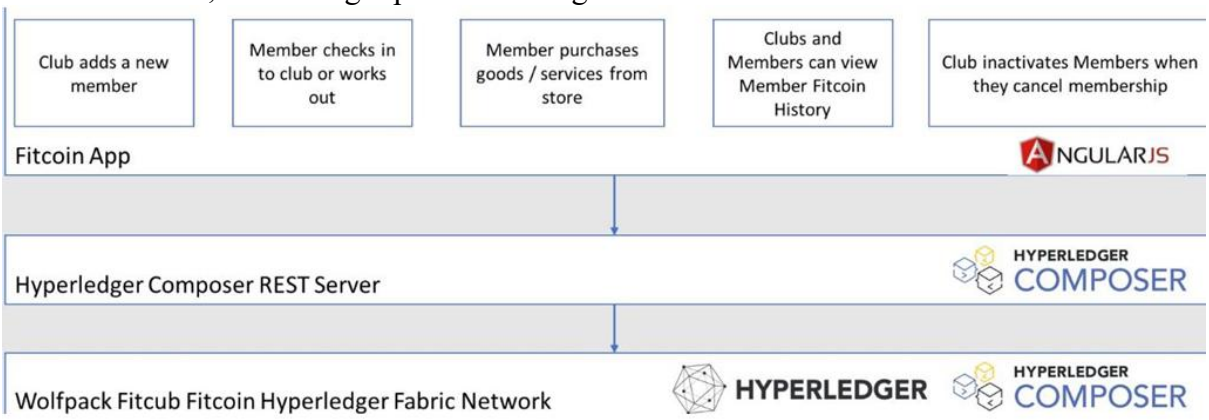
Practical No. 5

Aim - Use block chain to track fitness club rewards.

Objective: To implement a loyalty or rewards program where points are immutable and transparent.

Theory:

- Tokenization of Value: Rewards points are digital assets. Using blockchain ensures that points cannot be double-spent or forged.
- Identity & Membership: In a fitness club scenario, the blockchain verifies the identity of the member (User) and the club (Admin).
- Transaction Flow:
  - Earn: A member completes a workout -> IoT sensor or manual entry triggers a transaction -> Smart contract calculates points -> Ledger is updated.
  - Burn (Redeem): Member requests a reward -> Smart contract checks balance -> Deducts points -> Records exchange.
- Benefit: This eliminates disputes between the club and members regarding point balances, as the ledger provides a single source of truth.



Steps:

1. Install Hyperledger Fabric and Composer.

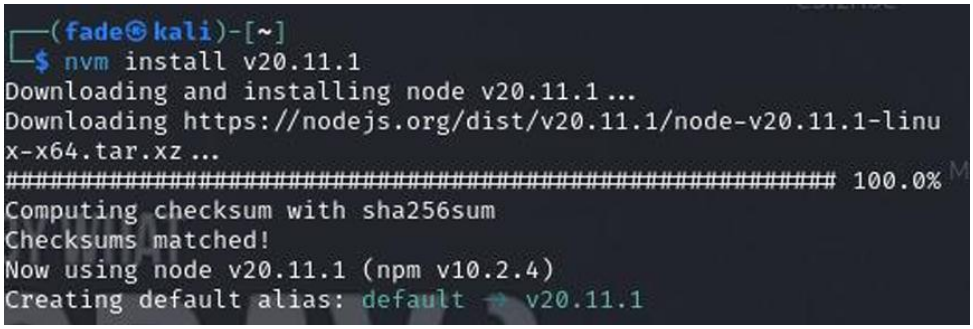
```
rdnc@ubuntu:~/Desktop$ curl -O https://hyperledger.github.io/composer/latest/pre
reqs-ubuntu.sh && chmod u+x prereqs-ubuntu.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4001  100  4001    0     0   9830      0 --:--:-- --:--:-- --:--:--  9830
```

2. Build and Deploy the Fitcoin Blockchain Network.

3. Build and run the Fitcoin Angular Web App.

Note:
If you're running on Ubuntu, you can download the prerequisites using the following commands:
curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh
chmod u+x prereqs-ubuntu.sh

Next run the script - as this briefly uses sudo during its execution, you will be prompted for your
password.
./prereqs-ubuntu.sh
Install the latest (long term support) version of Node:
nvm install --lts
Switch to the LTS version of Node:
nvm use --lts
Check that Node is installed:
node –version



Practical No. 6

Aim - Build a web app that uses Hyperledger Fabric to track and trace member rewards.

Objective: To learn how to build a user-friendly frontend (Web App) that interacts with the backend blockchain network.

Theory:

- Application Layers:
  1. Frontend: HTML/CSS/JS interface (e.g., Angular, React) where the user clicks "Check Balance".
  2. Middleware (API Layer): Usually a Node.js Express server. It receives the request from the frontend and uses the Hyperledger Fabric SDK to talk to the blockchain.
  3. Blockchain Network: The peers and orderers that execute the transaction.
- Track and Trace: This concept is vital for supply chains but applied here to rewards. It allows an admin to query the history of a specific member's rewards (e.g., "Show me the entire history of Member A's points from Jan to Dec").
- Fabric Node SDK: A library that allows the application to connect to the Gateway, submit transactions, and listen for events (e.g., notifying the app when a reward is successfully issued).

Step:

Creating the smart contract project

Installing hardhart

mkdir smart-contract

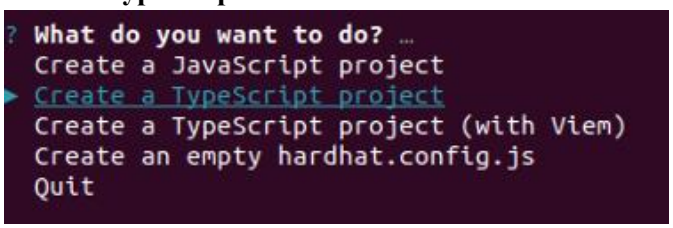
cd smart-contract

npm i -D hardhat

Creating a hardhat project

npx hardhat init

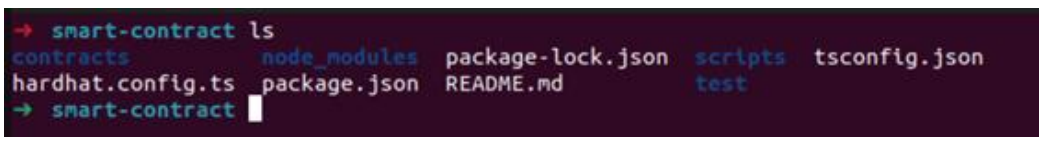
Choose TypeScript



Rest can be left default

Contents after project initialization

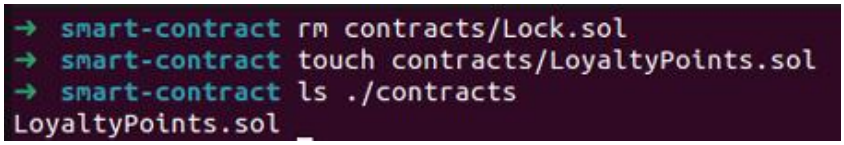
ls



Writing the smart contract:

Navigate the contracts directory and delete the existing Contract Lock.sol and

Create a new Solidity file LoyaltyPoints.sol
rm contracts/Lock.sol
touch contracts/LoyaltyPoints.sol
ls ./contracts



Open the project in an editor and append the below code in LoyaltyPoints.sol

Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract LoyaltyPoints {
    // model a member
    struct Member {
        address memberAddress;
        string firstName;
        string lastName;
        string email;
        uint points;
        bool isRegistered;
    }
    // model a partner
    struct Partner {
        address partnerAddress;
        string name;
        bool isRegistered;
    }
    // model points transaction
    enum TransactionType { Earned, Redeemed }
    struct PointsTransaction {
        uint points;
        TransactionType transactionType;
        address memberAddress;
        address partnerAddress;
    }
    //members and partners on the network mapped with their address
    mapping(address => Member) public members;
    mapping(address => Partner) public partners;
    //public transactions and partners information
    Partner[] public partnersInfo;
    PointsTransaction[] public transactionsInfo;
    //register sender as member
    function registerMember (string memory _firstName, string memory
_lastName, string memory _email) public {
      //check msg.sender in existing members
      require(!members[msg.sender].isRegistered, "Account already registered
as Member");
```

```
      //check msg.sender in existing partners
      require(!partners[msg.sender].isRegistered, "Account already
registered as Partner");
      //add member account
      members[msg.sender] = Member(msg.sender, _firstName, _lastName,
_email, 0, true);
    }
    //register sender as partner
    function registerPartner (string memory _name) public {
      //check msg.sender in existing members
      require(!members[msg.sender].isRegistered, "Account already registered
as Member");
      //check msg.sender in existing partners
      require(!partners[msg.sender].isRegistered, "Account already
registered as Partner");
      //add partner account
      partners[msg.sender] = Partner(msg.sender, _name, true);
      //add partners info to be shared with members
      partnersInfo.push(Partner(msg.sender, _name, true));
    }
    //update member with points earned
    function earnPoints (uint _points, address _partnerAddress ) public {
      // only member can call
      require(members[msg.sender].isRegistered, "Sender not registered as
Member");
      // verify partner address
      require(partners[_partnerAddress].isRegistered, "Partner address not
found");
      // update member account
      members[msg.sender].points = members[msg.sender].points + _points;
      // add transction
      transactionsInfo.push(PointsTransaction({
        points: _points,
        transactionType: TransactionType.Earned,
        memberAddress: members[msg.sender].memberAddress,
        partnerAddress: _partnerAddress
      }));
    }
    //update member with points used
    function usePoints (uint _points, address _partnerAddress) public {
      // only member can call
      require(members[msg.sender].isRegistered, "Sender not registered as Member");
      // verify partner address
      require(partners[_partnerAddress].isRegistered, "Partner address not found");
      // verify enough points for member
      require(members[msg.sender].points >= _points, "Insufficient points");
      // update member account
      members[msg.sender].points = members[msg.sender].points - _points;
```

```
    // add transction
    transactionsInfo.push(PointsTransaction({
      points: _points,
      transactionType: TransactionType.Redeemed,
      memberAddress: members[msg.sender].memberAddress,
      partnerAddress: _partnerAddress
    }));
  }
  //get length of transactionsInfo array
  function transactionsInfoLength() public view returns(uint256) {
    return transactionsInfo.length;
  }
  //get length of partnersInfo array
  function partnersInfoLength() public view returns(uint256) {
    return partnersInfo.length;
  }
}
```

Compiling the smart contract

Save the file and run the following

npx hardhat compile

```
→ smart-contract npx hardhat compile
Downloading compiler 0.8.24
Generating typings for: 1 artifacts in dir: typechain-types for target: ethers-v6
Successfully generated 6 typings!
Compiled 1 Solidity file successfully (evm target: paris).
```

Deploying the smart contract

We will be deploying the smart contract to the local hardhat network.

First we need to write the script that will deploy the smart contract open scripts/deploy.ts and replace its content with below code

Code:

```
import { ethers } from "hardhat";
async function main() {
  const loyaltyPoints = await
ethers.deployContract("LoyaltyPoints");
  await loyaltyPoints.waitForDeployment();
  console.log(`LoyaltyPoints deployed to
${loyaltyPoints.target}`);
}
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

To deploy the contract we need to make sure the hardhat local network is running, open a new terminal in the same project and run the following code and keep it open

npx hardhat node

```
→ smart-contract npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts
========

WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39Fd6e51aad88F6F4ce6aB8827279cfffFb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

Account #1: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8 (10000 ETH)
Private Key: 0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d

Account #2: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC (10000 ETH)
Private Key: 0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a
```

The output gives us the JSON RPC server information and some accounts that we can use

Go back to the previous terminal and deploy the contract

npx hardhat run scripts/deploy.ts --network localhost

```
→ smart-contract npx hardhat run scripts/deploy.ts --network localhost
LoyaltyPoints deployed to 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

The contract is successfully deployed. Save the address for later use

Making the web app

Clone the given repository

git clone https://github.com/IBM/loyalty-points-evm-fabric.git

```
→ prac7 git clone https://github.com/IBM/loyalty-points-evm-fabric.git
Cloning into 'loyalty-points-evm-fabric'...
remote: Enumerating objects: 1497, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 1497 (delta 0), reused 1 (delta 0), pack-reused 1494
Receiving objects: 100% (1497/1497), 4.66 MiB | 9.02 MiB/s, done.
Resolving deltas: 100% (336/336), done.
```

Navigate into the webapp directory and install the dependencies

cd loyalty-points-evm-fabric/web-app

npm i

```
→ prac7 cd loyalty-points-evm-fabric/web-app
→ web-app git:(master) npm i
npm WARN old lockfile
npm WARN old lockfile The package-lock.json file was c
npm WARN old lockfile so supplemental metadata must be
npm WARN old lockfile
npm WARN old lockfile This is a one-time fix-up, pleas
npm WARN old lockfile
npm WARN deprecated har-validator@5.1.0: this library
npm WARN deprecated uuid@3.3.2: Please upgrade  to ver
m for details.
npm WARN deprecated request@2.88.0: request has been c

added 105 packages, and audited 106 packages in 12s
```

Editing the dapp.js config file

There are a series of edits in the dapp.js file

First lets update the contract address and provider. Contract address will be the address it was deployed for you and provider will the JSON RPC url we got when we started the hardhat node

```
var contractAddress = '0x5fbdb2315678afecb367f032d93f642f64180aa3';
var provider = "http://127.0.0.1:8545";
```

Next we need to add contract ABI: for this open the smart contract project and navigate to artifacts/contracts/LoyaltyPoints.sol there will be a json file named LoyaltyPoints.json copy it into your web app project directory

cd artifacts/contracts/LoyaltyPoints.sol
cp LoyaltyPoints.json ../../../../loyalty-points-evm-fabric/web
app
cd ../../../../loyalty-points-evm-fabric/web-app
ls

```
→ smart-contract cd artifacts/contracts/LoyaltyPoints.sol
→ LoyaltyPoints.sol cp LoyaltyPoints.json ../../../../loyalty-points-evm-fabric/web-app
→ LoyaltyPoints.sol cd ../../../../loyalty-points-evm-fabric/web-app
→ web-app git:(master) ✗ ls
app.js  dapp.js  LoyaltyPoints.json  node_modules  package.json  package-lock.json  public  validate.js
```

Changing the code for loyaltyABI variable. Remove loyaltyABI variable and **replace it with the below code**

const fs = require("fs")
const loyaltyABI =
JSON.parse(fs.readFileSync("LoyaltyPoints.json")).abi

New code will look like this

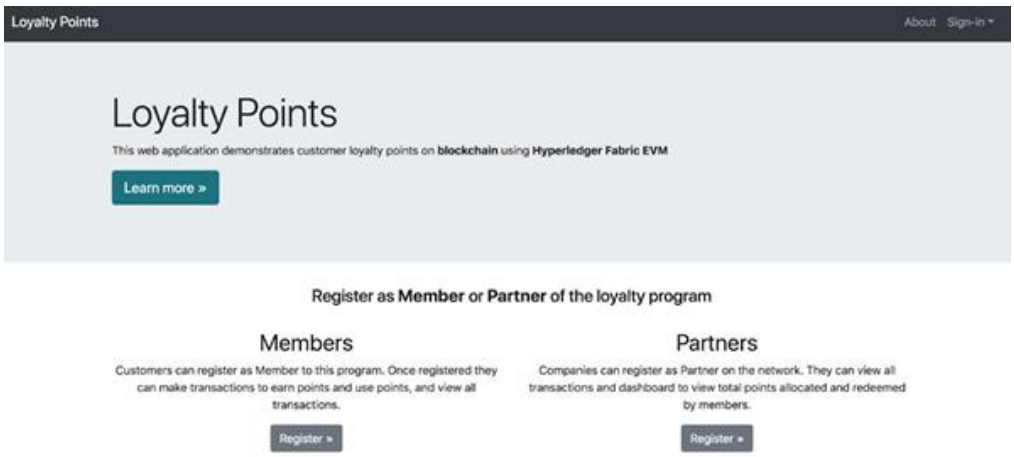
```
var address = contractAddress;

console.log("Got address: " + address)

const fs = require("fs")
const loyaltyABI = JSON.parse(fs.readFileSync("LoyaltyPoints.json")).abi;

var LoyaltyContract = web3.eth.contract(loyaltyABI);
myContract = LoyaltyContract.at(address);
return myContract;
```

Running the app

npm start

```
→ web-app git:(master) ✗ npm start

> loyaly-points@0.0.1 start
> node app.js

app running on port: 8000
```

Open the browser and enter the url http://localhost:8000



Practical No. 7

Aim - Car auction network: A Hello World example with Hyperledger Fabric Node SDK and IBM Block chain Starter Plan. Use Hyperledger Fabric to invoke chaincode while storing results and data in the starter plan.

Objective: To simulate a marketplace where assets (cars) are auctioned to the highest bidder.

Theory:

- State Machine Logic: An auction has distinct states: Open, Bidding, Closed. The smart contract must enforce that bids are only accepted when the state is Open.
- Invoking vs. Querying:
  1. Invoke: A transaction that changes the ledger (e.g., "Make a Bid"). This must be endorsed by peers and ordered.
  2. Query: A read-only operation (e.g., "Check current highest bid"). This is faster as it reads the World State without generating a new block.
- Data Visibility: In a car auction, you might use Private Data Collections if you want the bidder's identity to be kept secret from other bidders but known to the auctioneer.

Code:

Setup the blockchain network

Open a new terminal and make sure the network is down before starting

cd test-network

./network.sh down

```
→ fabric-samples git:(v2.2.0) cd test-network
→ test-network git:(v2.2.0) ./network.sh down
```

Remove any existing/running docker containers

docker rm -f $(docker ps -aq)

docker rmi -f $(docker images | grep fabcar | awk '{print $3}')

```
→ test-network git:(v2.2.0) docker rm -f $(docker ps -aq)
docker rmi -f $(docker images | grep fabcar | awk '{print $3}')

"docker rm" requires at least 1 argument.
See 'docker rm --help'.

Usage:  docker rm [OPTIONS] CONTAINER [CONTAINER...]

Remove one or more containers
"docker rmi" requires at least 1 argument.
See 'docker rmi --help'.

Usage:  docker rmi [OPTIONS] IMAGE [IMAGE...]

Remove one or more images
→ test-network git:(v2.2.0) S
```

Starting the network

For this Network we will be using Javascript

cd fabcar

./startFabric.sh javascript

```
→ fabric-samples git:(v2.2.0) cd fabcar
→ fabcar git:(v2.2.0) ./startFabric.sh javascript
~/web3/fabric-samples/test-network ~/web3/fabric-samples/fabcar
Stopping network
Removing network fabric_test
WARNING: Network fabric_test not found.
```

You need to be in the javascript directory

Run the following command to install the Fabric dependencies for the applications. It will take about a minute to complete:

cd javascript

npm i

```
→ fabcar git:(v2.2.0) cd javascript
→ javascript git:(v2.2.0) npm i
npm WARN deprecated uuid@3.4.0: Please upgrade  to version 7 or higher.
th.random() in certain circumstances, which is known to be problematic.
ath-random for details.
npm WARN deprecated mkdirp@0.5.1: Legacy versions of mkdirp are no longe
 to mkdirp 1.x. (Note that the API surface has changed to use Promises i
npm WARN deprecated sinon@7.5.0: 16.1.1

added 358 packages, and audited 359 packages in 29s
```

Once npm i completes, everything is in place to run the application. For this tutorial, you'll primarily be using the application JavaScript files in the fabcar/javascript directory. Let's take a look at what's inside:

ls

```
→ javascript git:(v2.2.0) ls
enrollAdmin.js  node_modules  package-lock.json  registerUser.js
invoke.js       package.json  query.js           wallet
→ javascript git:(v2.2.0)
```

Enrolling the admin user

We will subsequently register and enroll a new application user which will be used by our application to interact with the blockchain.

node enrollAdmin.js

```
→ javascript git:(v2.2.0) node enrollAdmin.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Successfully enrolled admin user "admin" and imported it into the wallet
```

Register and Enroll

Now that we have the administrator's credentials in a wallet, we can enroll a new user user1 which will be used to query and update the ledger:

node registerUser.js

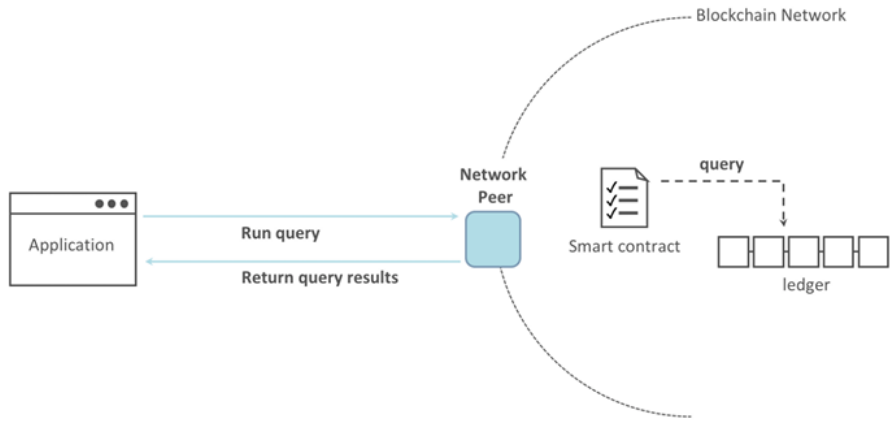
```
→ javascript git:(v2.2.0) node registerUser.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Successfully registered and enrolled admin user "appUser" and imported it into the wallet
→ javascript git:(v2.2.0)
```

Similar to the admin enrollment, this program uses a CSR to enroll user1 and store its credentials alongside those of admin in the wallet. We now have identities for two separate users admin and user1 and these are used by our application.

Querying the ledger

Each peer in a blockchain network hosts a copy of the ledger, and an application program can query the ledger by invoking a smart contract which queries the most recent value of the ledger and returns it to the application.

Here is a simplified representation of how a query works:



First, let's run our query.js program to return a listing of all the cars on the ledger. This program uses our second identity – user1 – to access the ledger:

node query.js

```
→ javascript git:(v2.2.0) node query.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Transaction has been evaluated, result is: [{"Key":"CAR0","Record":{"color":"blue","docType":"car"
,"make":"Toyota","model":"Prius","owner":"Tomoko"}},{"Key":"CAR1","Record":{"color":"red","docType
":"car","make":"Ford","model":"Mustang","owner":"Brad"}},{"Key":"CAR2","Record":{"color":"green","
docType":"car","make":"Hyundai","model":"Tucson","owner":"Jin Soo"}},{"Key":"CAR3","Record":{"colo
r":"yellow","docType":"car","make":"Volkswagen","model":"Passat","owner":"Max"}},{"Key":"CAR4","Re
cord":{"color":"black","docType":"car","make":"Tesla","model":"S","owner":"Adriana"}},{"Key":"CAR5
","Record":{"color":"purple","docType":"car","make":"Peugeot","model":"205","owner":"Michel"}},{"K
ey":"CAR6","Record":{"color":"white","docType":"car","make":"Chery","model":"S22L","owner":"Aarav"
}},{"Key":"CAR7","Record":{"color":"violet","docType":"car","make":"Fiat","model":"Punto","owner":
"Part"}},{"Key":"CAR8","Record":{"color":"indigo","docType":"car","make":"Tata","model":"Nano","ow
ner":"Valeria"}},{"Key":"CAR9","Record":{"color":"brown","docType":"car","make":"Holden","model":"
Barina","owner":"Shotaro"}}]
→ javascript git:(v2.2.0)
```

Changing query.js code

change the evaluateTransaction request to query CAR4.The query program should now look like this:

const result = await contract.evaluateTransaction('queryCar', 'CAR4');

```
const result = await contract.evaluateTransaction('queryCar', 'CAR4');
console.log(`Transaction has been evaluated, result is: ${result.toString()}`);
```

Save the program and run the query program again:

node query.js

```
→ javascript git:(v2.2.0) ✗ node query.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Transaction has been evaluated, result is: {"color":"black","docType":"car","make":"Tesla","model":"S","owner":"Adriana"}
→ javascript git:(v2.2.0) ✗
```

Updating the ledger

Our first update to the ledger will create a new car. We have a separate program called invoke.js that we will use to make updates to the ledger. Just as with queries, use an editor to open the program and navigate to the code block where we construct our transaction and submit it to the network:

```
await contract.submitTransaction('createCar', 'CAR12', 'Honda', 'Accord', 'Black', 'Tom');
console.log('Transaction has been submitted');
```

Run the program

node invoke.js

```
→ javascript git:(v2.2.0) ✗ node invoke.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Transaction has been submitted
→ javascript git:(v2.2.0) ✗
```

Changing the query.js to fetch the new transaction

```
const result = await contract.evaluateTransaction('queryCar', 'CAR12');
console.log(`Transaction has been evaluated, result is: ${result.toString()}`);
```

Save and run

node query.js

```
→ javascript git:(v2.2.0) ✗ node query.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Transaction has been evaluated, result is: {"color":"Black","docType":"car","make":"Honda","model":"Accord","owner":"Tom"}
→ javascript git:(v2.2.0) ✗
```

Changing the ownership of car by making changes in invoke.js To do this, go back to invoke.js and change the smart contract transaction from createCar to changeCarOwner with a corresponding change in input arguments

await contract.submitTransaction('changeCarOwner', 'CAR12', 'Dave');

```
// await contract.submitTransaction('createCar', 'CAR12', 'Honda', 'Accord', 'Black', 'Tom');
await contract.submitTransaction('changeCarOwner', 'CAR12', 'Dave');
console.log('Transaction has been submitted');
```

Save and run

node invoke.js

```
→ javascript git:(v2.2.0) ✗ node invoke.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Transaction has been submitted
→ javascript git:(v2.2.0) ✗
```

Verifying if the Owner for Car12 is updated

node query.js

```
→ javascript git:(v2.2.0) ✗ node query.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Transaction has been evaluated, result is: {"color":"Black","docType":"car","make":"Honda","model":"Accord","owner":"Dave"}
→ javascript git:(v2.2.0) ✗
```

The ownership of CAR12 has been changed from Tom to Dave.

Practical No. 8

Aim - Develop a voting application using Hyperledger and Ethereum.

Objective: To explore electronic voting (e-voting) where security, anonymity, and immutability are paramount.

Theory:

- The E-Voting Challenge: A voting system must ensure:
  1. Eligibility: Only allowed users can vote.
  2. Uniqueness: Users can only vote once.
  3. Privacy: No one can link a vote to a specific voter.
  4. Integrity: No one can change the vote count.

- Hyperledger (Permissioned) Approach: Good for corporate or internal voting. The MSP handles identity (voter registration), but the vote itself must be anonymized (often using zero-knowledge proofs or separated databases).

- Ethereum (Public) Approach: Good for transparent public voting.
  1. Smart Contract: Stores the candidates and vote counts.
  2. Gas: The cost to execute the vote transaction.
  3. Metamask: Used as the wallet/identity for the voter to sign their ballot.

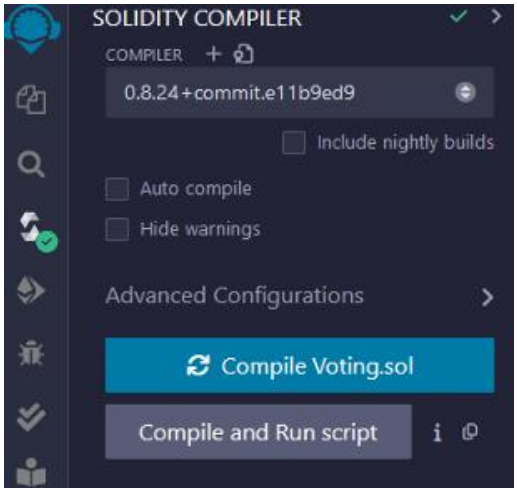
Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Voting {
    // Declare variables to store the total votes for each team
    uint256 public votesForTeamA;
    uint256 public votesForTeamB;
    uint256 public votesForTeamC;
    address public owner;
    mapping(address => bool) authorizedVoters;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyAuthorizedVoter() {
        require(
            authorizedVoters[msg.sender] == true,
            "You are not authorized to vote"
        );
```

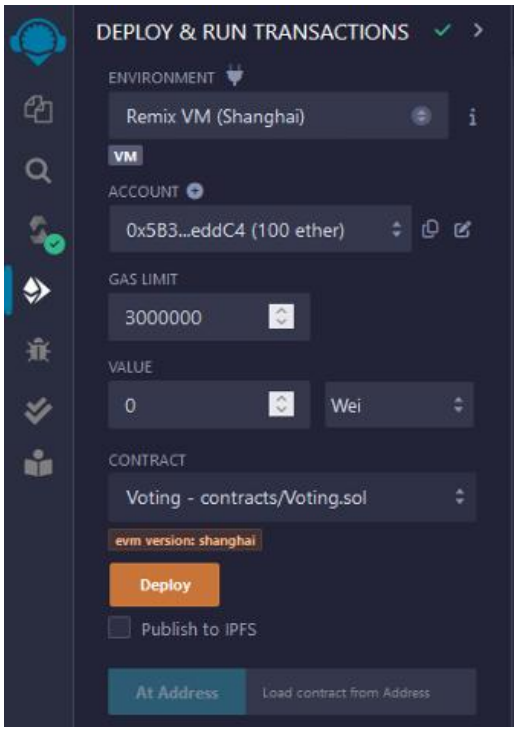
```
        _;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can call this
function");
        _;
    }

    // Function to allow a voter to cast their vote for a team
    function vote(uint256 _team) public onlyAuthorizedVoter {
        // Check the value of _team and increment the corresponding team's
vote count
        if (_team == 1) {
            votesForTeamA += 1;
        } else if (_team == 2) {
            votesForTeamB += 1;
        } else if (_team == 3) {
            votesForTeamC += 1;
        }
    }

    // Function to add a voter to the list of authorized voters
    function addVoter(address _voter) public onlyOwner {
        require(msg.sender == owner, "Only the owner can add a voter");
        authorizedVoters[_voter] = true;
    }

    // Function to declare results of the vote
    function getWinner() public view returns (string memory result) {
        // Check the vote counts and return the winner
        if (votesForTeamA > votesForTeamB && votesForTeamA > votesForTeamC)
{
            result = "Team A is the winner!";
        } else if (
            votesForTeamB > votesForTeamA && votesForTeamB > votesForTeamC
        ) {
            result = "Team B is the winner!";
        } else if (
            votesForTeamC > votesForTeamA && votesForTeamC > votesForTeamB
        ) {
            result = "Team C is the winner!";
        } else {
            result = "There is a tie!";
        }
    }

    function getTotalVotes() public view returns (uint256) {
```

```
        return votesForTeamA + votesForTeamB + votesForTeamC;
    }
}
```

Compile The Contract

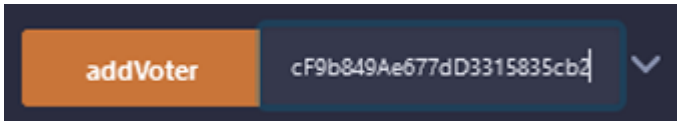


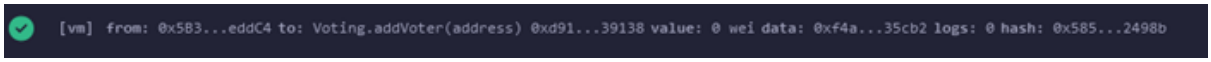
Deploy The Contract



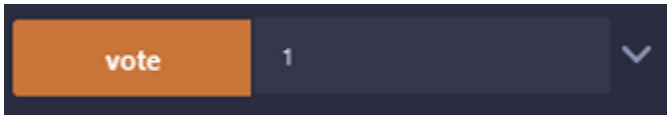
Adding voters

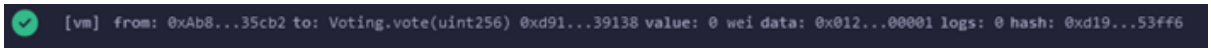
(Note: you can get wallet address from above dropdown; change it to another and copy it)





Voting





Checking the winner