INDEX

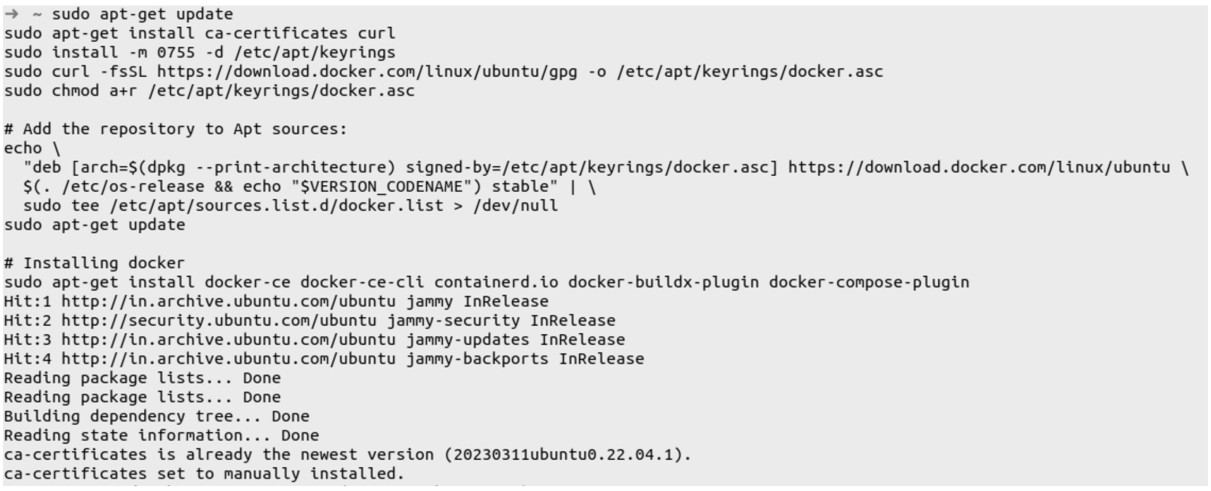
Practical 1

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.
(Note: All the practicals are performed in Linux (Ubuntu))

Docker:

Installation:

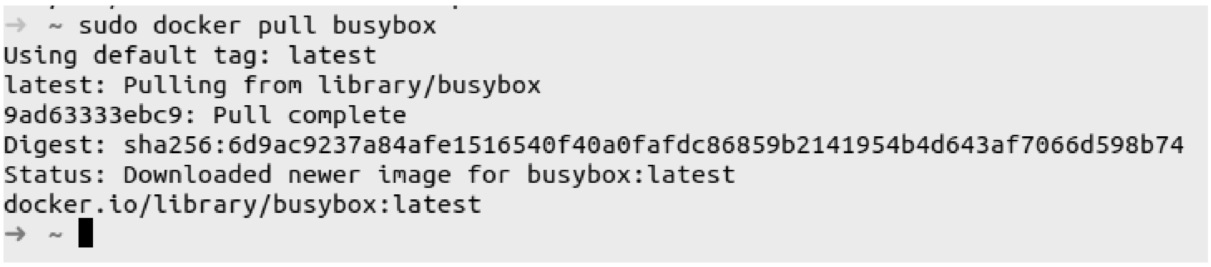
```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
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

# Installing docker
sudo apt-get install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin docker-compose

# Adding current user to docker group
# (This gives user root privileges so be careful this is
# just for our convenience )
sudo usermod -aG docker $USER
newgrp docker

# reboot the system to take effect
```



```
➜ ~ sudo apt-get update
sudo apt-get install 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

# Add the repository to Apt sources:
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

# Installing docker
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
Hit:1 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:3 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:4 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
ca-certificates set to manually installed.
```

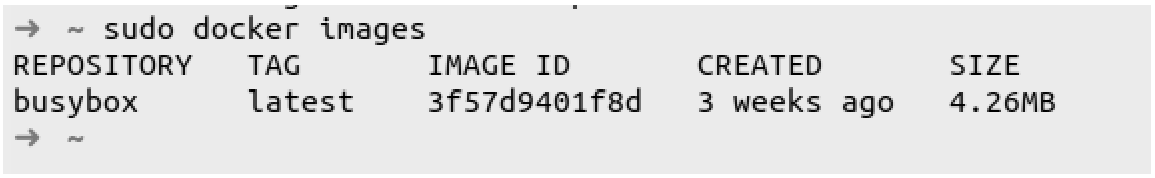
Verify if docker is working

sudo docker pull busybox



```
➜ ~ sudo docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
9ad63333ebc9: Pull complete
Digest: sha256:6d9ac9237a84afe1516540f40a0fafdc86859b2141954b4d643af7066d598b74
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
➜ ~
```

sudo docker images

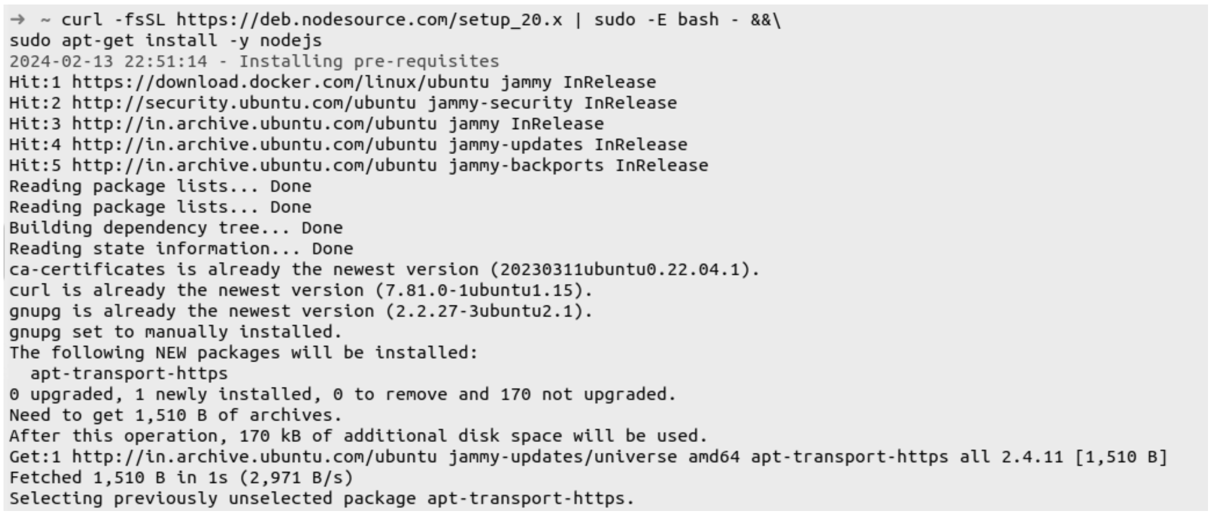


```
➜ ~ sudo docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
busybox      latest    3f57d9401f8d   3 weeks ago   4.26MB
➜ ~
```

Node.js

Installation:

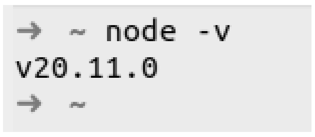
```
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
&&\
sudo apt-get install -y nodejs
```



```
→ ~ curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash - &&\
sudo apt-get install -y nodejs
2024-02-13 22:51:14 - Installing pre-requisites
Hit:1 https://download.docker.com/linux/ubuntu jammy InRelease
Hit:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:3 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Hit:4 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:5 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
curl is already the newest version (7.81.0-1ubuntu1.15).
gnupg is already the newest version (2.2.27-3ubuntu2.1).
gnupg set to manually installed.
The following NEW packages will be installed:
  apt-transport-https
0 upgraded, 1 newly installed, 0 to remove and 170 not upgraded.
Need to get 1,510 B of archives.
After this operation, 170 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 apt-transport-https all 2.4.11 [1,510 B]
Fetched 1,510 B in 1s (2,971 B/s)
Selecting previously unselected package apt-transport-https.
```

Verifying the version of node.js

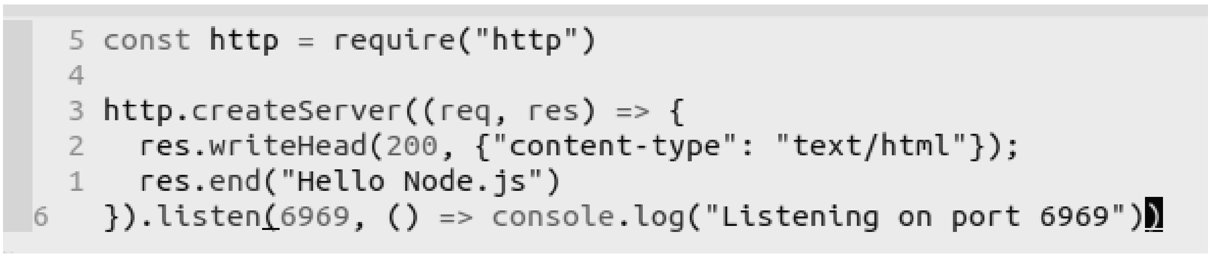
node -v



```
→ ~ node -v
v20.11.0
→ ~
```

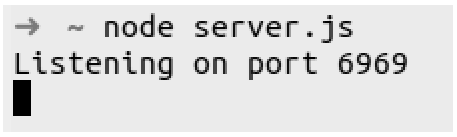
Creating a simple web server in node.js

Create a file named server.js with the following contents



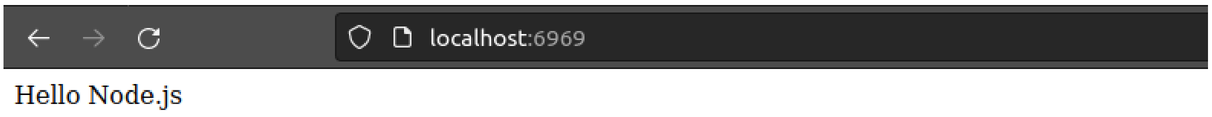
```
const http = require("http")

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

Run the server using: **node server.js**



```
→ ~ node server.js
Listening on port 6969
```

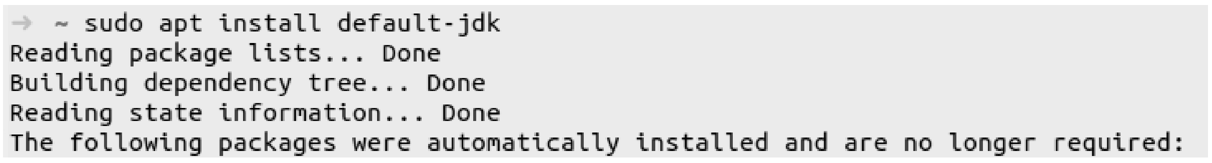
Now open the browser and type http://localhost:6969 you will see a webpage



Hello Node.js

Installing Java

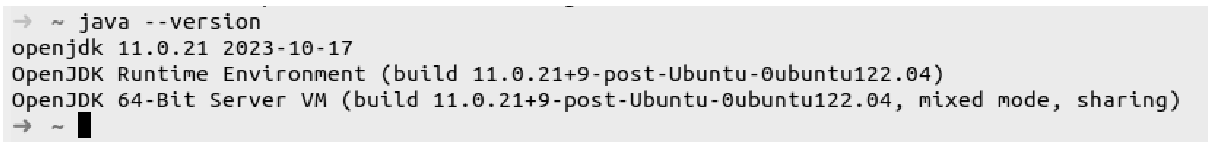
```
sudo apt install default-jdk
```



```
→ ~ sudo apt install default-jdk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
```

Verifying java installation

```
java --version
```



```
→ ~ java --version
openjdk 11.0.21 2023-10-17
OpenJDK Runtime Environment (build 11.0.21+9-post-Ubuntu-0ubuntu122.04)
OpenJDK 64-Bit Server VM (build 11.0.21+9-post-Ubuntu-0ubuntu122.04, mixed mode, sharing)
→ ~
```

Golang:
Download latest version of go from the website for linux (instructions)

```
 sudo rm -rf /usr/local/go && sudo tar -C /usr/local -xzf
go1.22.0.linux-amd64.tar.gz
```

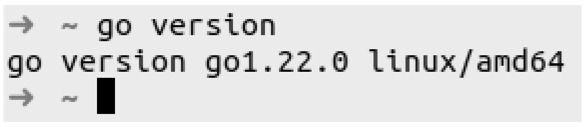
Append the below line in your rc file (~/.bashrc or ~/.zshrc)

```
export PATH=$PATH:/usr/local/go/bin
```

Once done restart your terminal

Verify golang

```
go version
```



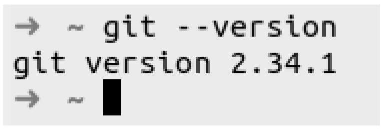
```
→ ~ go version
go version go1.22.0 linux/amd64
→ ~
```

Git:

```
sudo apt install git
```

Verify git

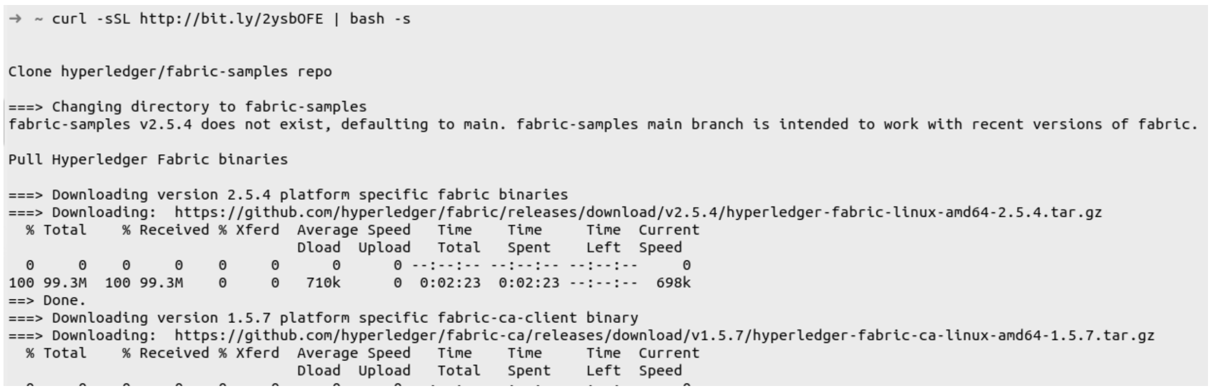
```
git --version
```



```
➜  ~ git --version
git version 2.34.1
➜  ~ █
```

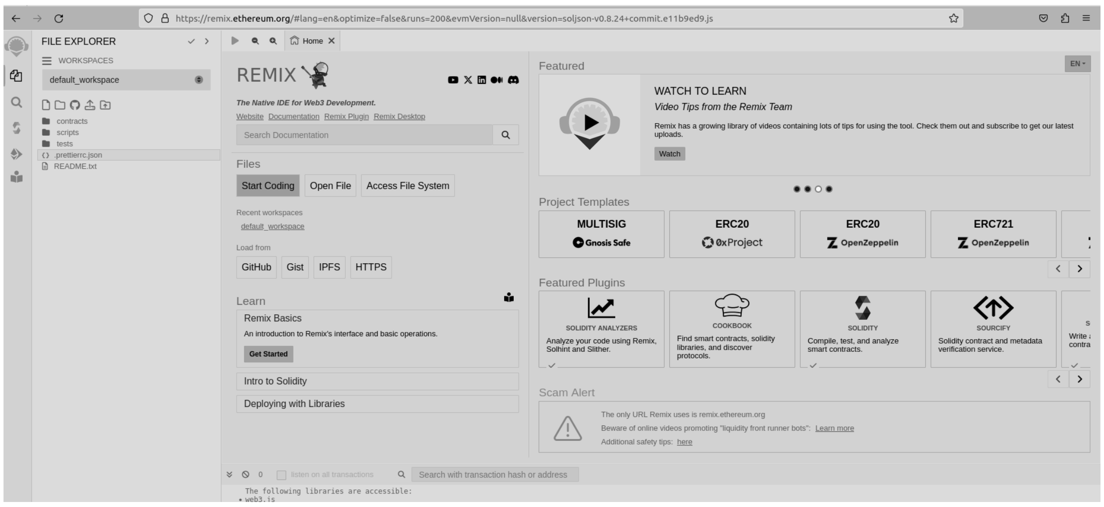
Hyperledger Fabric:
Before installing hyper ledger make sure docker is up and running (**sudo systemctl status docker)**

```
curl -sSL http://bit.ly/2ysbOFE | bash -s 2.2.0
```



```
➜ ~ curl -sSL http://bit.ly/2ysbOFE | bash -s


Clone hyperledger/fabric-samples repo

===> Changing directory to fabric-samples
fabric-samples v2.5.4 does not exist, defaulting to main. fabric-samples main branch is intended to work with recent versions of fabric.

Pull Hyperledger Fabric binaries

===> Downloading version 2.5.4 platform specific fabric binaries
===> Downloading:  https://github.com/hyperledger/fabric/releases/download/v2.5.4/hyperledger-fabric-linux-amd64-2.5.4.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 99.3M  100 99.3M    0     0   710k      0  0:02:23  0:02:23 --:--:--  698k
==> Done.
===> Downloading version 1.5.7 platform specific fabric-ca-client binary
===> Downloading:  https://github.com/hyperledger/fabric-ca/releases/download/v1.5.7/hyperledger-fabric-ca-linux-amd64-1.5.7.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

Ethereum:
Running Ethereum IDE online (Remix IDE)


https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.24+commit.e11b9ed9.js
FILE EXPLORER
WORKSPACES
default_workspace
contracts
scripts
tests
.prettierrc.json
README.txt
Home
REMIX
The Native IDE for Web3 Development.
Website Documentation Remix Plugin Remix Desktop
Search Documentation
Files
Start Coding
Open File
Access File System
Recent workspaces
default_workspace
Load from
GitHub
Gist
IPFS
HTTPS
Learn
Remix Basics
An introduction to Remix's interface and basic operations.
Get Started
Intro to Solidity
Deploying with Libraries
Featured
WATCH TO LEARN
Video Tips from the Remix Team
Remix has a growing library of videos containing lots of tips for using the tool. Check them out and subscribe to get our latest uploads.
Watch
Project Templates
MULTISIG
Gnosis Safe
ERC20
0xProject
ERC20
OpenZeppelin
ERC721
OpenZeppelin
Featured Plugins
SOLIDITY ANALYZERS
Analyze your code using Remix, Solhint and Slither.
COOKBOOK
Find smart contracts, solidity libraries, and discover protocols.
SOLIDITY
Compile, test, and analyze smart contracts.
SOURCIFY
Solidity contract and metadata verification service.
Scam Alert
The only URL Remix uses is remix.ethereum.org
Beware of online videos promoting "liquidity front runner bots": Learn more
Additional safety tips: here
listen on all transactions
Search with transaction hash or address
The following libraries are accessible:
web3.js

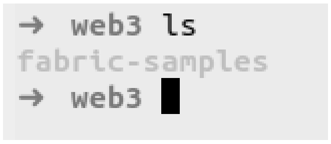
Practical 2

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

Install Hyperledger (If not already done)

```
curl -sSL http://bit.ly/2ysbOFE | bash -s 2.2.0
```

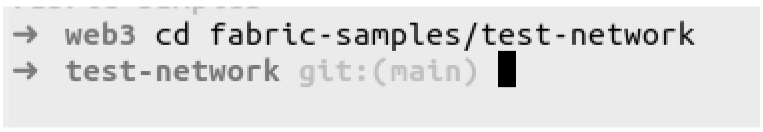
Once its installed there will be a directory named fabric-samples



```
➜  web3 ls
fabric-samples
➜  web3 █
```

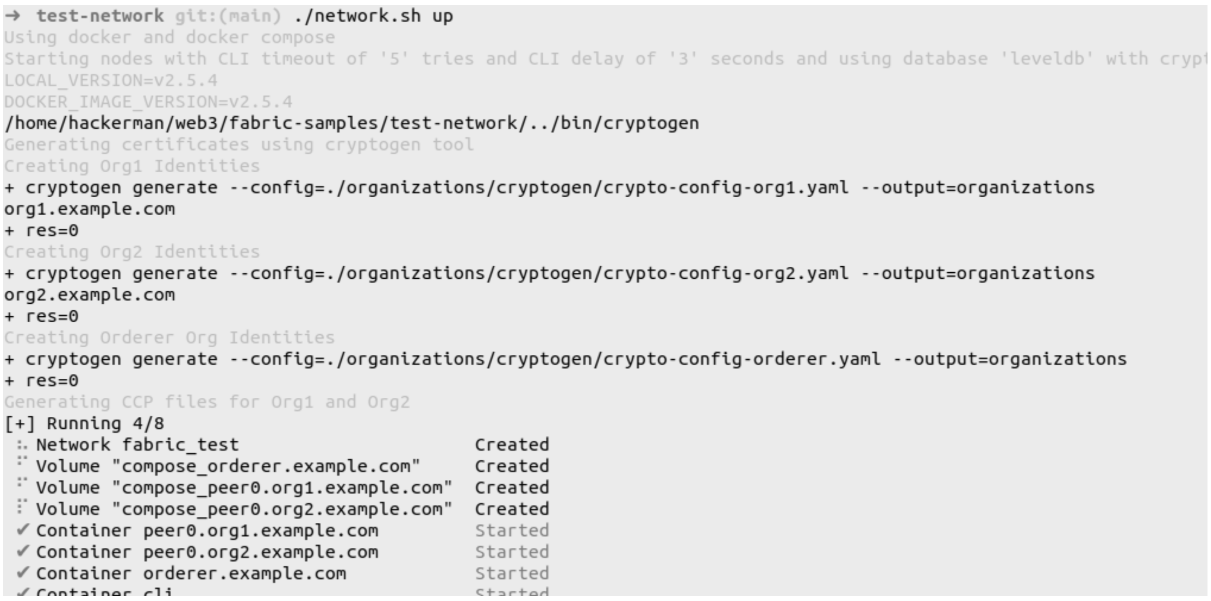
Starting the hyperledger test network

```
cd fabric-samples/test-network
```



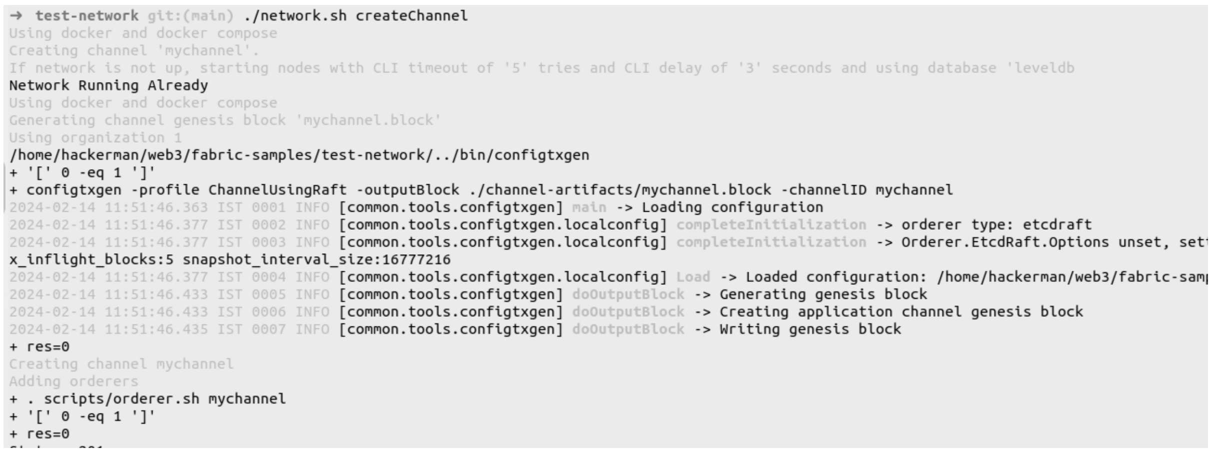
```
➜  web3 cd fabric-samples/test-network
➜  test-network git:(main) █
```

```
./network.sh up
```



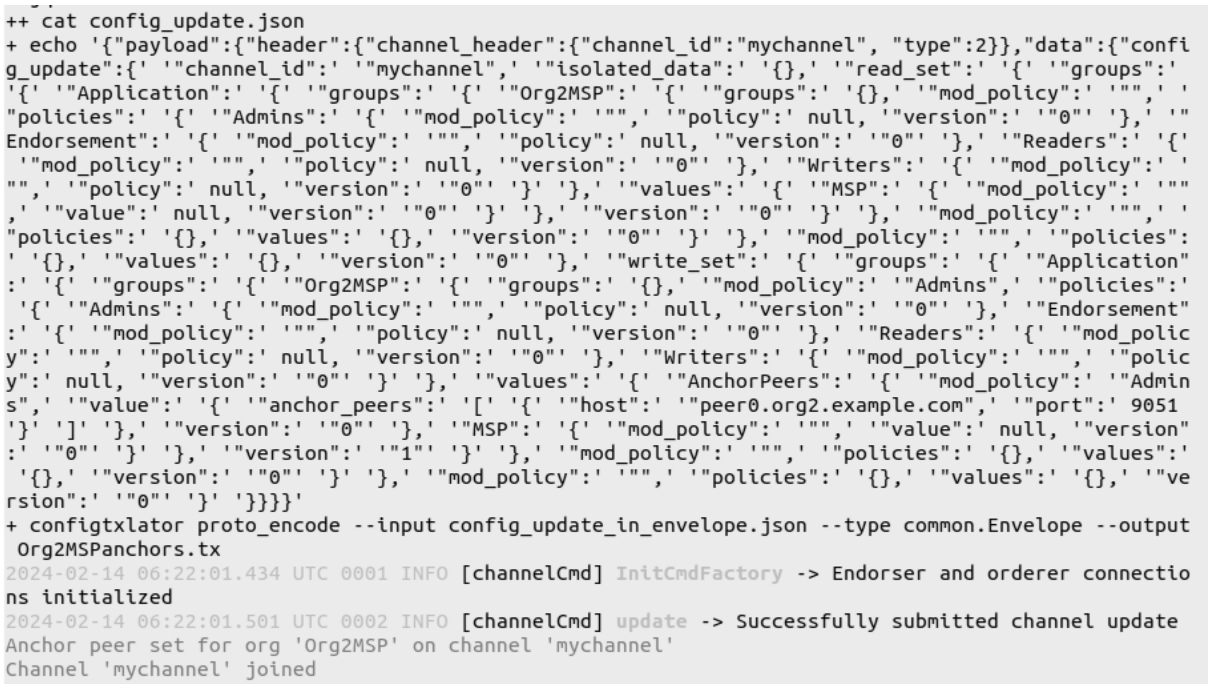
```
➜  test-network git:(main) ./network.sh up
Using docker and docker compose
Starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb' with crypt
LOCAL_VERSION=v2.5.4
DOCKER_IMAGE_VERSION=v2.5.4
/home/hackerman/web3/fabric-samples/test-network/../bin/cryptogen
Generating certificates using cryptogen tool
Creating Org1 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org1.yaml --output=organizations
org1.example.com
+ res=0
Creating Org2 Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-org2.yaml --output=organizations
org2.example.com
+ res=0
Creating Orderer Org Identities
+ cryptogen generate --config=./organizations/cryptogen/crypto-config-orderer.yaml --output=organizations
+ res=0
Generating CCP files for Org1 and Org2
[+] Running 4/8
 :. Network fabric_test                      Created
 :: Volume "compose_orderer.example.com"     Created
 :: Volume "compose_peer0.org1.example.com"  Created
 :: Volume "compose_peer0.org2.example.com"  Created
 ✓ Container peer0.org1.example.com          Started
 ✓ Container peer0.org2.example.com          Started
 ✓ Container orderer.example.com             Started
 ✓ Container cli                             Started
```

Creating HyperLedger channels



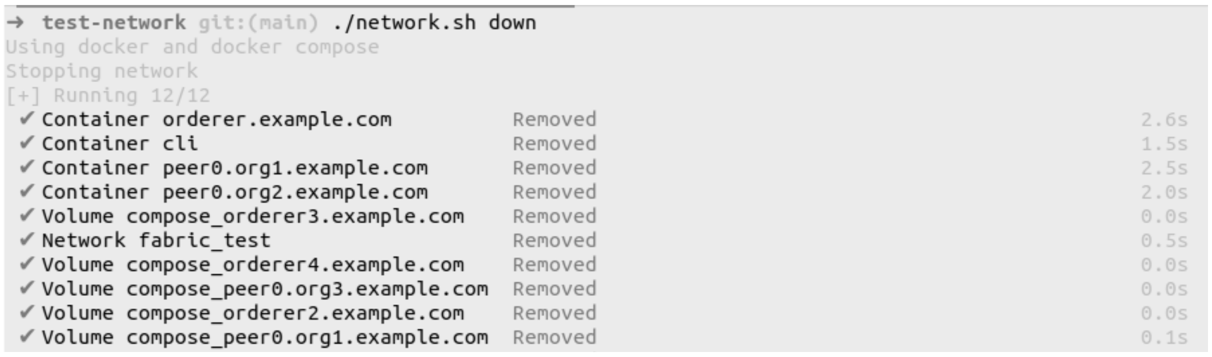
```
➜  test-network git:(main) ./network.sh createChannel
Using docker and docker compose
Creating channel 'mychannel'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb
Network Running Already
Using docker and docker compose
Generating channel genesis block 'mychannel.block'
Using organization 1
/home/hackerman/web3/fabric-samples/test-network/../bin/configtxgen
+ '[' 0 -eq 1 ']'
+ configtxgen -profile ChannelUsingRaft -outputBlock ./channel-artifacts/mychannel.block -channelID mychannel
2024-02-14 11:51:46.363 IST 0001 INFO [common.tools.configtxgen] main -> Loading configuration
2024-02-14 11:51:46.377 IST 0002 INFO [common.tools.configtxgen.localconfig] completeInitialization -> orderer type: etcdraft
2024-02-14 11:51:46.377 IST 0003 INFO [common.tools.configtxgen.localconfig] completeInitialization -> Orderer.EtcdRaft.Options unset, sett
x_inflight_blocks:5 snapshot_interval_size:16777216
2024-02-14 11:51:46.377 IST 0004 INFO [common.tools.configtxgen.localconfig] Load -> Loaded configuration: /home/hackerman/web3/fabric-samp
2024-02-14 11:51:46.433 IST 0005 INFO [common.tools.configtxgen] doOutputBlock -> Generating genesis block
2024-02-14 11:51:46.433 IST 0006 INFO [common.tools.configtxgen] doOutputBlock -> Creating application channel genesis block
2024-02-14 11:51:46.435 IST 0007 INFO [common.tools.configtxgen] doOutputBlock -> Writing genesis block
+ res=0
Creating channel mychannel
Adding orderers
+ . scripts/orderer.sh mychannel
+ '[' 0 -eq 1 ']'
+ res=0
```

Checking the configs



```
++ cat config_update.json
+ echo '{"payload":{"header":{"channel_header":{"channel_id":"mychannel", "type":2}},"data":{"confi
g_update":{' '"channel_id":' '"mychannel",' '"isolated_data":' '{},' '"read_set":' '{' '"groups":'
'{' '"Application":' '{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":' '"",' '
"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"
Endorsement":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{'
 '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Writers":' '{' '"mod_policy":' '"
",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"MSP":' '{' '"mod_policy":' '""
,' '"value":' null, '"version":' '"0"' '}' '},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '
"policies":' '{},' '"values":' '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":
' '{},' '"values":' '{},' '"version":' '"0"' '},' '"write_set":' '{' '"groups":' '{' '"Application"
:' '{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":' '"Admins",' '"policies":'
 '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Endorsement"
:' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_polic
y":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Writers":' '{' '"mod_policy":' '"",' '"polic
y":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"AnchorPeers":' '{' '"mod_policy":' '"Admin
s",' '"value":' '{' '"anchor_peers":' '[' '{' '"host":' '"peer0.org2.example.com",' '"port":' 9051
'}' ']' '},' '"version":' '"0"' '},' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"version"
:' '"0"' '}' '},' '"version":' '"1"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":'
 '{},' '"version":' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"ve
rsion":' '"0"' '}' '}}}}'
+ configtxlator proto_encode --input config_update_in_envelope.json --type common.Envelope --output
 Org2MSPanchors.tx
2024-02-14 06:22:01.434 UTC 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer connectio
ns initialized
2024-02-14 06:22:01.501 UTC 0002 INFO [channelCmd] update -> Successfully submitted channel update
Anchor peer set for org 'Org2MSP' on channel 'mychannel'
Channel 'mychannel' joined
```

Shutting down the network



```
➜  test-network git:(main) ./network.sh down
Using docker and docker compose
Stopping network
[+] Running 12/12
 ✔ Container orderer.example.com        Removed                                   2.6s
 ✔ Container cli                        Removed                                   1.5s
 ✔ Container peer0.org1.example.com     Removed                                   2.5s
 ✔ Container peer0.org2.example.com     Removed                                   2.0s
 ✔ Volume compose_orderer3.example.com  Removed                                   0.0s
 ✔ Network fabric_test                  Removed                                   0.5s
 ✔ Volume compose_orderer4.example.com  Removed                                   0.0s
 ✔ Volume compose_peer0.org3.example.com Removed                                  0.0s
 ✔ Volume compose_orderer2.example.com  Removed                                   0.0s
 ✔ Volume compose_peer0.org1.example.com Removed                                  0.1s
```

Practical 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.

Installing dependencies

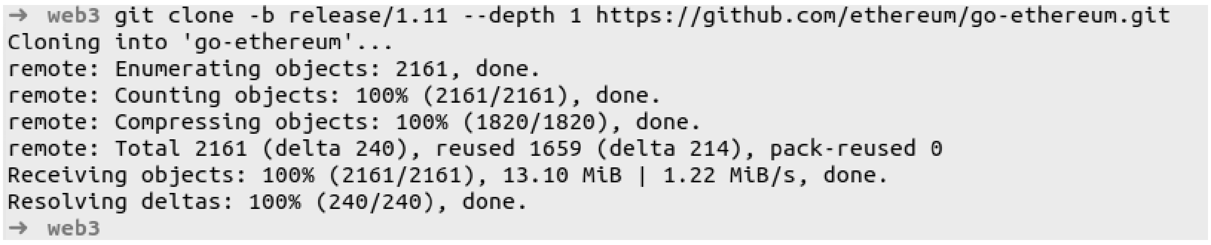
```
sudo apt install make
```

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.

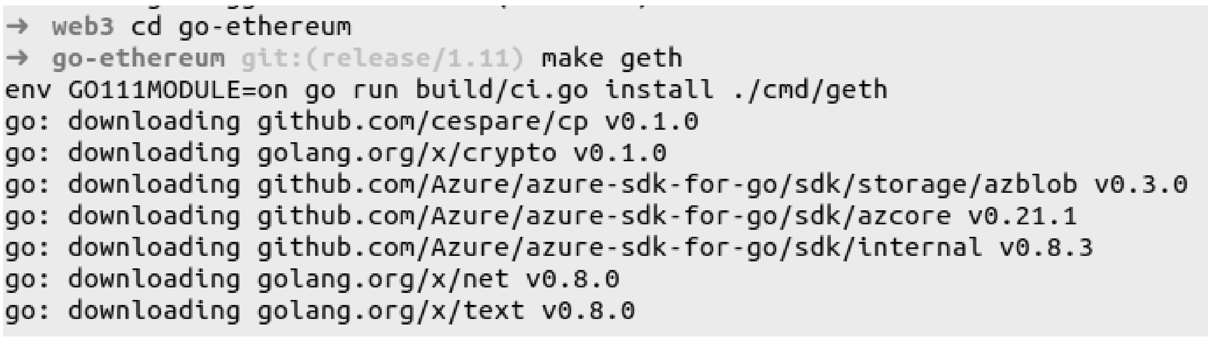
To install Geth in Ubuntu/Debian follow the following steps:

```
git clone -b release/1.11 --depth 1
https://github.com/ethereum/go-ethereum.git
```

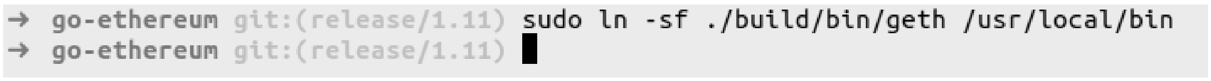


```
➜ web3 git clone -b release/1.11 --depth 1 https://github.com/ethereum/go-ethereum.git
Cloning into 'go-ethereum'...
remote: Enumerating objects: 2161, done.
remote: Counting objects: 100% (2161/2161), done.
remote: Compressing objects: 100% (1820/1820), done.
remote: Total 2161 (delta 240), reused 1659 (delta 214), pack-reused 0
Receiving objects: 100% (2161/2161), 13.10 MiB | 1.22 MiB/s, done.
Resolving deltas: 100% (240/240), done.
➜ web3
```

```
cd go-ethereum
make geth
sudo ln -f ./build/bin/geth /usr/local/bin
```

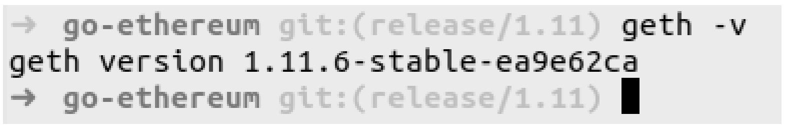


```
➜ web3 cd go-ethereum
➜ go-ethereum git:(release/1.11) make geth
env GO111MODULE=on go run build/ci.go install ./cmd/geth
go: downloading github.com/cespare/cp v0.1.0
go: downloading golang.org/x/crypto v0.1.0
go: downloading github.com/Azure/azure-sdk-for-go/sdk/storage/azblob v0.3.0
go: downloading github.com/Azure/azure-sdk-for-go/sdk/azcore v0.21.1
go: downloading github.com/Azure/azure-sdk-for-go/sdk/internal v0.8.3
go: downloading golang.org/x/net v0.8.0
go: downloading golang.org/x/text v0.8.0
```



```
➜ go-ethereum git:(release/1.11) sudo ln -sf ./build/bin/geth /usr/local/bin
➜ go-ethereum git:(release/1.11) █
```

Verifying installation



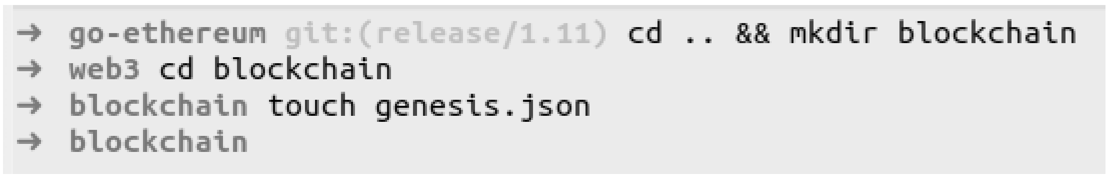
```
➜  go-ethereum git:(release/1.11) geth -v
geth version 1.11.6-stable-ea9e62ca
➜  go-ethereum git:(release/1.11) 
```

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.

```
cd .. && mkdir blockchain
cd blockchain
touch genesis.json
```



```
➜  go-ethereum git:(release/1.11) cd .. && mkdir blockchain
➜  web3 cd blockchain
➜  blockchain touch genesis.json
➜  blockchain
```

Config for genesis.json

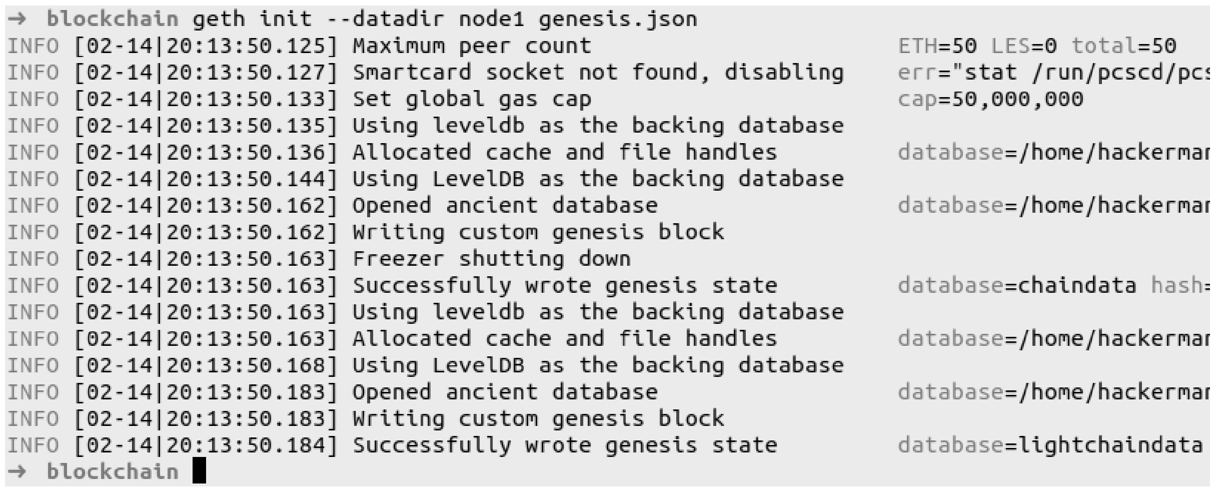
```
{
  "config": {
      "chainId": 6969,
      "homesteadBlock": 0,
      "eip150Block": 0,
      "eip155Block": 0,
      "eip158Block": 0,
      "byzantiumBlock": 0,
      "constantinopleBlock": 0,
      "petersburgBlock": 0,
      "ethash": {}
  },
  "difficulty": "4",
  "gasLimit": "8000000",
  "alloc": {}
}
```

Start Database (The Bootnode)
The bootnode is the first node created when the blockchain is started. To start the database to first node execute:

```
geth init --datadir node1 genesis.json
```

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

The terminal result must be something like this:



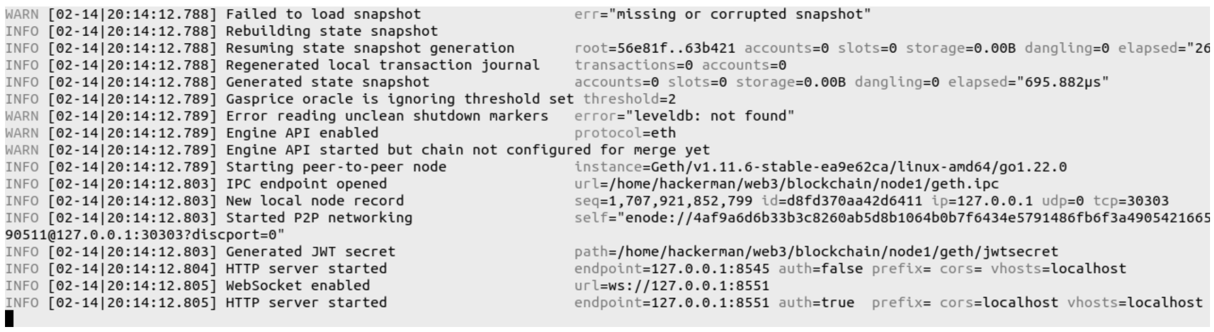
```
➜ blockchain geth init --datadir node1 genesis.json
INFO [02-14|20:13:50.125] Maximum peer count                       ETH=50 LES=0 total=50
INFO [02-14|20:13:50.127] Smartcard socket not found, disabling    err="stat /run/pcscd/pcs
INFO [02-14|20:13:50.133] Set global gas cap                       cap=50,000,000
INFO [02-14|20:13:50.135] Using leveldb as the backing database
INFO [02-14|20:13:50.136] Allocated cache and file handles         database=/home/hackerma
INFO [02-14|20:13:50.144] Using LevelDB as the backing database
INFO [02-14|20:13:50.162] Opened ancient database                  database=/home/hackerma
INFO [02-14|20:13:50.162] Writing custom genesis block
INFO [02-14|20:13:50.163] Freezer shutting down
INFO [02-14|20:13:50.163] Successfully wrote genesis state         database=chaindata hash=
INFO [02-14|20:13:50.163] Using leveldb as the backing database
INFO [02-14|20:13:50.163] Allocated cache and file handles         database=/home/hackerma
INFO [02-14|20:13:50.168] Using LevelDB as the backing database
INFO [02-14|20:13:50.183] Opened ancient database                  database=/home/hackerma
INFO [02-14|20:13:50.183] Writing custom genesis block
INFO [02-14|20:13:50.184] Successfully wrote genesis state         database=lightchaindata
➜ blockchain █
```

Start Node

```
geth --datadir node1 --networkid 6969 --http
--allow-insecure-unlock --nodiscover
--rpc.enabledeprecatedpersonal --http.corsdomain "*"
```

- 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:



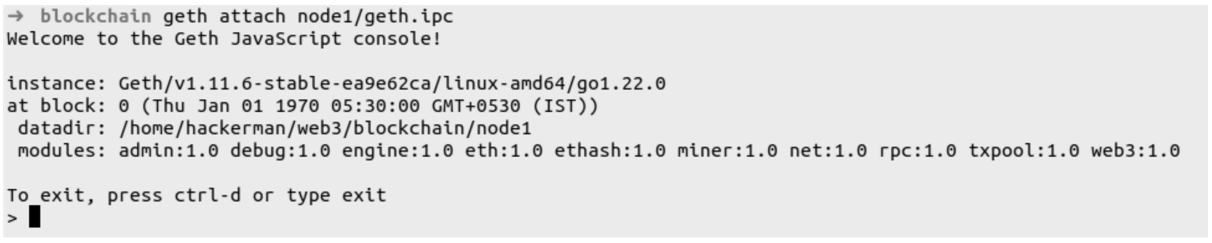
```
WARN [02-14|20:14:12.788] Failed to load snapshot                  err="missing or corrupted snapshot"
INFO [02-14|20:14:12.788] Rebuilding state snapshot
INFO [02-14|20:14:12.788] Resuming state snapshot generation       root=56e81f..63b421 accounts=0 slots=0 storage=0.00B dangling=0 elapsed="26
INFO [02-14|20:14:12.788] Regenerated local transaction journal    transactions=0 accounts=0
INFO [02-14|20:14:12.788] Generated state snapshot                 accounts=0 slots=0 storage=0.00B dangling=0 elapsed="695.882μs"
INFO [02-14|20:14:12.789] Gasprice oracle is ignoring threshold set threshold=2
WARN [02-14|20:14:12.789] Error reading unclean shutdown markers   error="leveldb: not found"
WARN [02-14|20:14:12.789] Engine API enabled                       protocol=eth
WARN [02-14|20:14:12.789] Engine API started but chain not configured for merge yet
INFO [02-14|20:14:12.789] Starting peer-to-peer node               instance=Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0
INFO [02-14|20:14:12.803] IPC endpoint opened                      url=/home/hackerman/web3/blockchain/node1/geth.ipc
INFO [02-14|20:14:12.803] New local node record                    seq=1,707,921,852,799 id=d8fd370aa42d6411 ip=127.0.0.1 udp=0 tcp=30303
INFO [02-14|20:14:12.803] Started P2P networking                   self="enode://4af9a6d6b33b3c8260ab5d8b1064b0b7f6434e5791486fb6f3a4905421665
90511@127.0.0.1:30303?discport=0"
INFO [02-14|20:14:12.803] Generated JWT secret                     path=/home/hackerman/web3/blockchain/node1/geth/jwtsecret
INFO [02-14|20:14:12.804] HTTP server started                      endpoint=127.0.0.1:8545 auth=false prefix= cors= vhosts=localhost
INFO [02-14|20:14:12.805] WebSocket enabled                        url=ws://127.0.0.1:8551
INFO [02-14|20:14:12.805] HTTP server started                      endpoint=127.0.0.1:8551 auth=true  prefix= cors=localhost vhosts=localhost
█
```

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

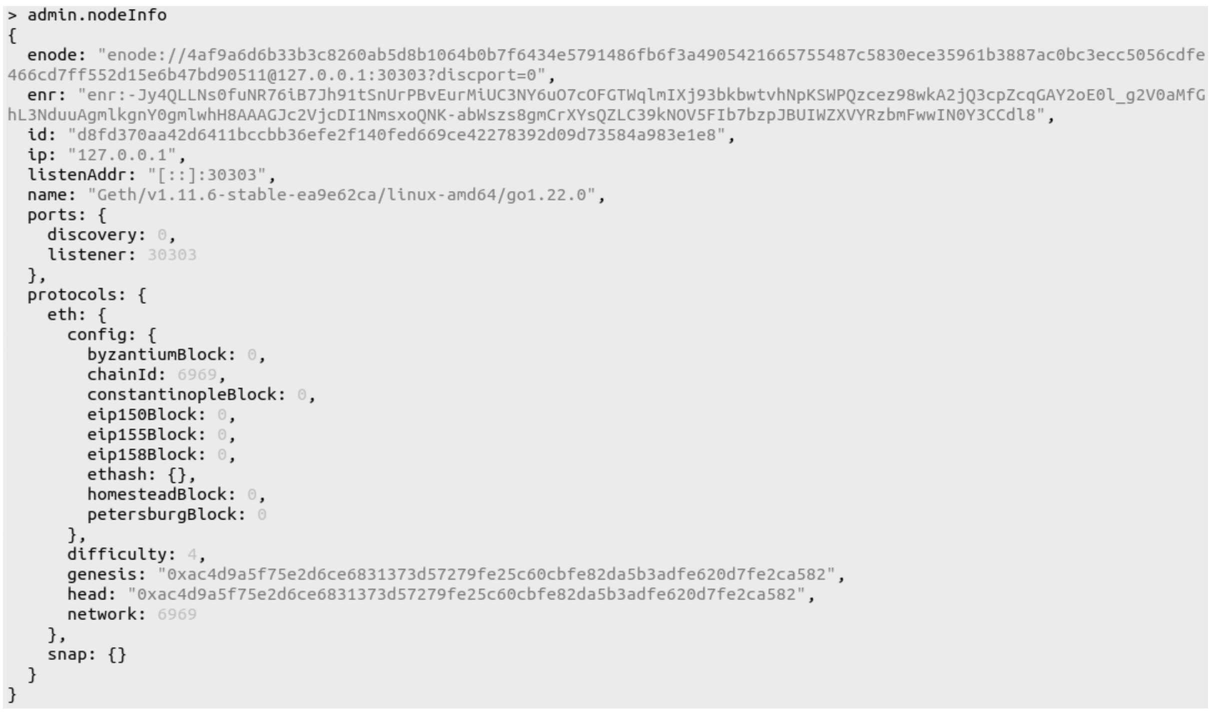
Let's go do some tests to verify this. Open another terminal window in the same folder *my-blockchain* and execute the following command:

```
geth attach node1/geth.ipc
```

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



```
→ blockchain geth attach node1/geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0
at block: 0 (Thu Jan 01 1970 05:30:00 GMT+0530 (IST))
 datadir: /home/hackerman/web3/blockchain/node1
 modules: admin:1.0 debug:1.0 engine:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> 
```

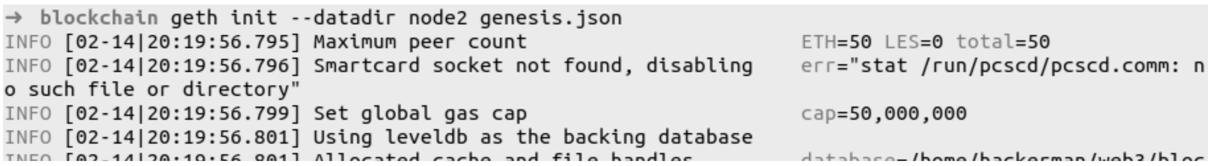


```
> admin.nodeInfo
{
  enode: "enode://4af9a6d6b33b3c8260ab5d8b1064b0b7f6434e5791486fb6f3a4905421665755487c5830ece35961b3887ac0bc3ecc5056cdfe
466cd7ff552d15e6b47bd90511@127.0.0.1:30303?discport=0",
  enr: "enr:-Jy4QLLNs0fuNR76iB7Jh91tSnUrPBvEurMiUC3NY6uO7cOFGTWqlmIXj93bkbwtvhNpKSWPQzcez98wkA2jQ3cpZcqGAY2oE0l_g2V0aMfG
hL3NduuAgmlkgnY0gmlwhH8AAAGJc2VjcDI1NmsxoQNK-abWszs8gmCrXYsQZLC39kNOV5FIb7bzpJBUIWZXVYRzbmFwwIN0Y3CCdl8",
  id: "d8fd370aa42d6411bccbb36efe2f140fed669ce42278392d09d73584a983e1e8",
  ip: "127.0.0.1",
  listenAddr: "[::]:30303",
  name: "Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0",
  ports: {
    discovery: 0,
    listener: 30303
  },
  protocols: {
    eth: {
      config: {
        byzantiumBlock: 0,
        chainId: 6969,
        constantinopleBlock: 0,
        eip150Block: 0,
        eip155Block: 0,
        eip158Block: 0,
        ethash: {},
        homesteadBlock: 0,
        petersburgBlock: 0
      },
      difficulty: 4,
      genesis: "0xac4d9a5f75e2d6ce6831373d57279fe25c60cbfe82da5b3adfe620d7fe2ca582",
      head: "0xac4d9a5f75e2d6ce6831373d57279fe25c60cbfe82da5b3adfe620d7fe2ca582",
      network: 6969
    },
    snap: {}
  }
}
```

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:

```
geth init --datadir node2 genesis.json
```



```
→ blockchain geth init --datadir node2 genesis.json
INFO [02-14|20:19:56.795] Maximum peer count                       ETH=50 LES=0 total=50
INFO [02-14|20:19:56.796] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: n
o such file or directory"
INFO [02-14|20:19:56.799] Set global gas cap                       cap=50,000,000
INFO [02-14|20:19:56.801] Using leveldb as the backing database
```

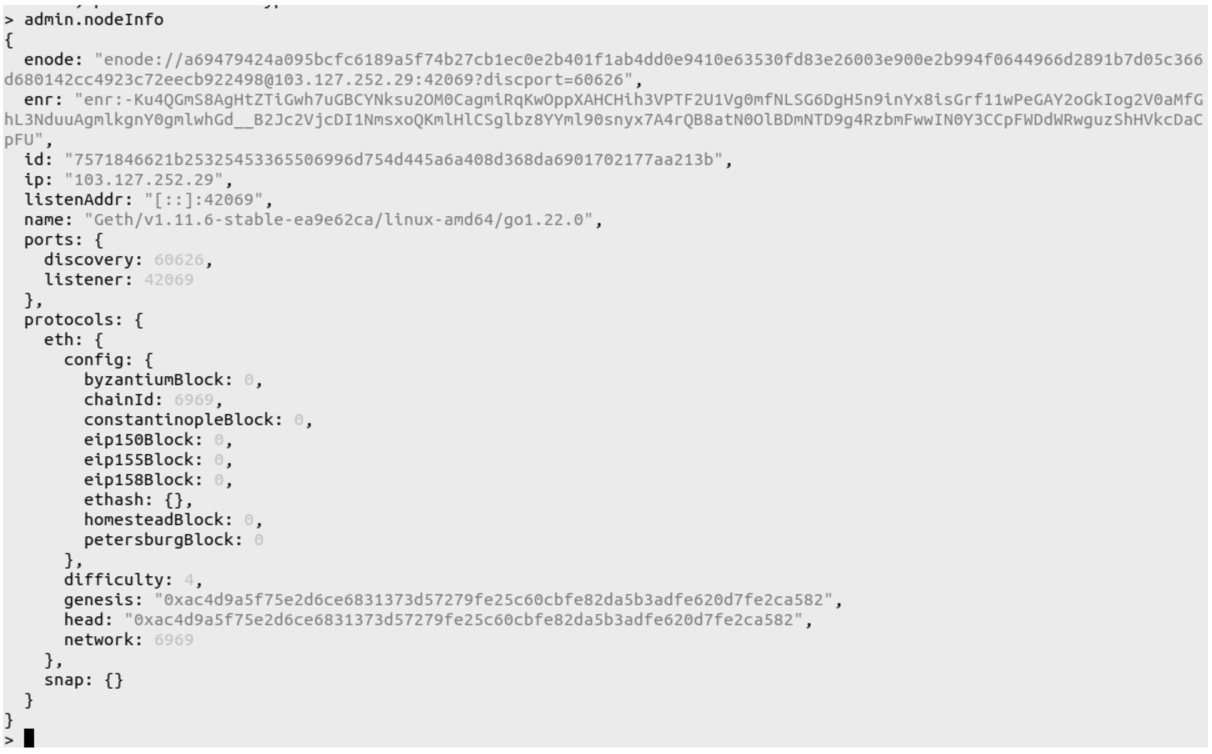
```
geth --datadir node2 --networkid 6969 --port 42069 --authrpc.port
8552 --rpc.enabledeprecatedpersonal --http.corsdomain "*"
```

The commands are very similar to the previous ones with some minor differences:

- The folder to database in this case is *node2*.
- The flags *--http* and *--allow-insecure-unlock* are not necessary.
- The flag *--port* it's necessary to differentiate this node from the previous one. The first node is running on default port **30303**.

Open a new javascript terminal to second node created and verify the informations executing:

```
geth attach node2/geth.ipc
```



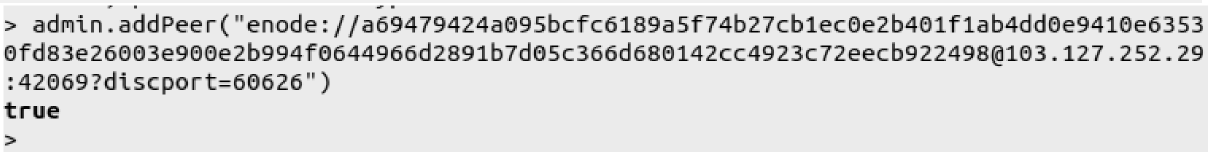
```
> admin.nodeInfo
{
  enode: "enode://a69479424a095bcfc6189a5f74b27cb1ec0e2b401f1ab4dd0e9410e63530fd83e26003e900e2b994f0644966d2891b7d05c366
d680142cc4923c72eecb922498@103.127.252.29:42069?discport=60626",
  enr: "enr:-Ku4QGmS8AgHtZTiGwh7uGBCYNksu2OM0CagmiRqKwOppXAHCHih3VPTF2U1Vg0mfNLSG6DgH5n9inYx8isGrf11wPeGAY2oGkIog2V0aMfG
hL3NduuAgmlkgnY0gmlwhGd__B2Jc2VjcDI1NmsxoQKmlHlCSglbz8YYml90snyx7A4rQB8atN0OlBDmNTD9g4RzbmFwwIN0Y3CCpFWDdWRwguzShHVkcDaC
pFU",
  id: "7571846621b25325453365506996d754d445a6a408d368da6901702177aa213b",
  ip: "103.127.252.29",
  listenAddr: "[::]:42069",
  name: "Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0",
  ports: {
    discovery: 60626,
    listener: 42069
  },
  protocols: {
    eth: {
      config: {
        byzantiumBlock: 0,
        chainId: 6969,
        constantinopleBlock: 0,
        eip150Block: 0,
        eip155Block: 0,
        eip158Block: 0,
        ethash: {},
        homesteadBlock: 0,
        petersburgBlock: 0
      },
      difficulty: 4,
      genesis: "0xac4d9a5f75e2d6ce6831373d57279fe25c60cbfe82da5b3adfe620d7fe2ca582",
      head: "0xac4d9a5f75e2d6ce6831373d57279fe25c60cbfe82da5b3adfe620d7fe2ca582",
      network: 6969
    },
    snap: {}
  }
}
>
```

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

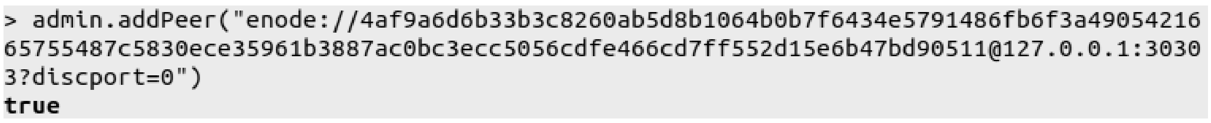
```
admin.addPeer("<enode value of second")
```



```
> admin.addPeer("enode://a69479424a095bcfc6189a5f74b27cb1ec0e2b401f1ab4dd0e9410e6353
0fd83e26003e900e2b994f0644966d2891b7d05c366d680142cc4923c72eecb922498@103.127.252.29
:42069?discport=60626")
true
>
```

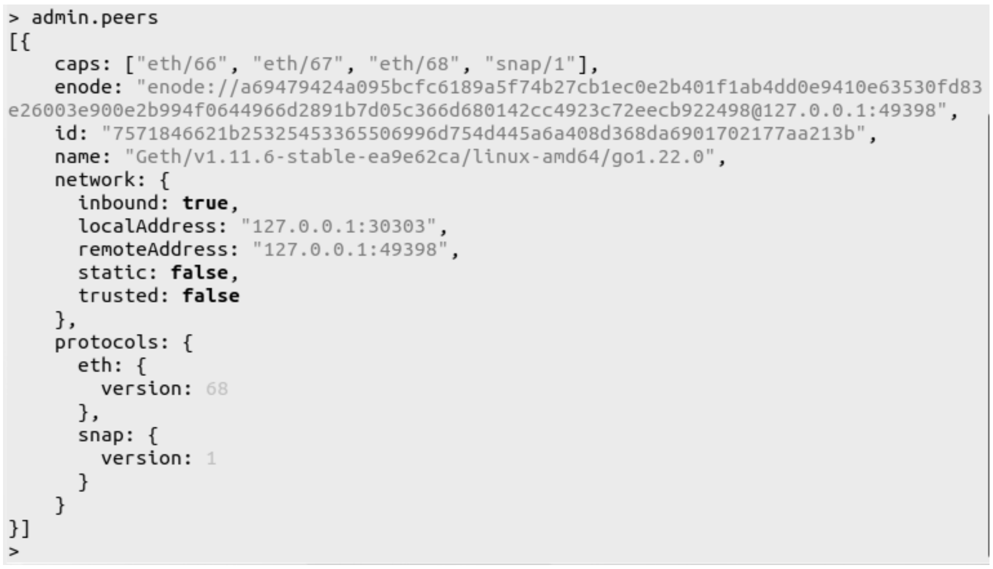
Copy the **enode** from the *nodeInfo* of the **first** node, then execute the following command in javascript terminal of the **second** node

```
admin.addPeer("<enode value of first")
```



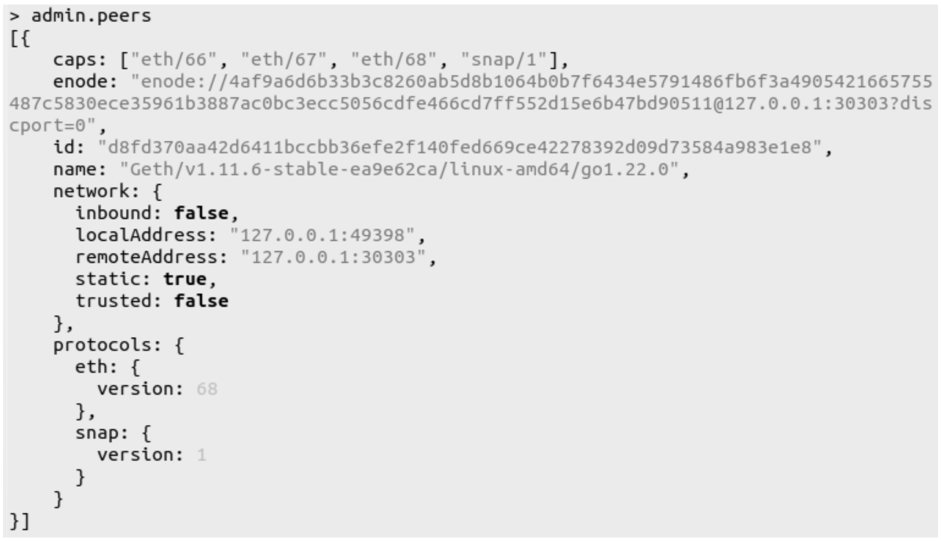
```
> admin.addPeer("enode://4af9a6d6b33b3c8260ab5d8b1064b0b7f6434e5791486fb6f3a49054216
65755487c5830ece35961b3887ac0bc3ecc5056cdfe466cd7ff552d15e6b47bd90511@127.0.0.1:3030
3?discport=0")
true
```

Node 1:



```
> admin.peers
[{
    caps: ["eth/66", "eth/67", "eth/68", "snap/1"],
    enode: "enode://a69479424a095bcfc6189a5f74b27cb1ec0e2b401f1ab4dd0e9410e63530fd83
e26003e900e2b994f0644966d2891b7d05c366d680142cc4923c72eecb922498@127.0.0.1:49398",
    id: "7571846621b25325453365506996d754d445a6a408d368da6901702177aa213b",
    name: "Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0",
    network: {
      inbound: true,
      localAddress: "127.0.0.1:30303",
      remoteAddress: "127.0.0.1:49398",
      static: false,
      trusted: false
    },
    protocols: {
      eth: {
        version: 68
      },
      snap: {
        version: 1
      }
    }
}]
>
```

Node: 2



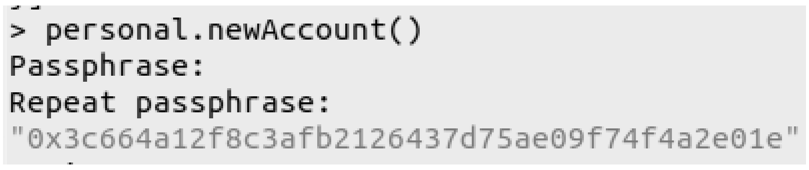
```
> admin.peers
[{
    caps: ["eth/66", "eth/67", "eth/68", "snap/1"],
    enode: "enode://4af9a6d6b33b3c8260ab5d8b1064b0b7f6434e5791486fb6f3a4905421665755
487c5830ece35961b3887ac0bc3ecc5056cdfe466cd7ff552d15e6b47bd90511@127.0.0.1:30303?dis
cport=0",
    id: "d8fd370aa42d6411bccbb36efe2f140fed669ce42278392d09d73584a983e1e8",
    name: "Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0",
    network: {
      inbound: false,
      localAddress: "127.0.0.1:49398",
      remoteAddress: "127.0.0.1:30303",
      static: true,
      trusted: false
    },
    protocols: {
      eth: {
        version: 68
      },
      snap: {
        version: 1
      }
    }
}]
```

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.

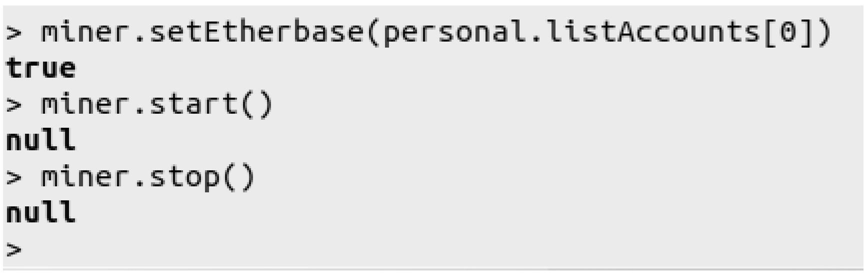
Mining

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.

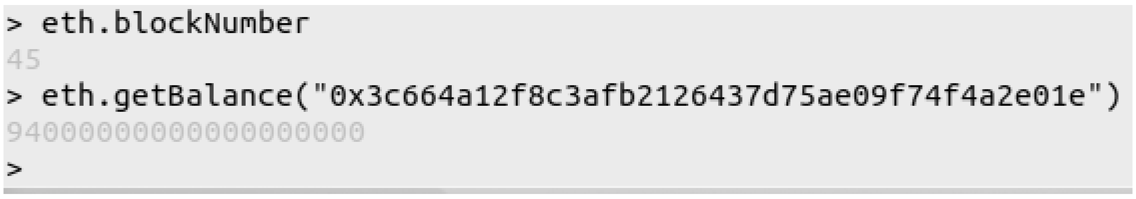
```
personal.newAccount()
miner.setEtherbase(personal.listAccounts[0])
miner.start()
# wait for 1-2 mins before stopping(when eth.blockNumber is not 0)
miner.stop()
eth.blockNumber
eth.getBalance("<address from the above newAccount command>")
```



```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0x3c664a12f8c3afb2126437d75ae09f74f4a2e01e"
```



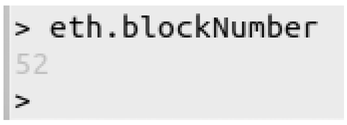
```
> miner.setEtherbase(personal.listAccounts[0])
true
> miner.start()
null
> miner.stop()
null
>
```



```
> eth.blockNumber
45
> eth.getBalance("0x3c664a12f8c3afb2126437d75ae09f74f4a2e01e")
940000000000000000000
>
```

You can verify the **blockNumber** also in javascript terminal of the **second** node after some seconds, the value must be the same presented in the first node. This means that in fact the nodes are connected and are updated.

Terminal of Node 2

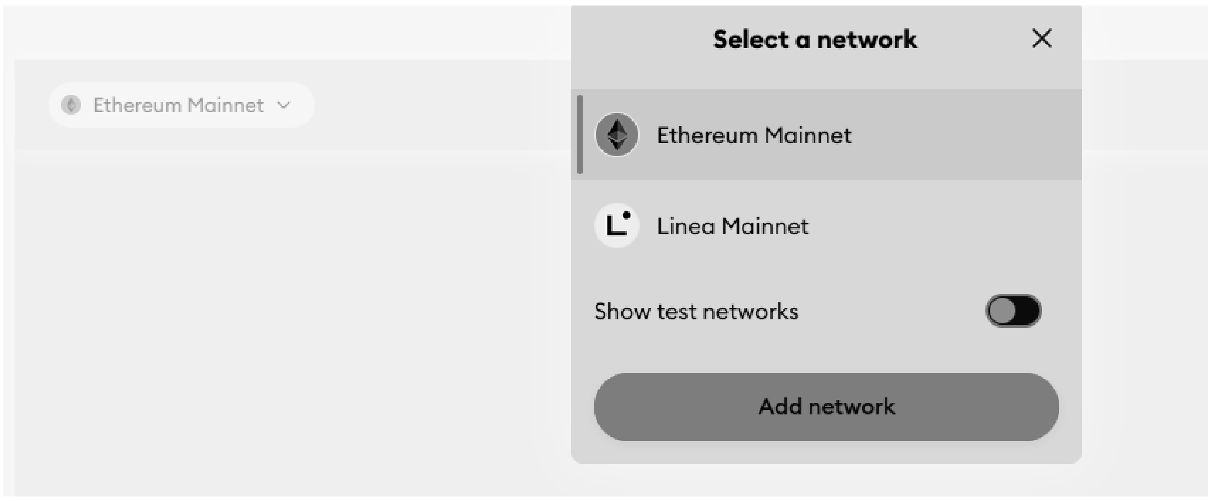


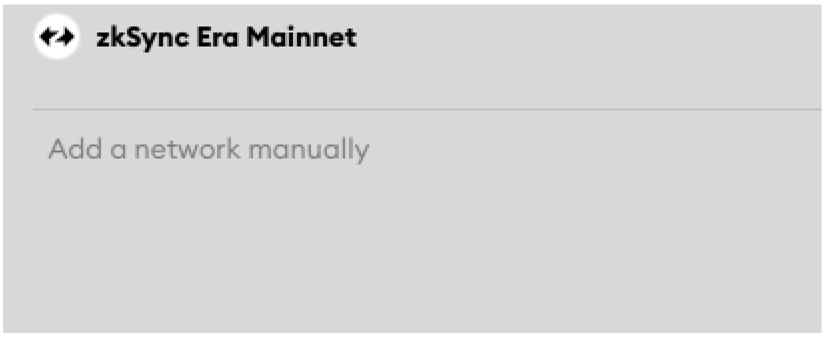
```
> eth.blockNumber
52
>
```

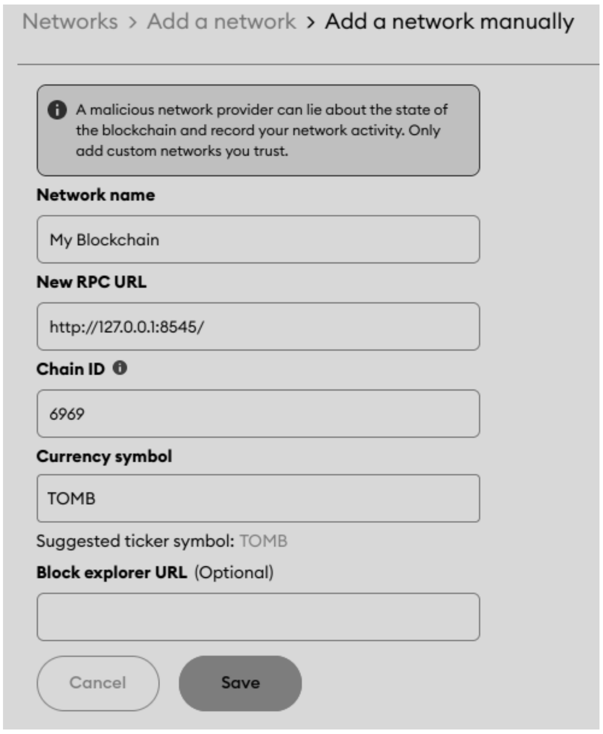
Install metamask and create a new account

Add My Blockchain Network

Let's add our Blockchain Network to MetaMask. Click on *Ethereum Mainnet* and select *Add New Network* option.







Networks > Add a network > Add a network manually

A malicious network provider can lie about the state of the blockchain and record your network activity. Only add custom networks you trust.

Network name: My Blockchain

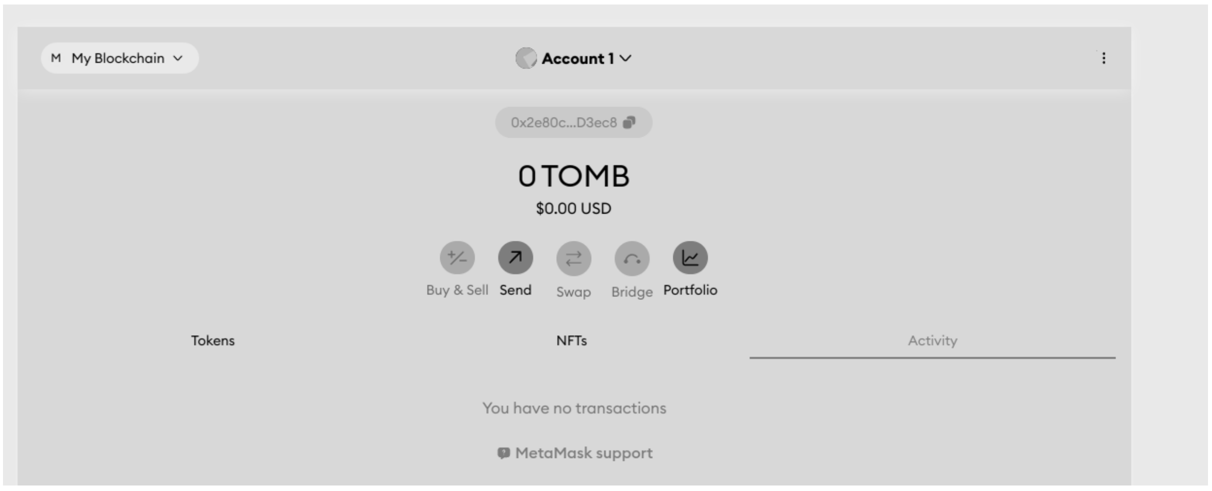
New RPC URL: http://127.0.0.1:8545/

Chain ID: 6969

Currency symbol: TOMB

Suggested ticker symbol: TOMB

Block explorer URL (Optional):

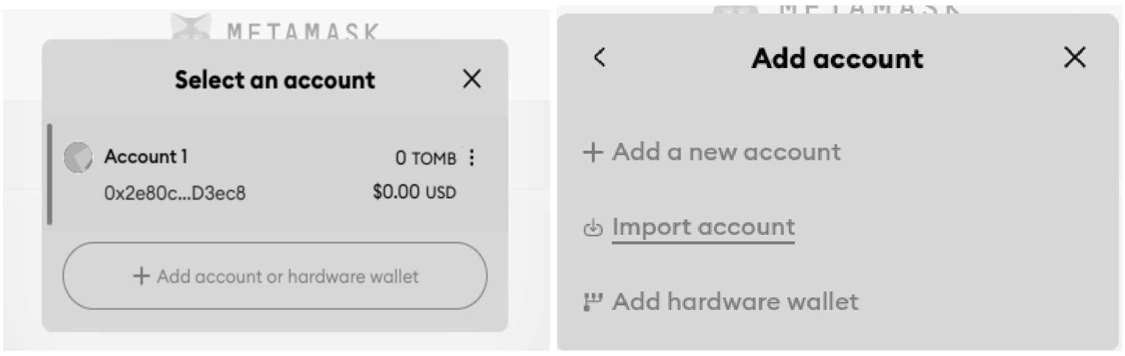


Importing accounts using json file

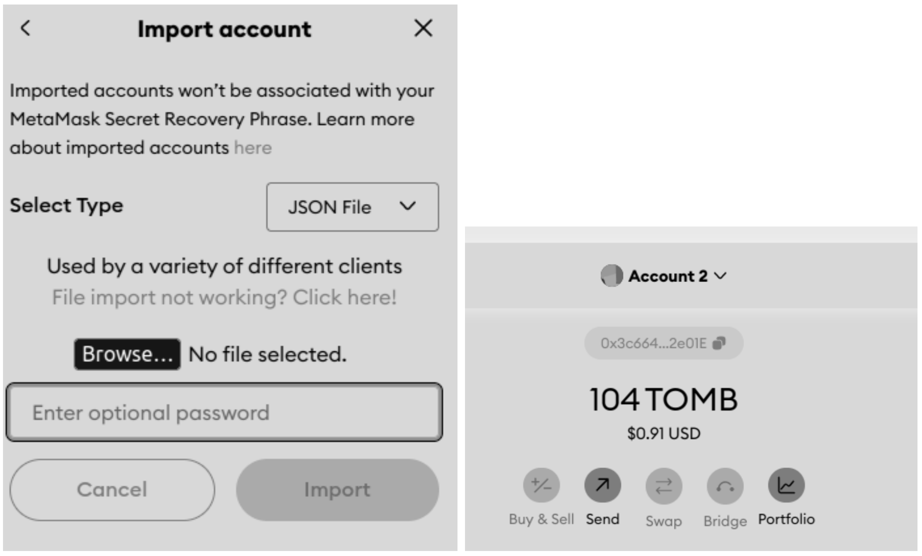
The *account json file* can be found in database of the *Node1* in your blockchain, the path to this file contains the account address, something like this:

```
node1/keystore/UTC--2022-03-17T20-40-36.929865867Z--bd3156b239e2bb
8d073406e67eba59a651be18f0
```

Click on Account > Import account



Select json file (Takes upto 1-2 mins)


Import account
Imported accounts won't be associated with your MetaMask Secret Recovery Phrase. Learn more about imported accounts here
Select Type
JSON File
Used by a variety of different clients
File import not working? Click here!
Browse...
No file selected.
Enter optional password
Cancel
Import
Account 2
0x3c664...2e01E
104 TOMB
$0.91 USD
Buy & Sell
Send
Swap
Bridge
Portfolio

Transfer Funds

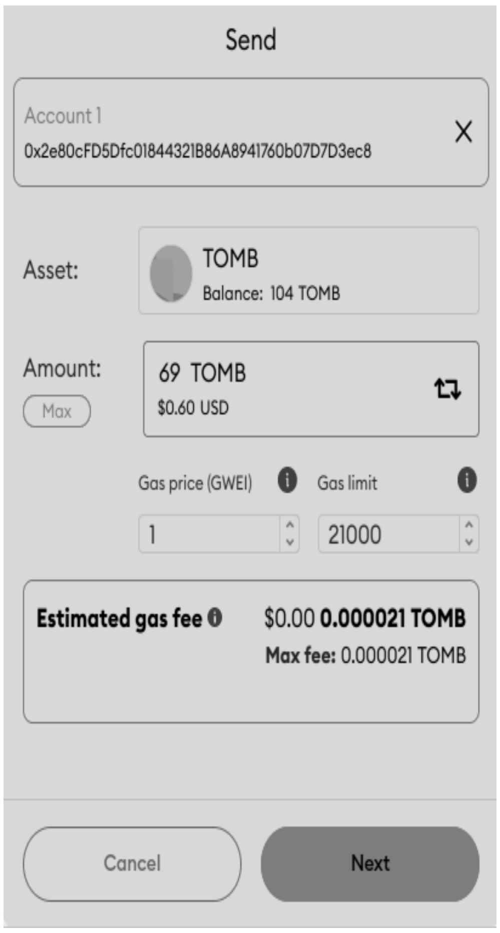
Let's transfer some value from *Account 2* to *Account 1* to validate the structure and finish this post.

First, go to *Account 1* in MetaMask and copy the *Account Public Address* and back to *Account 2*. In my case the public address of the *Account 1* is:

0x2e80cFD5Dfc01844321B86A8941760b07D7D3ec8.

In *Account 2* click on *"Send"* button, paste the public address of the *Account 1* and type the value to transfer. Click on *"Next"* button and then click on *"Confirm"* button.

The transaction will be with status *"Pending"*, waiting for a block to be mined for validation.



Send

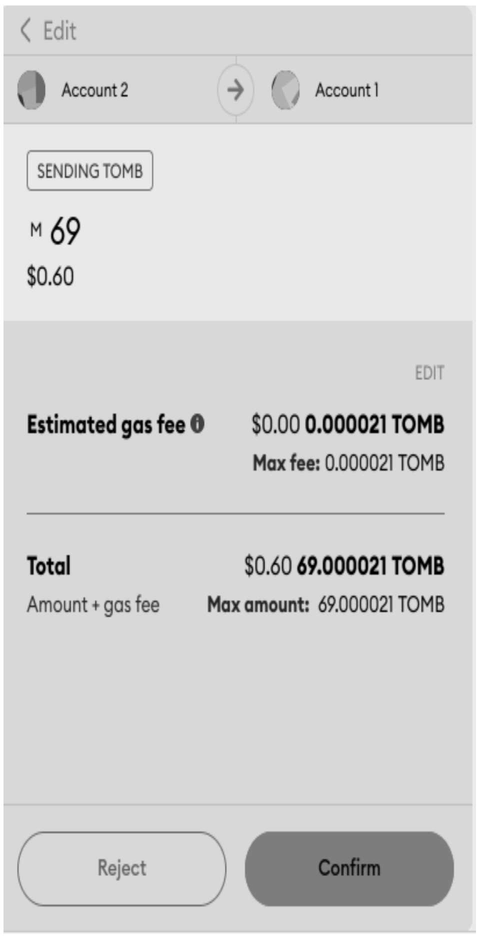
Account 1
0x2e80cFD5Dfc01844321B86A8941760b07D7D3ec8

Asset: TOMB
Balance: 104 TOMB

Amount: 69 TOMB
$0.60 USD

Gas price (GWEI): 1
Gas limit: 21000

Estimated gas fee $0.00 **0.000021 TOMB**
Max fee: 0.000021 TOMB



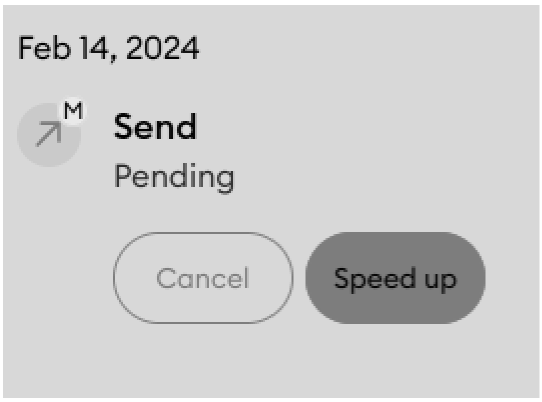
Account 2 → Account 1

SENDING TOMB

M 69
$0.60

Estimated gas fee $0.00 **0.000021 TOMB**
Max fee: 0.000021 TOMB

Total $0.60 **69.000021 TOMB**
Amount + gas fee **Max amount:** 69.000021 TOMB



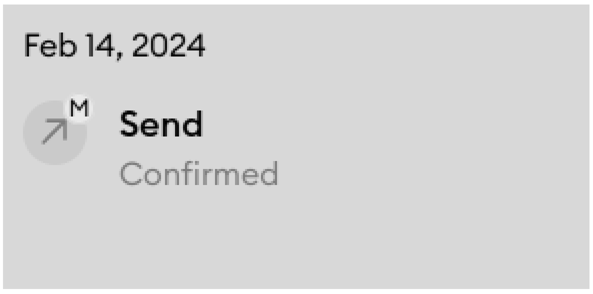
Feb 14, 2024

Send
Pending

Now, it's necessary to start mining to validate that transaction. Let's do this in our blockchain using Geth in Javascript Console from *Node1*. Type the command:

```
miner.start()
```

Wait for some seconds and verify your account in MetaMask again.



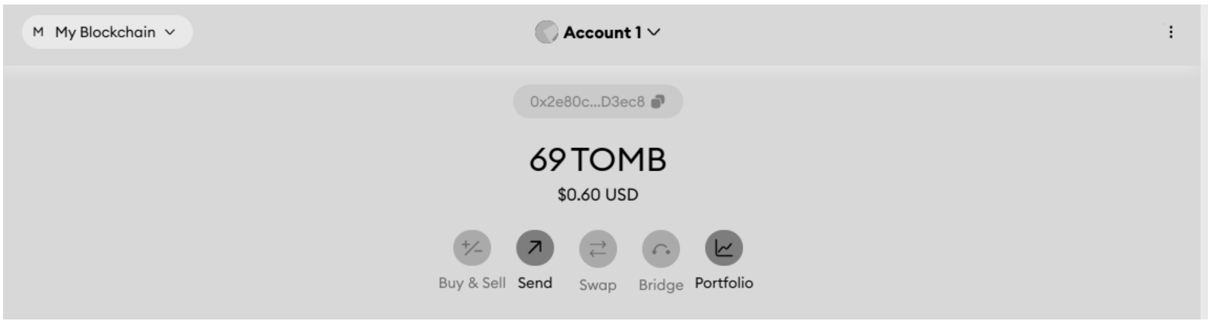
Feb 14, 2024

Send

Confirmed

Obs: If you want, you can stop the mining in *Node1* after the transaction validation. Type command to stop the mining:

```
miner.stop()
```



69 TOMB

$0.60 USD

The transaction was validated and the *Account 1* received the **69 TOMB**.

Practical 4

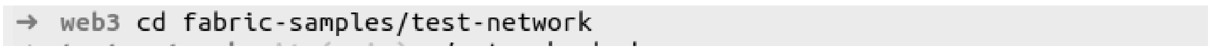
Aim: Deploy an asset-transfer app using block chain. Learn app development within a Hyperledger Fabric network.

Installing dependencies

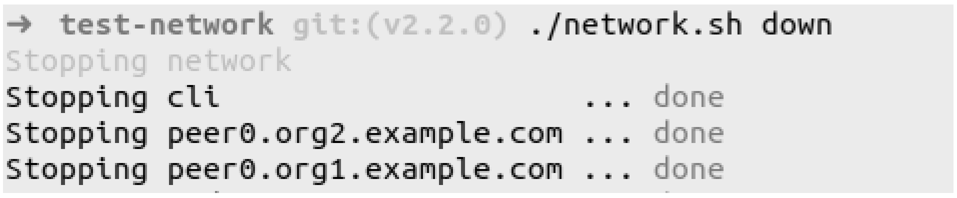
```
sudo apt install jq
```

Start the network and deploy the smart contract

```
cd fabric-samples/test-network
./network.sh down
```

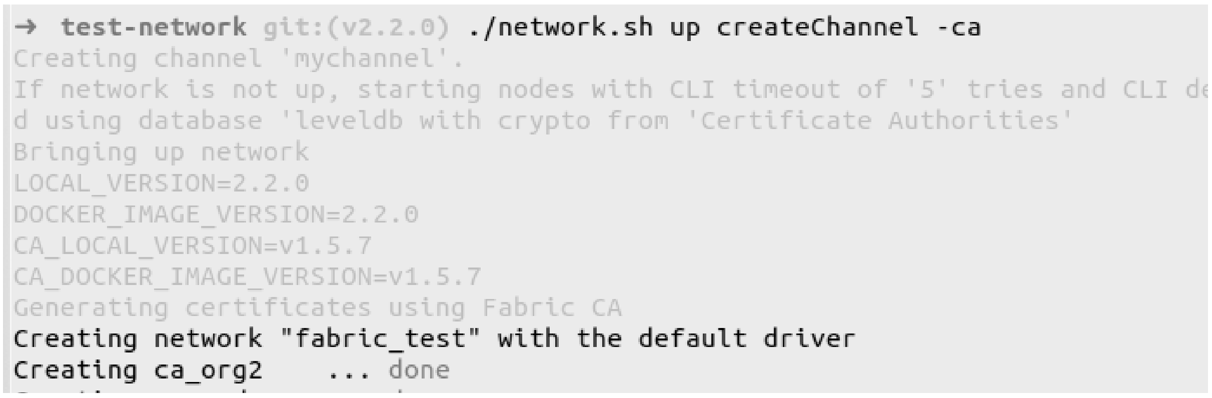


```
→ web3 cd fabric-samples/test-network
```



```
→ test-network git:(v2.2.0) ./network.sh down
Stopping network
Stopping cli                    ... done
Stopping peer0.org2.example.com ... done
Stopping peer0.org1.example.com ... done
```

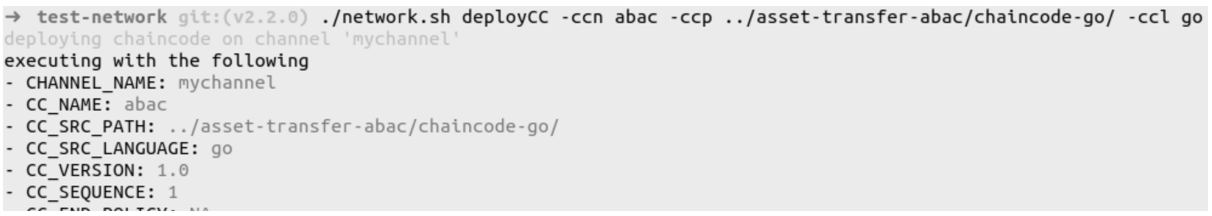
```
./network.sh up createChannel -ca
```



```
→ test-network git:(v2.2.0) ./network.sh up createChannel -ca
Creating channel 'mychannel'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI de
d using database 'leveldb with crypto from 'Certificate Authorities'
Bringing up network
LOCAL_VERSION=2.2.0
DOCKER_IMAGE_VERSION=2.2.0
CA_LOCAL_VERSION=v1.5.7
CA_DOCKER_IMAGE_VERSION=v1.5.7
Generating certificates using Fabric CA
Creating network "fabric_test" with the default driver
Creating ca_org2    ... done
```

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

```
./network.sh deployCC -ccn abac -ccp ../asset-transfer-abac/chaincode-go/ -ccl go
```



```
→ test-network git:(v2.2.0) ./network.sh deployCC -ccn abac -ccp ../asset-transfer-abac/chaincode-go/ -ccl go
deploying chaincode on channel 'mychannel'
executing with the following
- CHANNEL_NAME: mychannel
- CC_NAME: abac
- CC_SRC_PATH: ../asset-transfer-abac/chaincode-go/
- CC_SRC_LANGUAGE: go
- CC_VERSION: 1.0
- CC_SEQUENCE: 1
```

Register identities with attributes

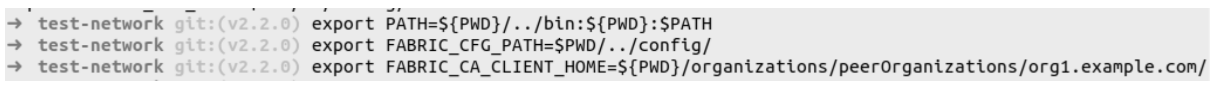
We can use the one of the test network Certificate Authorities to register and enroll identities with the attribute of **abac.creator=true**. First, we need to set the following environment variables in order to use the Fabric CA client.

```
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:

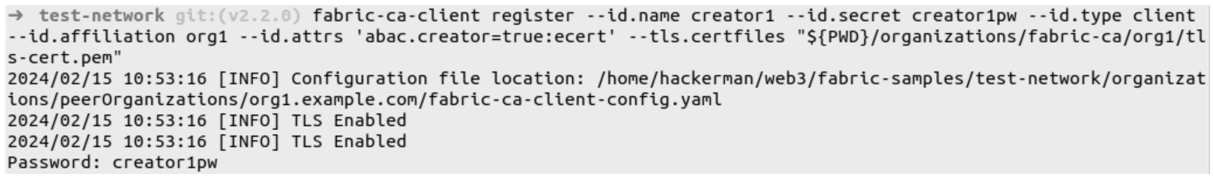
```
export
FABRIC_CA_CLIENT_HOME=${PWD}/organizations/peerOrganizations/org1.
example.com/
```

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**.



```
→ test-network git:(v2.2.0) export PATH=${PWD}/../bin:${PWD}:$PATH
→ test-network git:(v2.2.0) export FABRIC_CFG_PATH=$PWD/../config/
→ test-network git:(v2.2.0) export FABRIC_CA_CLIENT_HOME=${PWD}/organizations/peerOrganizations/org1.example.com/
```

```
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"
```

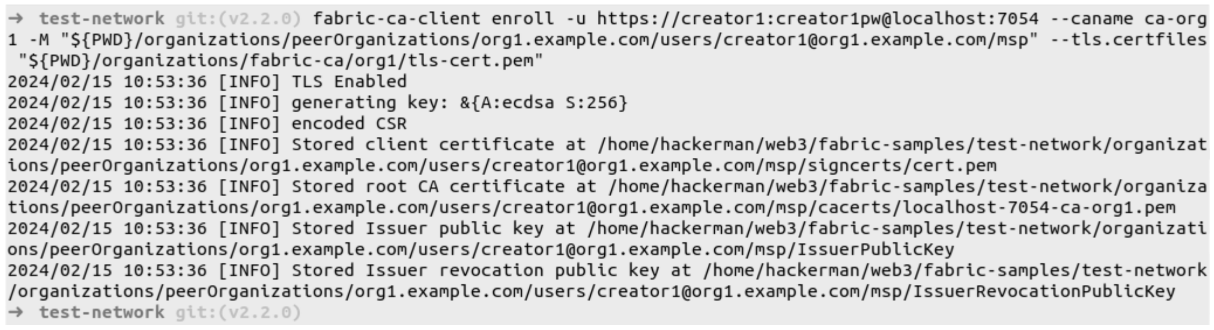


```
→ test-network git:(v2.2.0) fabric-ca-client register --id.name creator1 --id.secret creator1pw --id.type client
--id.affiliation org1 --id.attrs 'abac.creator=true:ecert' --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tl
s-cert.pem"
2024/02/15 10:53:16 [INFO] Configuration file location: /home/hackerman/web3/fabric-samples/test-network/organizat
ions/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2024/02/15 10:53:16 [INFO] TLS Enabled
2024/02/15 10:53:16 [INFO] TLS Enabled
Password: creator1pw
```

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.

```
fabric-ca-client enroll -u
https://creator1:creator1pw@localhost:7054 --caname ca-org1 -M
"${PWD}/organizations/peerOrganizations/org1.example.com/users/cre
```

```
ator1@org1.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```



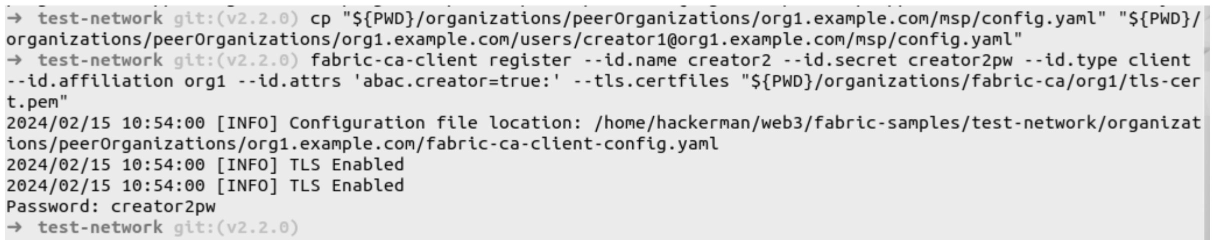
```
➜ test-network git:(v2.2.0) fabric-ca-client enroll -u https://creator1:creator1pw@localhost:7054 --caname ca-org
1 -M "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp" --tls.certfiles
 "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
2024/02/15 10:53:36 [INFO] TLS Enabled
2024/02/15 10:53:36 [INFO] generating key: &{A:ecdsa S:256}
2024/02/15 10:53:36 [INFO] encoded CSR
2024/02/15 10:53:36 [INFO] Stored client certificate at /home/hackerman/web3/fabric-samples/test-network/organizat
ions/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/signcerts/cert.pem
2024/02/15 10:53:36 [INFO] Stored root CA certificate at /home/hackerman/web3/fabric-samples/test-network/organiza
tions/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/cacerts/localhost-7054-ca-org1.pem
2024/02/15 10:53:36 [INFO] Stored Issuer public key at /home/hackerman/web3/fabric-samples/test-network/organizati
ons/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/IssuerPublicKey
2024/02/15 10:53:36 [INFO] Stored Issuer revocation public key at /home/hackerman/web3/fabric-samples/test-network
/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/IssuerRevocationPublicKey
➜ test-network git:(v2.2.0)
```

Now that we have enrolled the identity, run the command below to copy the Node OU configuration file into the creator1 MSP folder.

```
cp
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/confi
g.yaml"
"${PWD}/organizations/peerOrganizations/org1.example.com/users/cre
ator1@org1.example.com/msp/config.yaml"
```

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.

```
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"
```

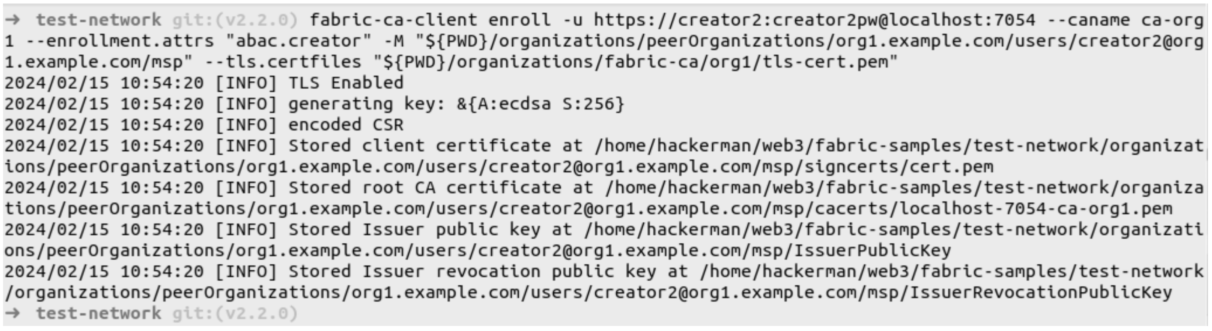


```
➜ test-network git:(v2.2.0) cp "${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.yaml" "${PWD}/
organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/config.yaml"
➜ test-network git:(v2.2.0) fabric-ca-client register --id.name creator2 --id.secret creator2pw --id.type client
--id.affiliation org1 --id.attrs 'abac.creator=true:' --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cer
t.pem"
2024/02/15 10:54:00 [INFO] Configuration file location: /home/hackerman/web3/fabric-samples/test-network/organizat
ions/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2024/02/15 10:54:00 [INFO] TLS Enabled
2024/02/15 10:54:00 [INFO] TLS Enabled
Password: creator2pw
➜ test-network git:(v2.2.0)
```

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

```
fabric-ca-client enroll -u
https://creator2:creator2pw@localhost:7054 --caname ca-org1
```

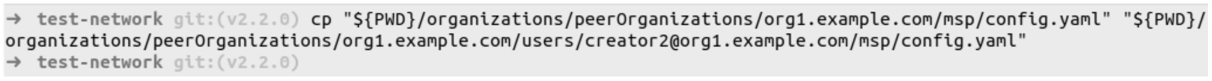
```
--enrollment.attrs "abac.creator" -M
"${PWD}/organizations/peerOrganizations/org1.example.com/users/cre
ator2@org1.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```



```
➜ test-network git:(v2.2.0) fabric-ca-client enroll -u https://creator2:creator2pw@localhost:7054 --caname ca-org
1 --enrollment.attrs "abac.creator" -M "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator2@org
1.example.com/msp" --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
2024/02/15 10:54:20 [INFO] TLS Enabled
2024/02/15 10:54:20 [INFO] generating key: &{A:ecdsa S:256}
2024/02/15 10:54:20 [INFO] encoded CSR
2024/02/15 10:54:20 [INFO] Stored client certificate at /home/hackerman/web3/fabric-samples/test-network/organizat
ions/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/signcerts/cert.pem
2024/02/15 10:54:20 [INFO] Stored root CA certificate at /home/hackerman/web3/fabric-samples/test-network/organiza
tions/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/cacerts/localhost-7054-ca-org1.pem
2024/02/15 10:54:20 [INFO] Stored Issuer public key at /home/hackerman/web3/fabric-samples/test-network/organizati
ons/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/IssuerPublicKey
2024/02/15 10:54:20 [INFO] Stored Issuer revocation public key at /home/hackerman/web3/fabric-samples/test-network
/organizations/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/IssuerRevocationPublicKey
➜ test-network git:(v2.2.0)
```

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

```
cp
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/confi
g.yaml"
"${PWD}/organizations/peerOrganizations/org1.example.com/users/cre
ator2@org1.example.com/msp/config.yaml"
```

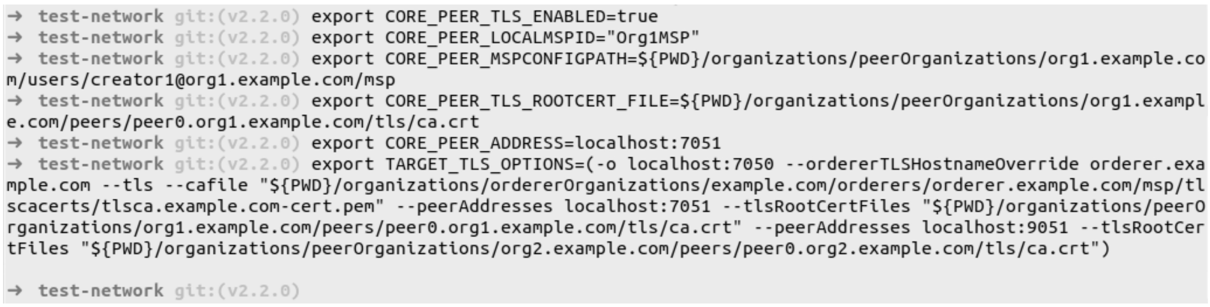


```
➜ test-network git:(v2.2.0) cp "${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.yaml" "${PWD}/
organizations/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/config.yaml"
➜ test-network git:(v2.2.0)
```

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:

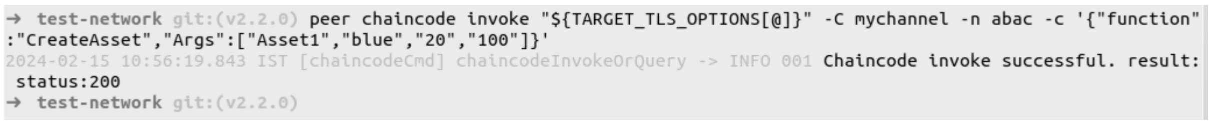
```
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.ex
ample.com/users/creator1@org1.example.com/msp
export
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org
1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_ADDRESS=localhost:7051
export TARGET_TLS_OPTIONS=(-o localhost:7050
--ordererTLSHostnameOverride orderer.example.com --tls --cafile
"${PWD}/organizations/ordererOrganizations/example.com/orderers/ordere
r.example.com/msp/tlscacerts/tlsca.example.com-cert.pem"
--peerAddresses localhost:7051 --tlsRootCertFiles
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.o
rg1.example.com/tls/ca.crt" --peerAddresses localhost:9051
--tlsRootCertFiles
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.o
rg2.example.com/tls/ca.crt")
```



```
➜ test-network git:(v2.2.0) export CORE_PEER_TLS_ENABLED=true
➜ test-network git:(v2.2.0) export CORE_PEER_LOCALMSPID="Org1MSP"
➜ test-network git:(v2.2.0) export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.co
m/users/creator1@org1.example.com/msp
➜ test-network git:(v2.2.0) export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.exampl
e.com/peers/peer0.org1.example.com/tls/ca.crt
➜ test-network git:(v2.2.0) export CORE_PEER_ADDRESS=localhost:7051
➜ test-network git:(v2.2.0) export TARGET_TLS_OPTIONS=(-o localhost:7050 --ordererTLSHostnameOverride orderer.exa
mple.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tl
scacerts/tlsca.example.com-cert.pem" --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerO
rganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCer
tFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt")

➜ test-network git:(v2.2.0)
```

Run the following command to create Asset1:

```
peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c
'{"function":"CreateAsset","Args":["Asset1","blue","20","100"]}'
```

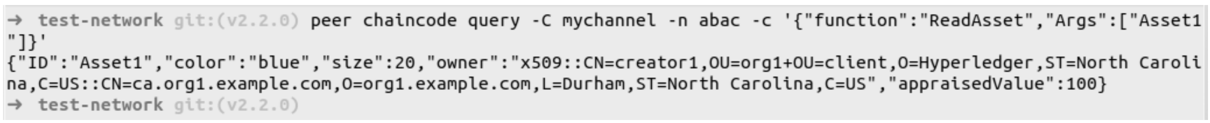


```
➜ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"CreateAsset","Args":["Asset1","blue","20","100"]}'
2024-02-15 10:56:19.843 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result:
 status:200
➜ test-network git:(v2.2.0)
```

You can use the command below to query the asset on the ledger:

```
peer chaincode query -C mychannel -n abac -c
'{"function":"ReadAsset","Args":["Asset1"]}'
```

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

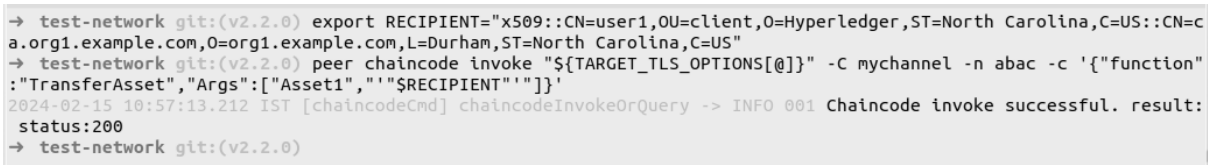


```
➜ test-network git:(v2.2.0) peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1
"]}'
{"ID":"Asset1","color":"blue","size":20,"owner":"x509::CN=creator1,OU=org1+OU=client,O=Hyperledger,ST=North Caroli
na,C=US::CN=ca.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US","appraisedValue":100}
➜ test-network git:(v2.2.0)
```

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:

```
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"

peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n
abac -c
'{"function":"TransferAsset","Args":["Asset1","'"$RECIPIENT"'"]}'
```

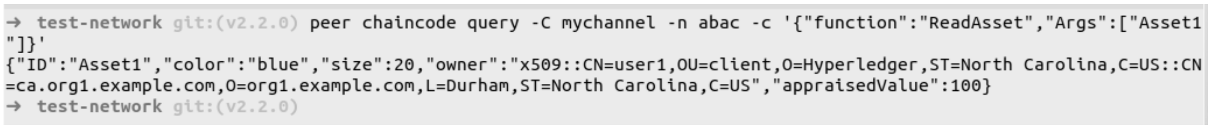


```
➜  test-network git:(v2.2.0) export RECIPIENT="x509::CN=user1,OU=client,O=Hyperledger,ST=North Carolina,C=US::CN=c
a.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US"
➜  test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"TransferAsset","Args":["Asset1","'"$RECIPIENT"'"]}'
2024-02-15 10:57:13.212 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result:
 status:200
➜  test-network git:(v2.2.0)
```

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

```
peer chaincode query -C mychannel -n abac -c
'{"function":"ReadAsset","Args":["Asset1"]}'
```

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



```
➜  test-network git:(v2.2.0) 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}
➜  test-network git:(v2.2.0)
```

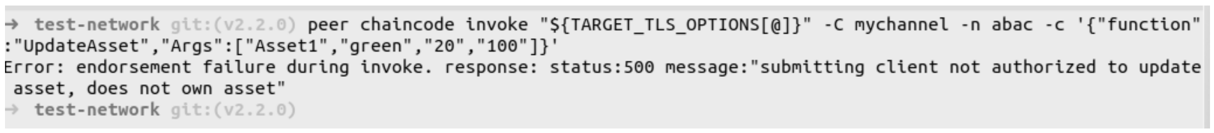
Update the asset

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:

```
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:

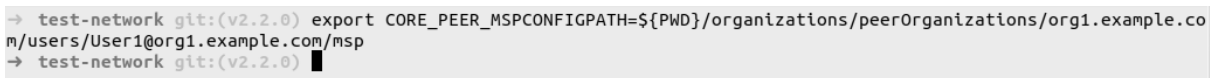
Error: endorsement failure during invoke. response: status:500 message:"submitting client not authorized to update asset, does not own asset"



```
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"UpdateAsset","Args":["Asset1","green","20","100"]}'
Error: endorsement failure during invoke. response: status:500 message:"submitting client not authorized to update
 asset, does not own asset"
→ test-network git:(v2.2.0)
```

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

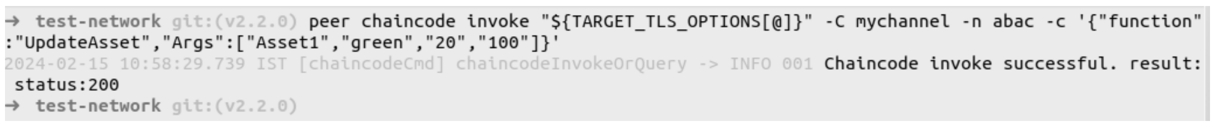
```
export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org
1.example.com/users/User1@org1.example.com/msp
```



```
→ test-network git:(v2.2.0) export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.co
m/users/User1@org1.example.com/msp
→ test-network git:(v2.2.0)
```

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.

```
peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n
abac -c
'{"function":"UpdateAsset","Args":["Asset1","green","20","100"]}'
```



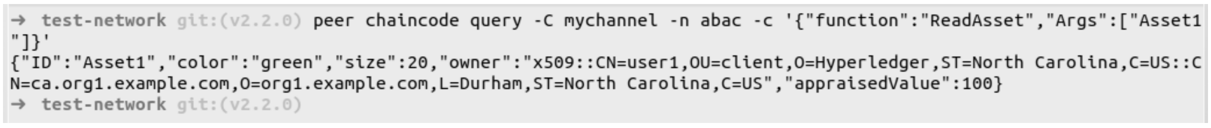
```
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"UpdateAsset","Args":["Asset1","green","20","100"]}'
2024-02-15 10:58:29.739 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result:
 status:200
→ test-network git:(v2.2.0)
```

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

```
peer chaincode query -C mychannel -n abac -c
```

```
'{"function":"ReadAsset","Args":["Asset1"]}'
```

The result will display that Asset1 is now green:

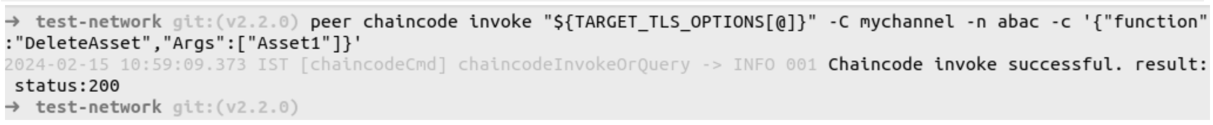


```
→ test-network git:(v2.2.0) peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1
"]}'
{"ID":"Asset1","color":"green","size":20,"owner":"x509::CN=user1,OU=client,O=Hyperledger,ST=North Carolina,C=US::C
N=ca.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US","appraisedValue":100}
→ test-network git:(v2.2.0)
```

Delete the asset

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

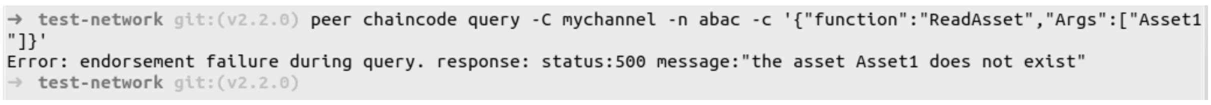
```
peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n
abac -c '{"function":"DeleteAsset","Args":["Asset1"]}'
```



```
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"DeleteAsset","Args":["Asset1"]}'
2024-02-15 10:59:09.373 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result:
 status:200
→ test-network git:(v2.2.0)
```

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

```
peer chaincode query -C mychannel -n abac -c
'{"function":"ReadAsset","Args":["Asset1"]}'
```



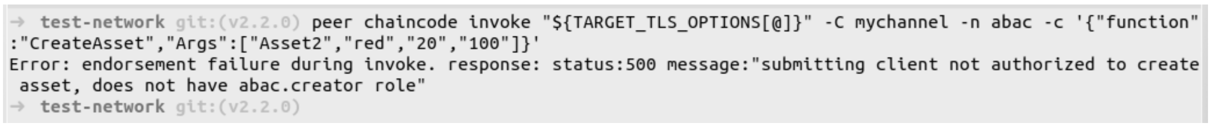
```
→ test-network git:(v2.2.0) peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1
"]}'
Error: endorsement failure during query. response: status:500 message:"the asset Asset1 does not exist"
→ test-network git:(v2.2.0)
```

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:

```
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"

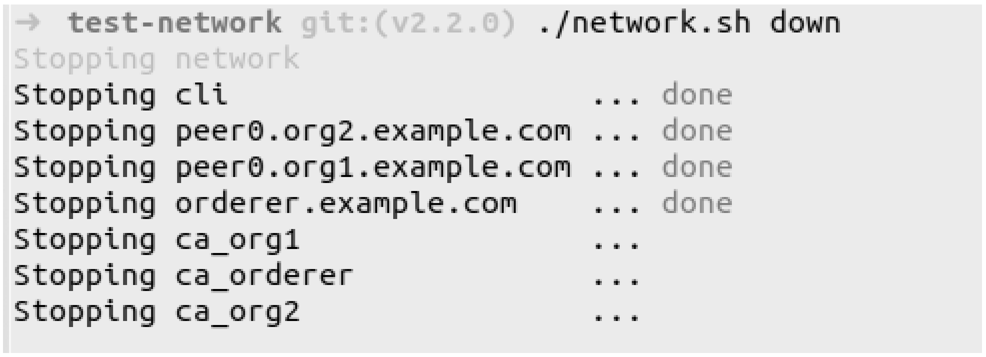


```
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"CreateAsset","Args":["Asset2","red","20","100"]}'
Error: endorsement failure during invoke. response: status:500 message:"submitting client not authorized to create
 asset, does not have abac.creator role"
→ test-network git:(v2.2.0)
```

Clean up

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

```
./network.sh down
```



```
→ test-network git:(v2.2.0) ./network.sh down
Stopping network
Stopping cli                    ... done
Stopping peer0.org2.example.com ... done
Stopping peer0.org1.example.com ... done
Stopping orderer.example.com    ... done
Stopping ca_org1                ...
Stopping ca_orderer             ...
Stopping ca_org2                ...
```

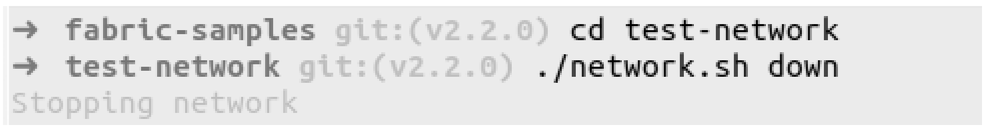
Practical 5

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

Setup the blockchain network

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

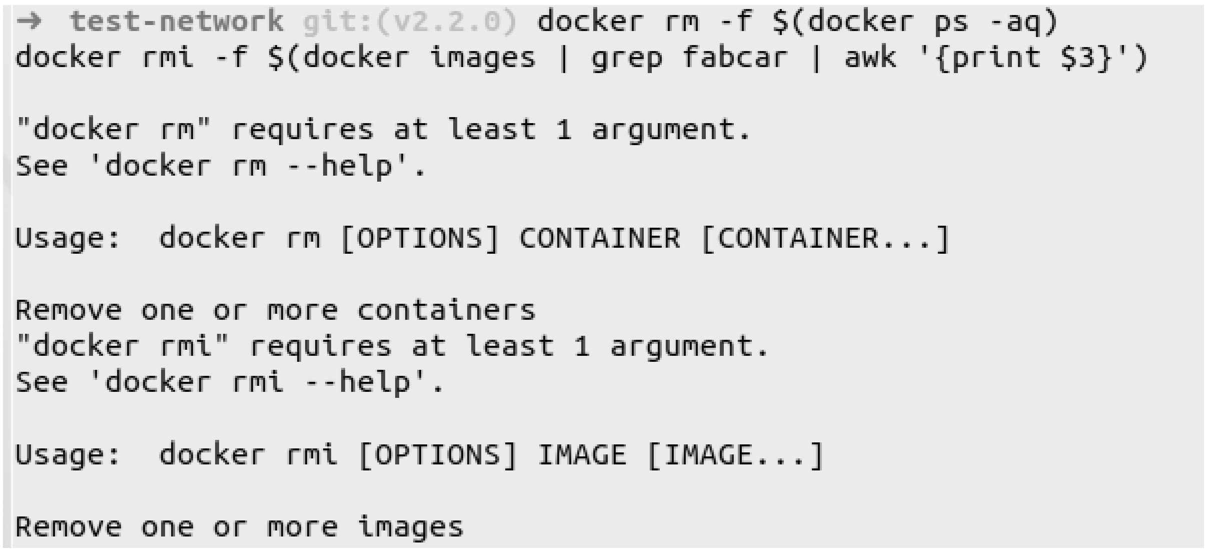
```
cd test-network
./network.sh down
```



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

Remove any existing/running docker containers

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

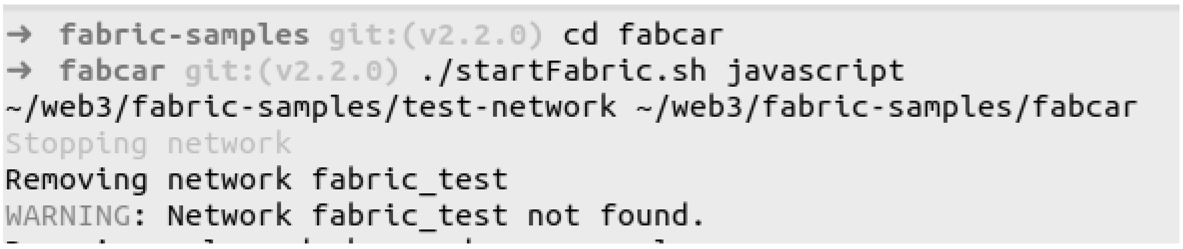


```
→  test-network git:(v2.2.0) docker rm -f $(docker ps -aq)
docker rmi -f $(docker images | grep fabcar | awk '{print $3}')

"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
```

Starting the network

For this Network we will be using Javascript

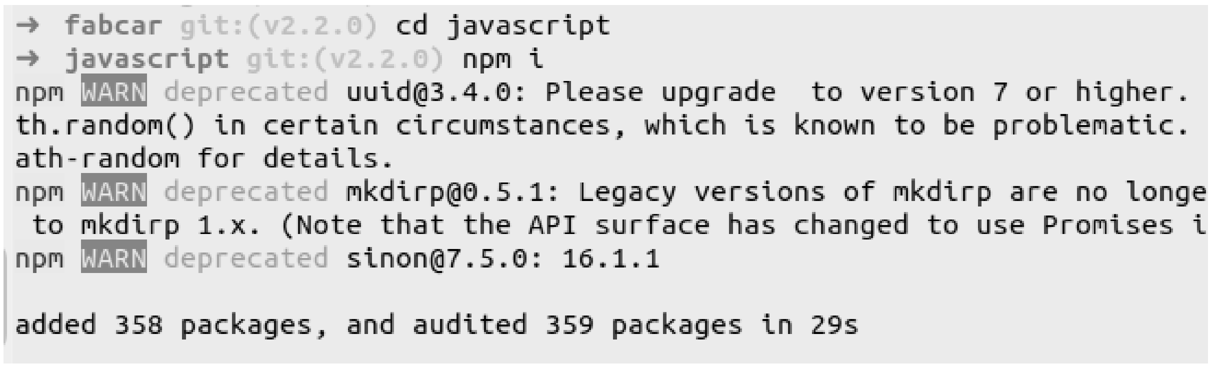
```
cd fabcar
./startFabric.sh javascript
```



```
➜ 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:

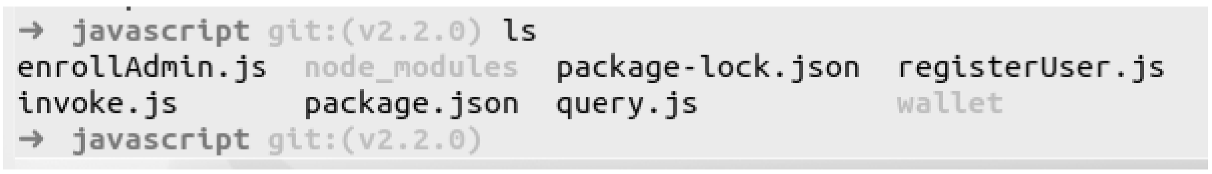
```
cd javascript
npm i
```



```
➜ 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
```

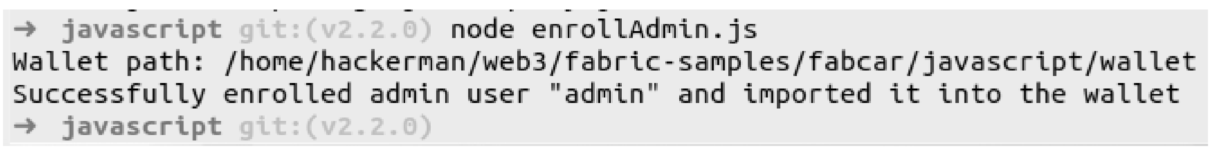


```
➜ javascript git:(v2.2.0) ls
enrollAdmin.js  node_modules  package-lock.json  registerUser.js
invoke.js       package.json  query.js           wallet
➜ javascript git:(v2.2.0)
```

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
```

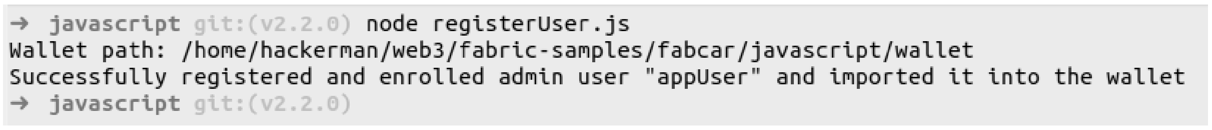


```
→ javascript git:(v2.2.0) node enrollAdmin.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Successfully enrolled admin user "admin" and imported it into the wallet
→ javascript git:(v2.2.0)
```

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
```



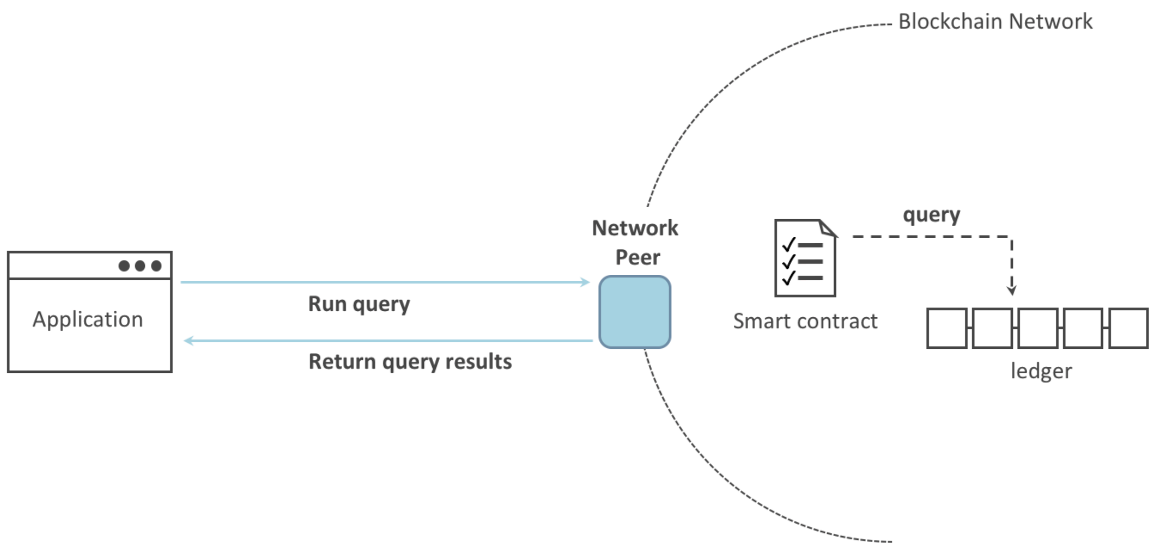
```
→ javascript git:(v2.2.0) node registerUser.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Successfully registered and enrolled admin user "appUser" and imported it into the wallet
→ javascript git:(v2.2.0)
```

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

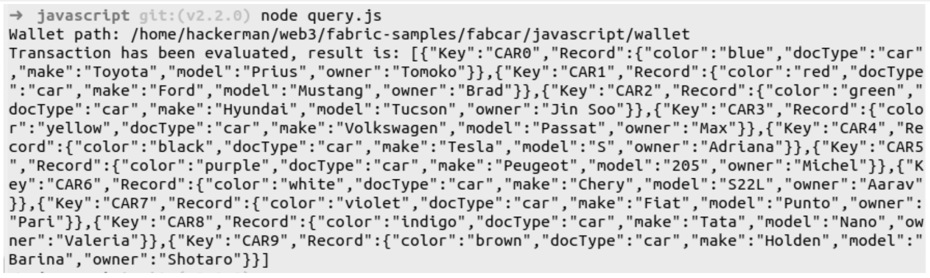
Each peer in a blockchain network hosts a copy of the ledger, and an application program can query the ledger by invoking a smart contract which queries the most recent value of the ledger and returns it to the application.

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
```

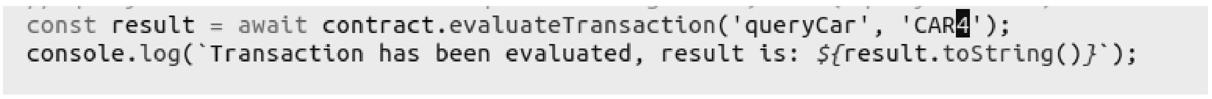


```
➜  javascript git:(v2.2.0) node query.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Transaction has been evaluated, result is: [{"Key":"CAR0","Record":{"color":"blue","docType":"car"
,"make":"Toyota","model":"Prius","owner":"Tomoko"}},{"Key":"CAR1","Record":{"color":"red","docType
":"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":
"Pari"}},{"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"}}]
```

Changing query.js code

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

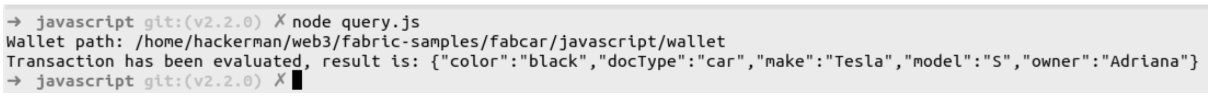
```
const result = await contract.evaluateTransaction('queryCar',
'CAR4');
```



```
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:

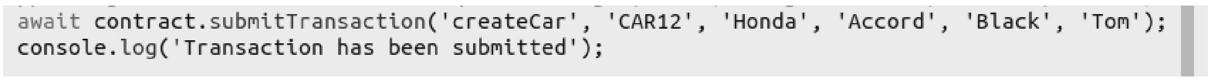
```
node query.js
```



```
➜  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

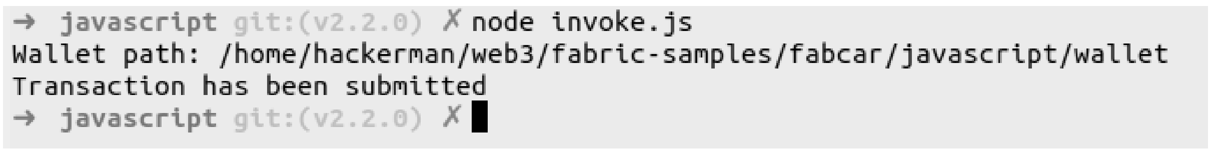
Our first update to the ledger will create a new car. We have a separate program called **invoke.js** that we will use to make updates to the ledger. Just as with queries, use an editor to open the program and navigate to the code block where we construct our transaction and submit it to the network:



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

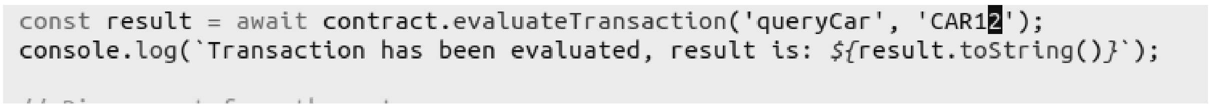
Run the program

```
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) ✗ 
```

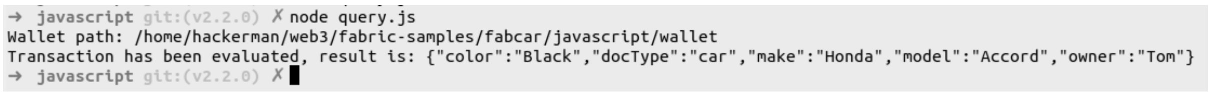
Changing the query.js to fetch the new transaction



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

Save and run

```
node query.js
```

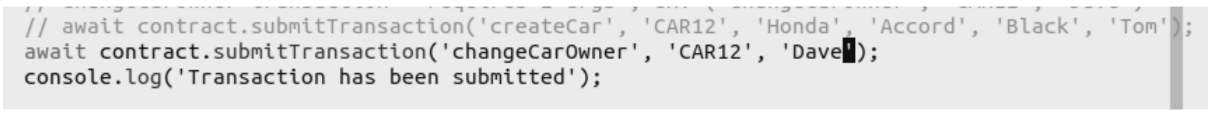


```
➜ javascript git:(v2.2.0) ✗ node query.js
Wallet path: /home/hackerman/web3/fabric-samples/fabcar/javascript/wallet
Transaction has been evaluated, result is: {"color":"Black","docType":"car","make":"Honda","model":"Accord","owner":"Tom"}
➜ javascript git:(v2.2.0) ✗ 
```

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

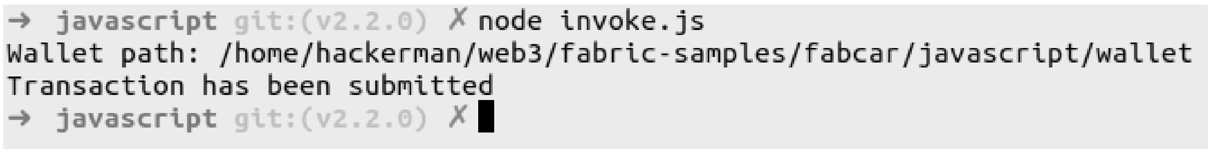
```
await contract.submitTransaction('changeCarOwner', 'CAR12',
'Dave');
```



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

Save and run

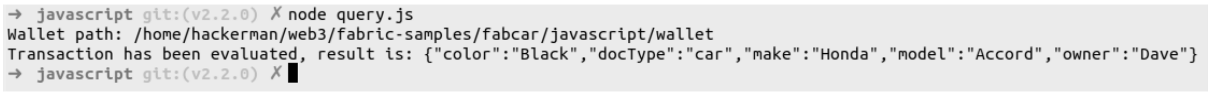
```
node invoke.js
```



```
➜ 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.

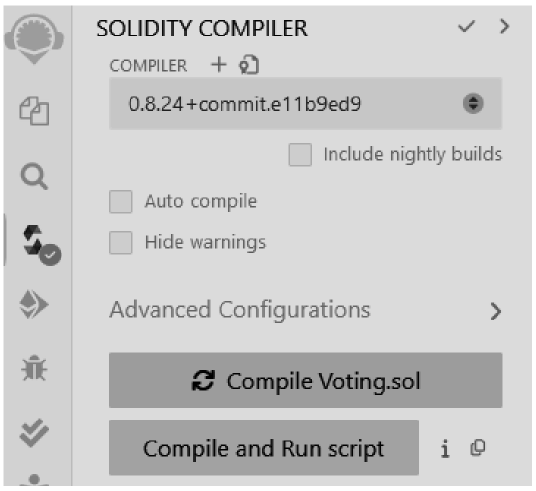
Practical 6

Aim: Develop a voting application using Hyperledger and Ethereum. Build a decentralised app that combines Ethereum's Web3 and Solidity smart contracts with Hyperledger hosting Fabric and Chaincode EVM.

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


SOLIDITY COMPILER

COMPILER

0.8.24+commit.e11b9ed9

Include nightly builds

Auto compile

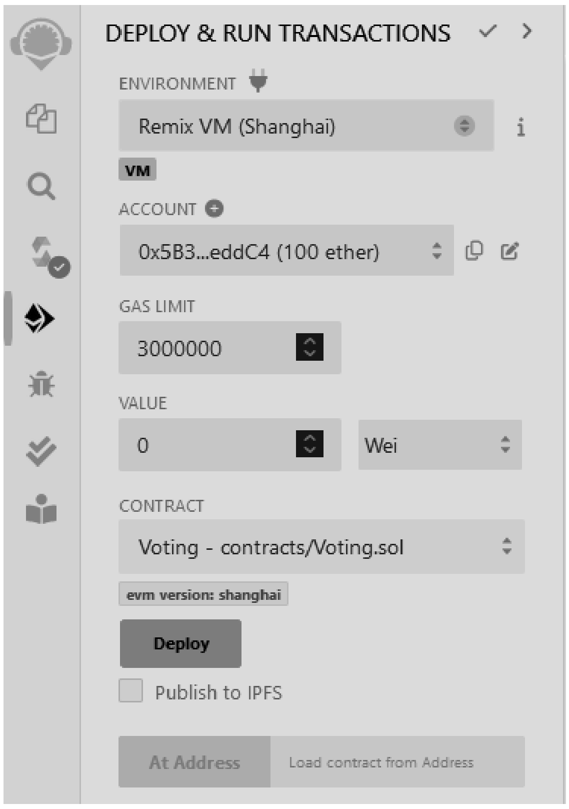
Hide warnings

Advanced Configurations

Compile Voting.sol

Compile and Run script

Deploy The Contract



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Remix VM (Shanghai)

ACCOUNT: 0x5B3...eddC4 (100 ether)

GAS LIMIT: 3000000

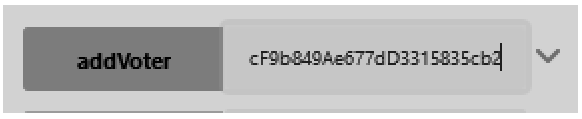
VALUE: 0 Wei

CONTRACT: Voting - contracts/Voting.sol

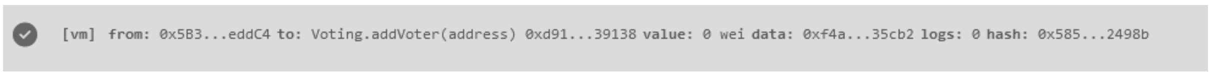
evm version: shanghai

Adding voters

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

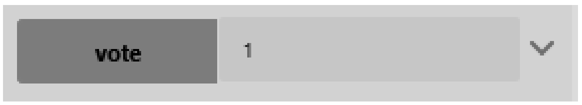


addVoter: cF9b849Ae677dD3315835cb2

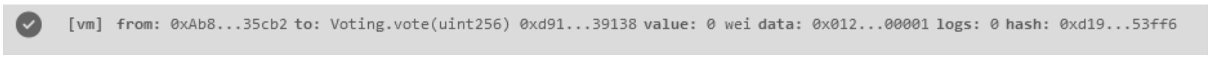


```
[vm] from: 0x5B3...eddC4 to: Voting.addVoter(address) 0xd91...39138 value: 0 wei data: 0xf4a...35cb2 logs: 0 hash: 0x585...2498b
```

Voting

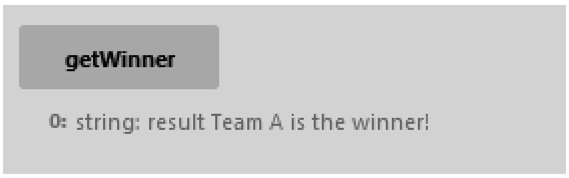


vote: 1



```
[vm] from: 0xAb8...35cb2 to: Voting.vote(uint256) 0xd91...39138 value: 0 wei data: 0x012...00001 logs: 0 hash: 0xd19...53ff6
```

Checking the winner



```
0: string: result Team A is the winner!
```

Practical 7

Aim: Create a blockchain app for loyalty points with Hyperledger Fabric Ethereum Virtual

Creating the smart contract project

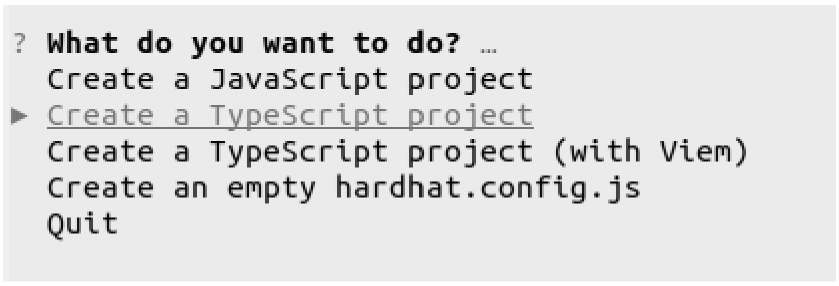
Installing hardhart

```
mkdir smart-contract
cd smart-contract
npm i -D hardhat
```

Creating a hardhat project

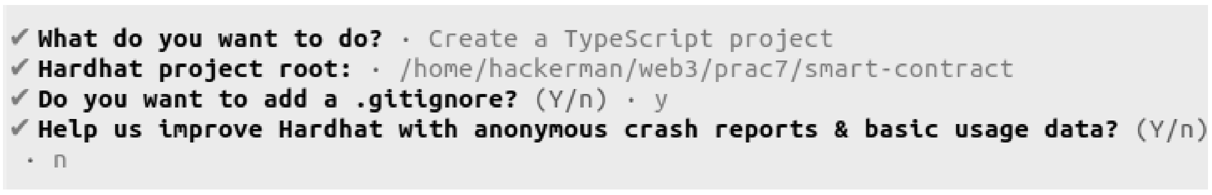
```
npx hardhat init
```

Choose **TypeScript**



```
? What do you want to do? …
  Create a JavaScript project
▶ Create a TypeScript project
  Create a TypeScript project (with Viem)
  Create an empty hardhat.config.js
  Quit
```

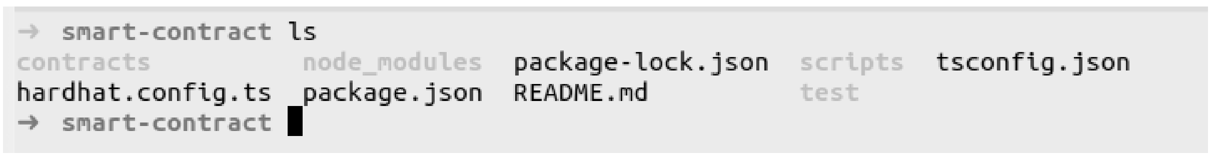
Rest can be left default



```
✔ What do you want to do? · Create a TypeScript project
✔ Hardhat project root: · /home/hackerman/web3/prac7/smart-contract
✔ Do you want to add a .gitignore? (Y/n) · y
✔ Help us improve Hardhat with anonymous crash reports & basic usage data? (Y/n)
 · n
```

Contents after project initialization

```
ls
```

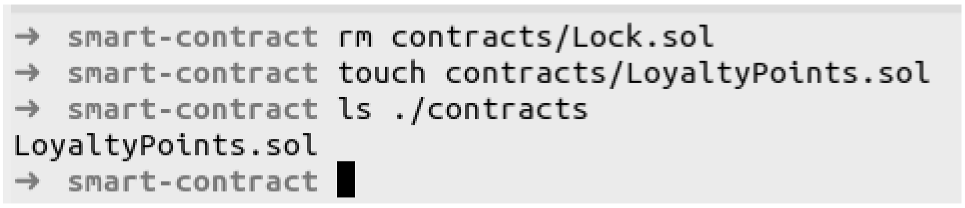


```
➜  smart-contract ls
contracts          node_modules   package-lock.json  scripts   tsconfig.json
hardhat.config.ts  package.json   README.md          test
➜  smart-contract █
```

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
```



```
➜  smart-contract rm contracts/Lock.sol
➜  smart-contract touch contracts/LoyaltyPoints.sol
➜  smart-contract ls ./contracts
LoyaltyPoints.sol
➜  smart-contract 
```

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

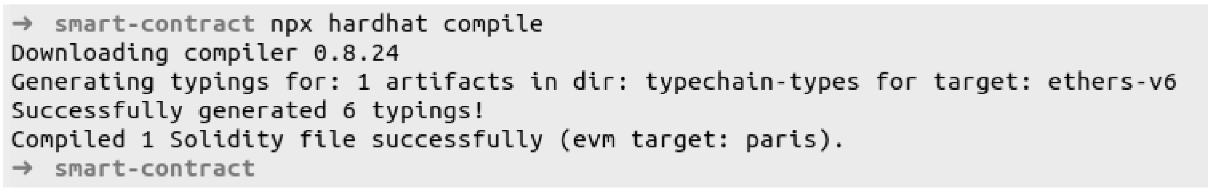
Code:

```
// 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).
➜  smart-contract
```

Deploying the smart contract

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

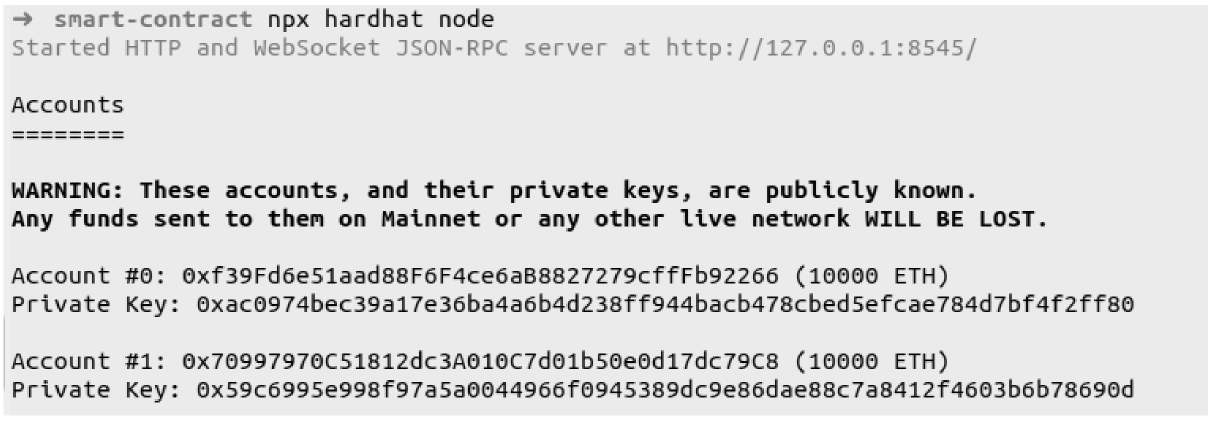
First we need to write the script that will deploy the smart contract open **scripts/deploy.ts** and replace its content with below code

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
```

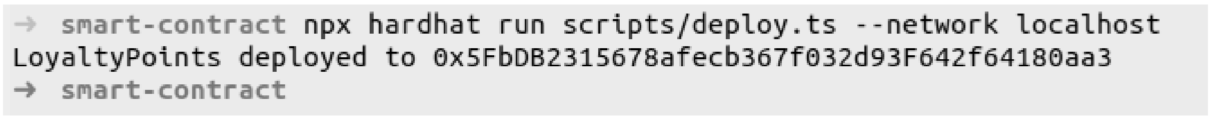


```
➜ smart-contract npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

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: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

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

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
```



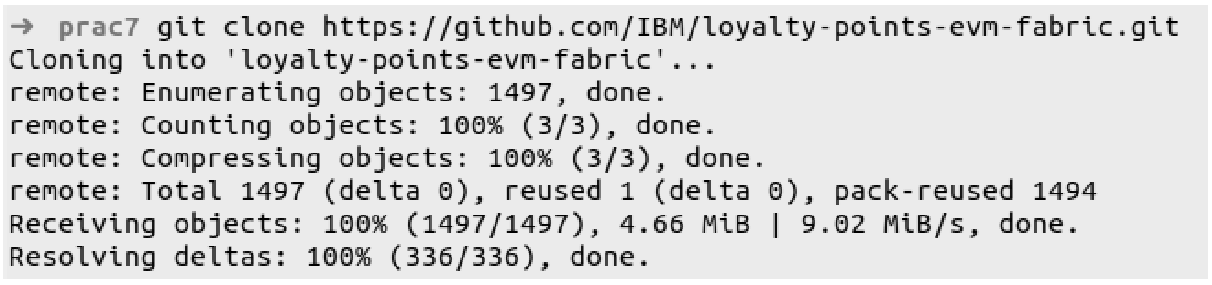
```
➜ smart-contract npx hardhat run scripts/deploy.ts --network localhost
LoyaltyPoints deployed to 0x5FbDB2315678afecb367f032d93F642f64180aa3
➜ smart-contract
```

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

Making the web app

Clone the given repository

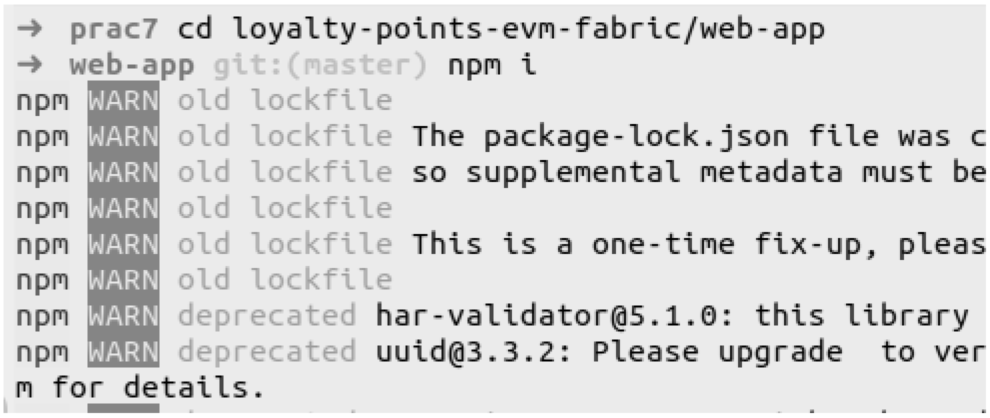
```
git clone https://github.com/IBM/loyalty-points-evm-fabric.git
```



```
➜ 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

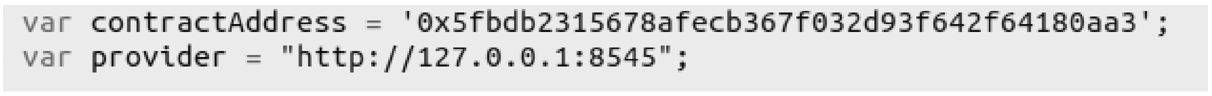
```
cd loyalty-points-evm-fabric/web-app
npm i
```



```
→ 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.
```

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

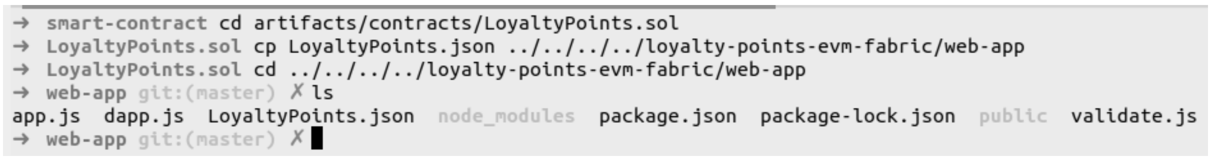
First lets update the **contract address** and **provider.** Contract address will be the **address** it was deployed for you and provider will the **JSON RPC** url we got when we started the **hardhat node**



```
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
```

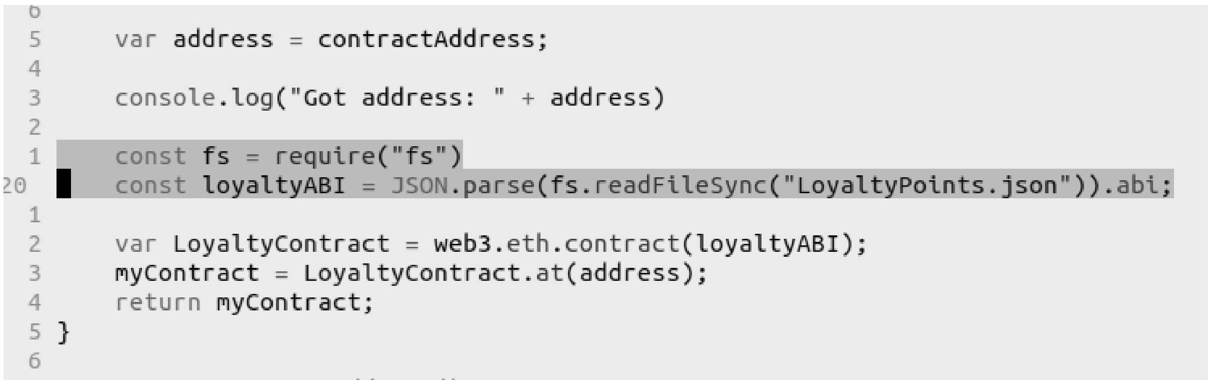


```
→ smart-contract cd artifacts/contracts/LoyaltyPoints.sol
→ LoyaltyPoints.sol cp LoyaltyPoints.json ../../../../loyalty-points-evm-fabric/web-app
→ LoyaltyPoints.sol cd ../../../../loyalty-points-evm-fabric/web-app
→ web-app git:(master) ✗ ls
app.js  dapp.js  LoyaltyPoints.json  node_modules  package.json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

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
```

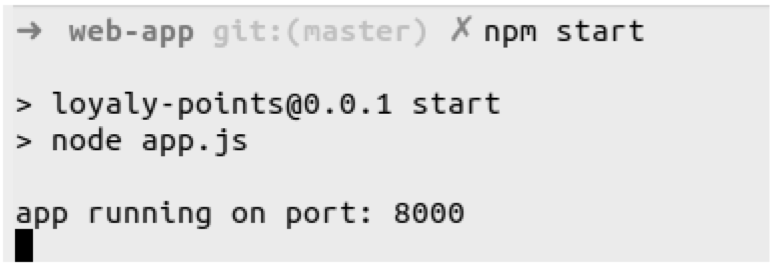
New code will look like this



```
    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;
}
```

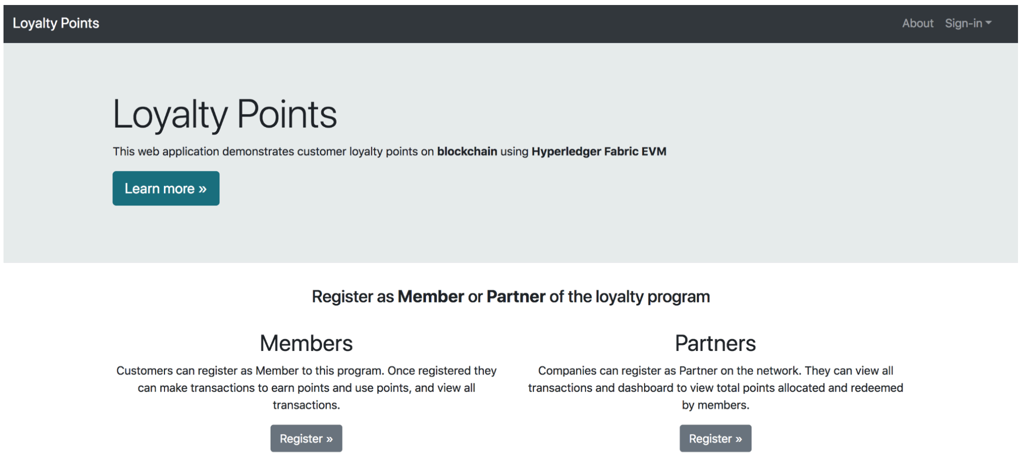
Running the app

```
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



Loyalty Points

This web application demonstrates customer loyalty points on **blockchain** using **Hyperledger Fabric EVM**

Register as **Member** or **Partner** of the loyalty program

Members

Customers can register as Member to this program. Once registered they can make transactions to earn points and use points, and view all transactions.

Partners

Companies can register as Partner on the network. They can view all transactions and dashboard to view total points allocated and redeemed by members.

Practical 8

Aim: Use block chain to track fitness club rewards Build a web app that uses Hyperledger Fabric to track and trace member rewards

Theory:

This is a sample web application that uses a Hyperledger Fabric blockchain to track and trace fitness rewards.

One of the biggest challenges Fitness Clubs face is maintaining members. It is always cheaper to keep a member than to attract new members. To retain members, some clubs will offer loyalty programs, but oftentimes, those are ineffective because they may provide future free months or potentially offer a discount to friends and family. Sometimes, the program only rewards members when they refer friends to the club.

With the Fitcoin app, a Fitness Club can add new members. For active members, you can simulate those members receiving rewards points, redeeming rewards points, and viewing their rewards points history. Clubs can also deactivate members to simulate members cancelling their memberships, something we hope never happens.

Steps:

1. Install Hyperledger Fabric and Composer.
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
```