LFX | Mentorship

HYPERLEDGER FOUNDATION

View Repository

View Site

GO Raft

...

Hyperledger Collaborative Learning - Integrate new BFT protocol with Fabric - Unpaid

HYPERLEDGER
FABRIC

Agenda

- Install Hyperledger Fabric network on Windows 10

- Project -1 Writing Your First Application:

  a- Chaincode Go

  b-Backend application JavaScript or GO

- Project -2 EndToEnd application

  a- (Chaincode) Java

  b- (Backend API) SpringBoot Restful API Backend application Java 17 OpenJDK
  Java 17 is the latest long-term support (LTS)

  c-(Frontend UI) VueJS 2.0 UI SPA (single page application)

HYPERLEDGER
FABRIC

Prerequisites

These prerequisites are recommended for Fabric users. If you are a Fabric developer, you should refer to the instructions for

- **Install Git**

  https://git-scm.com/downloads

- **Install cURL**

  https://curl.haxx.se/download.html

- **Install Go**

  https://go.dev/doc/install

- **Install NodeJs**

  https://nodejs.org/en/download/

HYPERLEDGER
FABRIC

Prerequisites

Windows extras

On Windows 10 you should use the native Docker distribution and you may use the Windows PowerShell. However, for the binaries command to succeed you will still need to have the uname command available. You can get it as part of Git but beware that only the 64bit version is supported.

Run the following commands:

```
git config --global core.autocrlf false
```

```
git config --global core.longpaths true
```

HYPERLEDGER
FABRIC

Prerequisites

Windows extras

You can check the setting of these parameters with the following commands:

```
git config --get core.autocrlf
git config --get core.longpaths
```

These need to be false and true respectively.

HYPERLEDGER
FABRIC

Prerequisites

Windows debug in case if issues

The `curl` command that comes with Git and Docker Toolbox is old and does not handle properly the redirect used in Getting Started. Make sure you have and use a newer version which can be downloaded from the cURL downloads page

https://curl.haxx.se/download.html

HYPERLEDGER
FABRIC

Prerequisites

Docker and Docker Compose

You will need the following installed on the platform on which you will be operating, or developing on (or for), Hyperledger Fabric:

- MacOSX, *nix, or Windows 10: Docker version 17.06.2-ce or greater is required.
  Use Docker for desktop version 2.5.0.1
  https://docs.docker.com/desktop/windows/release-notes/2.x/
- Older versions of Windows: Docker Toolbox - again, Docker version Docker 17.06.2-ce or greater is required.

```
docker --version
```

```
\>docker --version
Docker version 19.03.13, build 4484c46d9d
```

HYPERLEDGER
FABRIC

Prerequisites

(Linux only) Docker and Docker Compose

The following applies to Linux systems running systemd.

Make sure the docker daemon is running.

```
sudo systemctl start docker
```

Optional: If you want the docker daemon to start when the system starts, use the following:

```
sudo systemctl enable docker
```

Add your user to the docker group.

```
sudo usermod -a -G docker <username>
```

HYPERLEDGER
FABRIC

Prerequisites

Docker and Docker Compose

Installing Docker for Mac or Windows, or Docker Toolbox will also install Docker Compose.
If you already had Docker installed, you should check that you have Docker Compose version 1.14.0 or greater installed. If not, we recommend that you install a more recent version of Docker.

```
docker-compose --version
```

HYPERLEDGER
FABRIC

Install Fabric and Fabric Samples

Fabric community think the best way to understand something is to use it yourself. To help you use Fabric, we have created a simple Fabric test network using docker compose, and a set of sample applications that demonstrate its core capabilities. We have also

precompiled Fabric CLI tool binaries and Fabric Docker Images which will be downloaded to your environment, to get you going.

HYPERLEDGER FABRIC

Install Fabric and Fabric Samples

The cURL command in the instructions next slide sets up your environment so that you can run the Fabric test network. Specifically, it performs the following steps:

1. Clones the hyperledger/fabric-samples repository.

2. Downloads the latest Hyperledger Fabric Docker images and tags them as latest

3. Downloads the following platform-specific Hyperledger Fabric CLI tool binaries and config files into the fabric-samples /bin and /config directories. These binaries will help you interact with the test network.

configtxgen	discover	osnadmin	
configtxlator	idemixgen	peer	
cryptogen	orderer	fabric-ca-client	fabric-ca-server

HYPERLEDGER
FABRIC

Install Fabric and Fabric Samples

- Configtxgen - Creating network artifacts(genesis.block/channel.tx)
- Configtxlator - Utility for generating channel configuration
- Cryptogen - Utility for generating key material
- Discovery - Command line client for service discovery
- Idemixgen - Utility for generating key material to be used with identity mixer MSP
- Orderer - node
- Peer - node
- Fabric-ca-client - Client for creating Registering and Enrolling user

HYPERLEDGER
FABRIC

Install Fabric and Fabric Samples

Download Fabric samples, docker images, and binaries.

Download fabric-samples to the $HOME/go/src/github.com/<your_github_userid> directory. This is a Golang Community recommendation for Go projects.

```
$ mkdir -p $HOME/go/src/github.com/<your_github_userid>
$ cd $HOME/go/src/github.com/<your_github_userid>
```

Windows considerations

The recommended directory on Windows is

%USERPROFILE%\go\src\github.com\<your_github_userid>

If using another directory, please consult the Docker documentation for file sharing and the GOPATH environment documentation.

HYPERLEDGER FABRIC

Install Fabric and Fabric Samples

Download Fabric samples, docker images, and binaries.

Download the Fabric Version 2.2 release of Fabric samples, docker images, and binaries.

curl -sSL https://bit.ly/2ysbOFE | bash -s -- <fabric_version> <fabric-ca_version>

curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.2.2 1.4.9

You have completed installing Fabric samples, docker images, and binaries to your system.

HYPERLEDGER
FABRIC

Install Fabric and Fabric Samples

Download Fabric samples, docker images, and binaries.

Troubleshooting

- If you get an error running the cURL command
  - You may have too old a version of cURL that does not handle redirects or an unsupported environment. Please make sure you use a newer version from the cURL downloads page
  - Alternately, there might be an issue with the bit.ly, please retry the command with the un-shortened URL:

```
curl -sSL https://raw.githubusercontent.com/hyperledger/fabric/release-2.2/scripts/bootstrap.sh | bash -s
```

HYPERLEDGER
FABRIC

Fabric Contract APIs and Application APIs

Hyperledger Fabric offers a number of APIs to support developing smart contracts (chaincode) in various programming languages. Smart contract APIs are available for Go, Node.js, and Java.

- Go contract API and documentation.
- Node.js contract API and documentation.
- Java contract API and documentation.

HYPERLEDGER
FABRIC

Getting Started - Run Fabric

Now that you have downloaded Fabric and the samples, you can start running Fabric

- Running a Test Network
- Running a Fabric Application

HYPERLEDGER
FABRIC

Bring up the test network

You can find the scripts to bring up the network in the test-network directory of the fabric-samples repository. Navigate to the test network directory by using the following

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

In this directory, you can find an annotated script, network.sh, that stands up a Fabric network using the Docker images on your local machine. You can run ./network.sh -h to print the script help text:

HYPERLEDGER
FABRIC

Bring up the test network

From inside the test-network directory, run the following command to remove any containers or artifacts from any previous runs:

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

You can then bring up the network by issuing the following command. You will experience problems if you try to run the script from another directory:

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

HYPERLEDGER
FABRIC

Bring up the test network

This command creates a Fabric network that consists of two peer nodes, one ordering node. No channel is created when you run ./network.sh up, though we will get there in a future step. If the command completes successfully, you will see the logs of the nodes being created:

```
Creating network "fabric_test" with the default driver
Creating volume "net_orderer.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating peer0.org2.example.com ... done
Creating orderer.example.com    ... done
Creating peer0.org1.example.com ... done
Creating cli                    ... done
CONTAINER ID   IMAGE                               COMMAND             CREATED         STATUS
1667543b5634   hyperledger/fabric-tools:latest     "/bin/bash"         1 second ago    Up Less than
b6b117c81c7f   hyperledger/fabric-peer:latest      "peer node start"   2 seconds ago   Up 1 second
703ead770e05   hyperledger/fabric-orderer:latest   "orderer"           2 seconds ago   Up Less than
718d43f5f312   hyperledger/fabric-peer:latest      "peer node start"   2 seconds ago   Up 1 second
```

HYPERLEDGER
FABRIC

The components of the test network

After your test network is deployed, you can take some time to examine its components. Run the following command to list all of Docker containers that are running on your machine. You should see the three nodes that were created by the network.sh script:

```
docker ps -a
```

HYPERLEDGER
FABRIC

Creating a channel

```
./network.sh createChannel
```

If the command was successful, you can see the following message printed in your logs:

========= Channel successfully joined ==========

./network.sh createChannel -c channel1

If you want to bring up the network and create a channel in a single step, you can use the up and createChannel modes together:

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

HYPERLEDGER
FABRIC

Creating a channel

To verify the peer has joined the channel

- ssh inside the peer docker image

  docker exec -it peer0.org1.example.com sh

  You should go inside the peer docker image work directory:

  /opt/gopath/src/github.com/hyperledger/fabric/peer #

- List the channels that peer0 has joined by running:

  # peer channel list

```
Channels peers has joined:
mychannel
```

HYPERLEDGER FABRIC

Chaincode

The chaincode is a program that handles business logic agreed to by members of the network, so it may be considered as a “smart contract. Chaincode runs in a secured Docker container isolated from the endorsing peer process.

Chaincode initializes and manages ledger state through transactions submitted by applications.

It is installed and instantiated through an SDK or CLI onto a network of Hyperledger Fabric peer nodes, enabling interaction with that network’s shared ledger.

For more info about: Fabric chaincode lifecycle

https://hyperledger-fabric.readthedocs.io/en/release-2.2/chaincode_lifecycle.html

HYPERLEDGER FABRIC

Starting a chaincode on the channel

Deploy Chaincode in GO

After you have created a channel, you can start using smart contracts to interact with the channel ledger. Smart contracts contain the business logic that governs assets on the blockchain ledger. Applications run by members of the network can invoke smart contracts to create assets on the ledger, as well as change and transfer those assets. Applications also query smart contracts to read data on the ledger.

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

The deployCC subcommand will install the asset-transfer (basic) chaincode on peer0.org1.example.com and peer0.org2.example.com and then deploy the chaincode on the channel specified using the channel flag (or mychannel if no channel is specified). If you are deploying a chaincode for the first time, the script will install the chaincode dependencies. You can use the language flag, -l, to install the Go, typescript or javascript versions of the chaincode.

HYPERLEDGER
FABRIC

Interacting with the network

After you bring up the test network, you can use the peer CLI to interact with your network. The peer CLI allows you to invoke deployed smart contracts, update channels, or install and deploy new smart contracts from the CLI.

You can find the peer binaries in the bin folder of the fabric-samples repository. Use the following command to add those binaries to your CLI Path:

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

You also need to set the FABRIC_CFG_PATH to point to the core.yaml file in the fabric-samples repository:

```
export FABRIC_CFG_PATH=$PWD/../config/
```

HYPERLEDGER
FABRIC

Interacting with the network

You can now set the environment variables that allow you to operate the peer CLI as Org1:

```
# Environment variables for Org1

export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051
```

HYPERLEDGER
FABRIC

Interacting with the network

The *CORE_PEER_TLS_ROOTCERT_FILE* and *CORE_PEER_MSPCONFIGPATH* environment variables point to the Org1 crypto material in the *organizations* folder.
If you used *./network.sh deployCC -ccl go* to install and start the asset-transfer (basic) chaincode, you can invoke the *InitLedger* function of the (Go) chaincode to put an initial list of assets on the ledger (if using typescript or javascript *./network.sh deployCC -ccl javascript* for example, you will invoke the *InitLedger* function of the respective chaincodes).

Run the following command to initialize the ledger with assets:

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C
mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses localhost:9051 --
tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c
'{"function":"InitLedger","Args":[]}'
```

HYPERLEDGER
FABRIC

Interacting with the network

If successful, you should see similar output to below:

INFO 001 Chaincode invoke successful. result: status:200

```
!@WLRBSA2400911M1 MINGW64 ~/Desktop/Desktop/test/fabric-samples/test-netwo
rk ((v2.2.3))
$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.e
xample.com --tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/
orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychan
nel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles ${PWD}/organizati
ons/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt -
-peerAddresses localhost:9051 --tlsRootCertFiles ${PWD}/organizations/peerOrgani
zations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c '{"function"
:"InitLedger","Args":[]}'
←[34m2021-12-25 23:59:27.473 EST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 0
01←[0m Chaincode invoke successful. result: status:200
```

HYPERLEDGER
FABRIC

Interacting with the network

You can now query the ledger from your CLI. Run the following command to get the list of assets that were added to your channel ledger:

```
peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'
```

If successful, you should see the following output:


```
[
  {"ID": "asset1", "color": "blue", "size": 5, "owner": "Tomoko", "appraisedValue": 300},
  {"ID": "asset2", "color": "red", "size": 5, "owner": "Brad", "appraisedValue": 400},
  {"ID": "asset3", "color": "green", "size": 10, "owner": "Jin Soo", "appraisedValue": 500},
  {"ID": "asset4", "color": "yellow", "size": 10, "owner": "Max", "appraisedValue": 600},
  {"ID": "asset5", "color": "black", "size": 15, "owner": "Adriana", "appraisedValue": 700},
  {"ID": "asset6", "color": "white", "size": 15, "owner": "Michel", "appraisedValue": 800}
]
```


HYPERLEDGER
FABRIC

Bring up the network with Certificate Authorities

Hyperledger Fabric uses public key infrastructure (PKI) to verify the actions of all network participants. Every node, network administrator, and user submitting transactions needs to have a public certificate and private key to verify their identity.

These identities need to have a valid root of trust, establishing that the certificates were issued by an organization that is a member of the network.

The *network.sh* script creates all of the cryptographic material that is required to deploy and operate the network before it creates the peer and ordering nodes.

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

HYPERLEDGER
FABRIC

Bring up the network with Certificate Authorities

The test network uses the Fabric CA client to register node and user identities with the CA of each organization. The script then uses the enroll command to generate an MSP folder for each identity. The MSP folder contains the certificate and private key for each identity, and establishes the identity's role and membership in the organization that operated the CA.

```
organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/
└── msp
    ├── IssuerPublicKey
    ├── IssuerRevocationPublicKey
    ├── cacerts
    │   └── localhost-7054-ca-org1.pem
    ├── config.yaml
    ├── keystore
    │   └── 58e81e6f1ee8930df46841bf88c22a08ae53c1332319854608539ee78ed2fd65_sk
    ├── signcerts
    │   └── cert.pem
    └── user
```

HYPERLEDGER
FABRIC

Writing Your First Application

Clean up

- docker system prune -a --volumes
- rm –rf fabric-samples
- curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.2.2 1.4.9

HYPERLEDGER
FABRIC

Writing Your First Application

- **Set up the blockchain network**

We will also use our sample programs and a deployed Certificate Authority to generate the X.509 certificates that an application needs to interact with a permissioned blockchain.

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

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

This command will deploy the Fabric test network with two peers, an ordering service, and three certificate authorities (Orderer, Org1, Org2). Instead of using the cryptogen tool, we bring up the test network using Certificate Authorities, hence the -ca flag. Additionally, the org admin user registration is bootstrapped when the Certificate Authority is started. In a later step, we will show how the sample application completes the admin enrollment.

HYPERLEDGER
FABRIC

Writing Your First Application

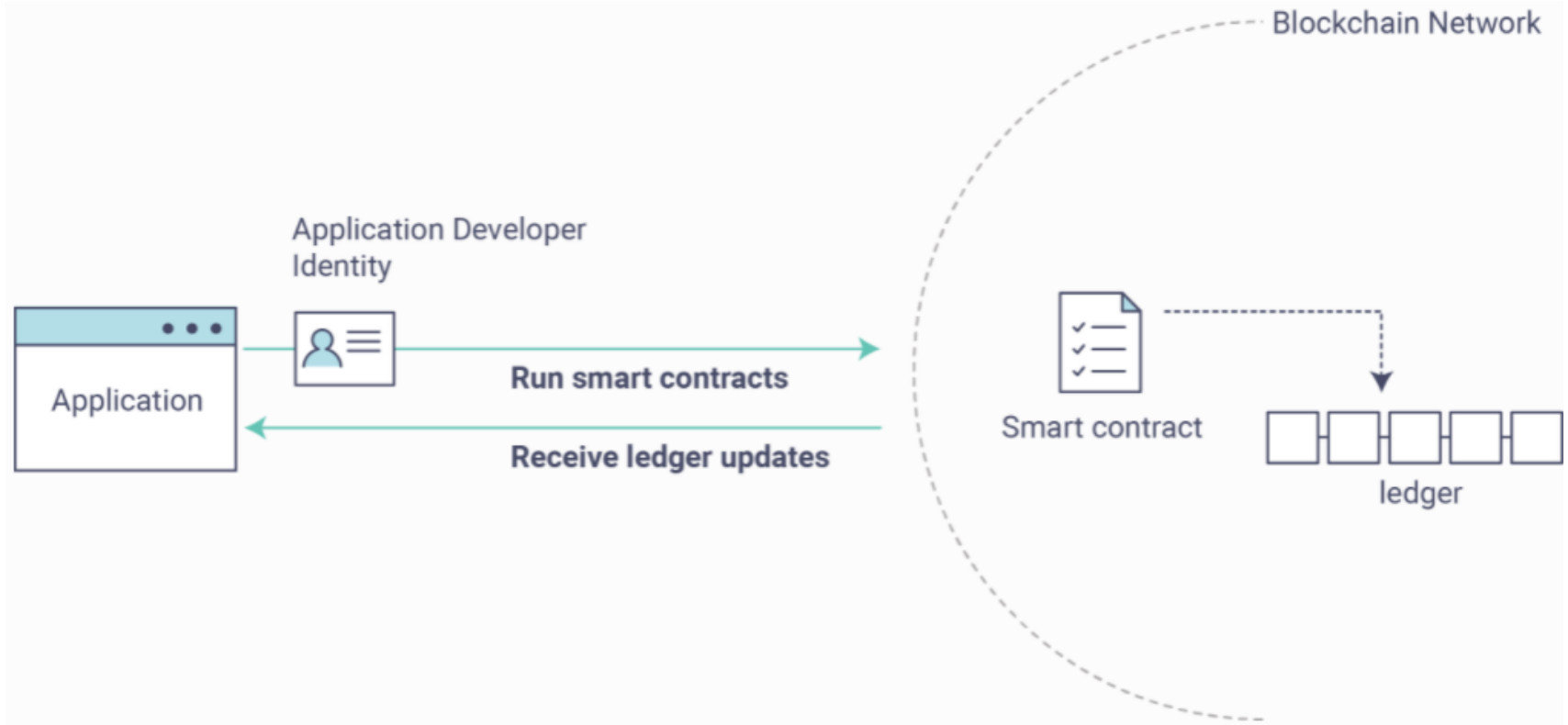
- **Set up the blockchain network with Couch DB**

```
./network.sh up createChannel -c mychannel -ca -s couchdb
```

This command will deploy a Fabric network consisting of a single channel named mychannel with two organizations (each maintaining one peer node), certificate authorities, and an ordering service while using CouchDB as the state database. Either LevelDB or CouchDB may be used with collections. CouchDB was chosen to demonstrate how to use indexes with private data

HYPERLEDGER
FABRIC

Writing Your First Application



HYPERLEDGER
FABRIC

Writing Your First Application

- **chaincode application**

1. Smart contract itself, implementing the transactions that involve interactions with the ledger. The smart contract (chaincode) is located in the following fabric-samples directory:

asset-transfer-basic/chaincode-(javascript, java, go, typescript

```
./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-javascript/ -ccl javascript
```

Note: Behind the scenes, this script uses the **chaincode lifecycle** to package, install, query installed chaincode, approve chaincode for both Org1 and Org2, and finally commit the chaincode.

HYPERLEDGER FABRIC

Writing Your First Application

- **Web application**

2. Sample application: which makes calls to the blockchain network, invoking transactions implemented in the chaincode (smart contract). The application is located in the following *fabric-samples* directory:

```
1. cd ../asset-transfer-basic/application-javascript/
```

applications use a software development kit (SDK) to access the **APIs** that permit queries and updates to the ledger.

```
2. npm install
```

Node.js module; it enables an application to use identities, wallets, and gateways to connect to channels, submit transactions, and wait for notifications. This tutorial also uses the *fabric-ca-client* module to enroll users with their respective certificate authorities, generating a valid identity which is then used by the *fabric-network* module to interact with the blockchain network.

HYPERLEDGER
FABRIC

Writing Your First Application

- **Web application**

```
$ ls
app.js  node_modules/  package.json  package-lock.json
```

The first part of the following section involves communication with the Certificate Authority. You may find it useful to stream the CA logs when running the upcoming programs by opening a new terminal shell and running docker logs -f ca_org1.

When we started the Fabric test network back in the first step, an admin user — literally called admin — was created as the registrar for the Certificate Authority (CA). Our first step is to generate the private key, public key, and X.509 certificate for admin by having the application call the enrollAdmin . This process uses a Certificate Signing Request (CSR) — the private and public key are first generated locally, and the public key is then sent to the CA which returns an encoded certificate for use by the application. These credentials are then stored in the wallet, allowing us to act as an administrator for the CA.

HYPERLEDGER
FABRIC

Writing Your First Application

- **Web application**

Run the application and then step through each of the interactions with the smart contract functions. From the *asset-transfer-basic/application-javascript* directory, run the following command:

```
node app.js
```

It is important to note that enrolling the admin and registering the app user are interactions that take place between the application and the Certificate Authority, not between the application and the chaincode. If you examine the chaincode in *asset-transfer-basic/chaincode-javascript/lib* you will find that the chaincode does not contain any functionality that supports enrolling the admin or registering the user.

HYPERLEDGER
FABRIC

Writing Your First Application

First, the application enrolls the admin user

In the sample application code below, you will see that after getting reference to the common connection profile path, making sure the connection profile exists, and specifying where to create the wallet, enrollAdmin() is executed and the admin credentials are generated from the Certificate Authority.

```
async function main() {
   try {
      // build an in memory object with the network configuration (also known as a connection profile)
      const ccp = buildCCPOrg1();
      // build an instance of the fabric ca services client based on
      // the information in the network configuration
      const caClient = buildCAClient(FabricCAServices, ccp, 'ca.org1.example.com');
      // setup the wallet to hold the credentials of the application user
      const wallet = await buildWallet(Wallets, walletPath);
      // in a real application this would be done on an administrative flow, and only once
      await enrollAdmin(caClient, wallet, mspOrg1);
```

This command stores the CA administrator's credentials in the wallet directory. You can find administrator's certificate and private key in the wallet/admin.id file.

HYPERLEDGER
FABRIC

Writing Your First Application

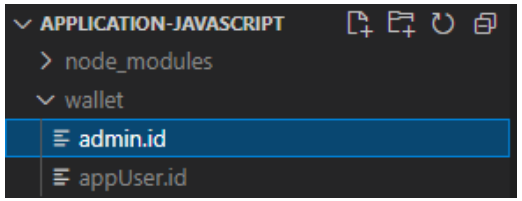
If you decide to start over by taking down the network and bringing it back up again, you will have to delete the wallet folder and its identities prior to re-running the javascript application or you will get an error. This happens because the Certificate Authority and its database are taken down when the test-network is taken down but the original wallet still remains in the application-javascript directory so it must be deleted. When you re-run the sample javascript application, a new wallet and credentials will be generated.

Second, the application registers and enrolls an application user

Now that we have the administrator's credentials in a wallet, the application uses the admin user to register and enroll an app user which will be used to interact with the blockchain network. The section of the application code is shown below.

```
    // in a real application this would be done only when a new user was required to be added and would be part of an administrative flow
    await registerAndEnrollUser(caClient, wallet, mspOrg1, org1UserId, 'org1.department1');
```

Similar to the admin enrollment, this function uses a CSR to register and enroll appUser and store its credentials alongside those of admin in the wallet. We now have identities for two separate users — admin and appUser — that can be used by our application.



HYPERLEDGER
FABRIC

Writing Your First Application

Third, the sample application prepares a connection to the channel and smart contract

the application is getting reference to the Contract using the contract name and channel name via Gateway:

```
try {
    // setup the gateway instance
    // The user will now be able to create connections to the fabric network and be able to
    // submit transactions and query. All transactions submitted by this gateway will be
    // signed by this user using the credentials stored in the wallet.
    await gateway.connect(ccp, {
        wallet,
        identity: org1UserId,
        discovery: { enabled: true, asLocalhost: true } // using asLocalhost as this gateway is using a fabric network deployed locally
    });

    // Build a network instance based on the channel where the smart contract is deployed
    const network = await gateway.getNetwork(channelName);

    // Get the contract from the network.
    const contract = network.getContract(chaincodeName);
```

HYPERLEDGER
FABRIC

Writing Your First Application

Fourth, the application initializes the ledger with some sample data

The submitTransaction() function is used to invoke the chaincode InitLedger function to populate the ledger with some sample data.

- **Sample application** 'InitLedger' call

```
// Initialize a of asset data on the channel using the chaincode 'InitLedger' function.
// This of transaction would only be run once by an application the first time it was started after it
// deployed the first time. Any updates to the chaincode deployed later would likely not need to run
// an "init“ function.
console.log('\n--> Submit Transaction: InitLedger, function creates the initial set of assets on the ledger’);
await contract.submitTransaction('InitLedger'); console.log('*** Result: committed');
```

- **Chaincode** 'InitLedger' function

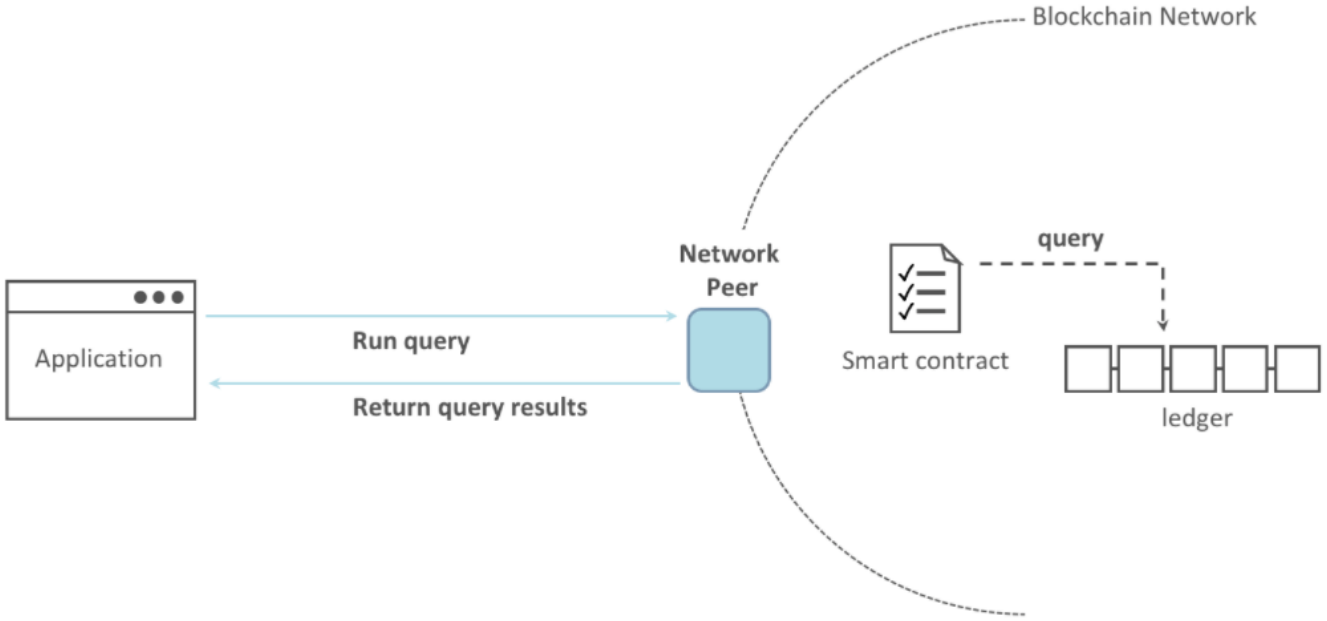
```
async InitLedger(ctx) {
  const assets = [ {
                    ID: 'asset1’,
                    Color: 'blue’,
                    Size: 5,
                    Owner: 'Tomoko’,  AppraisedValue:
300,
 },{…
```

HYPERLEDGER FABRIC

Writing Your First Application

Fifth, the application invokes each of the chaincode functions

Each peer in a blockchain network hosts a copy of the ledger. An application program can view the most recent data from the ledger using read-only invocations of a smart contract running on your peers called a query.



HYPERLEDGER
FABRIC

Writing Your First Application

Fifth, the application invokes each of the chaincode functions

The most common queries involve the current values of data in the ledger – its world state. The world state is represented as a set of key-value pairs, and applications can query data for a single key or multiple keys. Moreover, you can use complex queries to read the data on the ledger when you use CouchDB as your state database and model your data in JSON.

- Sample application 'GetAllAssets' call
- Chaincode 'GetAllAssets' function

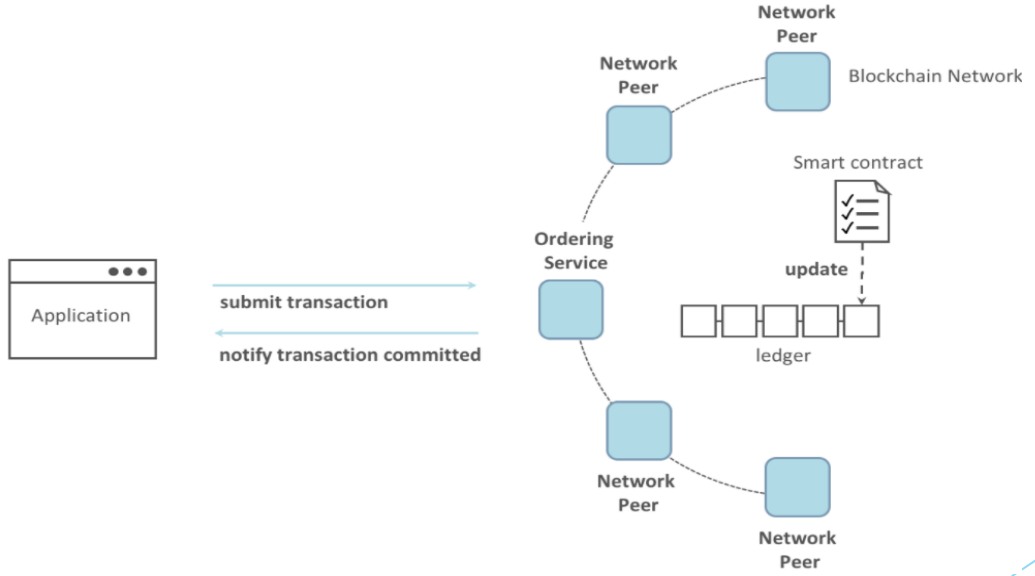
  - GetAllAssets
  - CreateAsset
  - ReadAsset
  - AssetExists
  - UpdateAsset
  - TransferAsset

HYPERLEDGER FABRIC

Writing Your First Application

Updating the ledger

From an application perspective, updating the ledger is simple. An application submits a transaction to the blockchain network, and when it has been validated and committed, the application receives a notification that the transaction has been successful. Behind the scenes, this involves the process of consensus whereby the different components of the blockchain network work together to ensure that every proposed update to the ledger is valid and performed in an agreed and consistent order.



HYPERLEDGER
FABRIC

EndToEnd Hyperledger Fabric Web-application

- Production ready application / using cutting edge technology :

  a- Chaincode Java

  b-SpringBoot Restful API Backend application Java OpenJDK 11-16-17

  c-VueJS 2.0 UI SPA (single page application)

In class

HYPERLEDGER
FABRIC

Hyperledger Fabric Documentation

https://hyperledger-fabric.readthedocs.io/en/release-2.2/

https://github.com/IBM/blockchainbean2

https://github.com/IBM/ibm-cloud-functions-serverless-blockchain

https://www.slideshare.net/HoreaPorutiu/deploy-a-blockchain-webapp-with-hyperledger-fabric-14-concepts-code

https://github.com/IBM/evote

HYPERLEDGER
FABRIC

Troubleshot

If problems persist, you can remove your images and start from scratch:

```
docker rm -f $(docker ps -aq)
```

```
docker rmi -f $(docker images -q)
```

```
docker volume ls
docker volume prune
```

```
docker system prune -a --volumes
```