Hyperledger Fabric

Theory and Applications of Blockchain (CS61065)

Hyperledger Foundation

HYPERLEDGER FOUNDATION

https://www.hyperledger.org/

- Open source community - focused on **enterprise-grade blockchain deployments**.
- Home for various distributed ledger frameworks including: Hyperledger Fabric, Sawtooth, Indy, etc.

Hyperledger Foundation

HYPERLEDGER FOUNDATION

https://www.hyperledger.org/

- Open source community - focused on **enterprise-grade blockchain deployments**.

- Home for various distributed ledger frameworks including: Hyperledger Fabric, Sawtooth, Indy, etc.

- Different companies / organizations want to collaborate
- Closed group: members know each other
- Do not fully trust each other
- Distributed shared ledger – based on permissioned consensus

Hyperledger Foundation Projects

HYPERLEDGER CELLO

↓

Tooling to serve as operational dashboard for Blockchains

HYPERLEDGER EXPLORER

↓

Tooling to invoke, deploy or query blocks

HYPERLEDGER FABRIC

↓

Permissioned Enterprise Blockchain

HYPERLEDGER BURROW

↓

Permissioned, EVM Based, BFT Consensus

HYPERLEDGER INDY

↓

Identity Management

What is Hyperledger Fabric ?

- **Open source,** enterprise-grade
- **Permissioned** DLT platform

- **Modular blockchain framework**
  - Designed for developing blockchain-based products, solutions, and applications using plug-and-play components that are aimed for use within private enterprises.

- Pluggable Components: Including **consensus and membership services**.

- **Smart contracts in general purpose languages** such as **Java, Go and Node.js**.

- Fabric introduces a new architecture for transactions that we call **execute-order-validate**.

Install Hyperledger Fabric - Prerequisites

- **Git**
  - https://git-scm.com/downloads
- **cURL**
  - https://curl.se/download.html
- **Docker** (Docker version 19.03.12 or greater is required)
  - https://docs.docker.com/engine/install/
- **Go**
  - https://golang.org/doc/install
- **Docker Compose** (Docker Compose version 1.27.2 or greater installed)
  - https://docs.docker.com/compose/install/

https://hyperledger-fabric.readthedocs.io/en/release-2.4/getting_started.html
https://hyperledger-fabric.readthedocs.io/en/release-2.4/prereqs.html

Installation

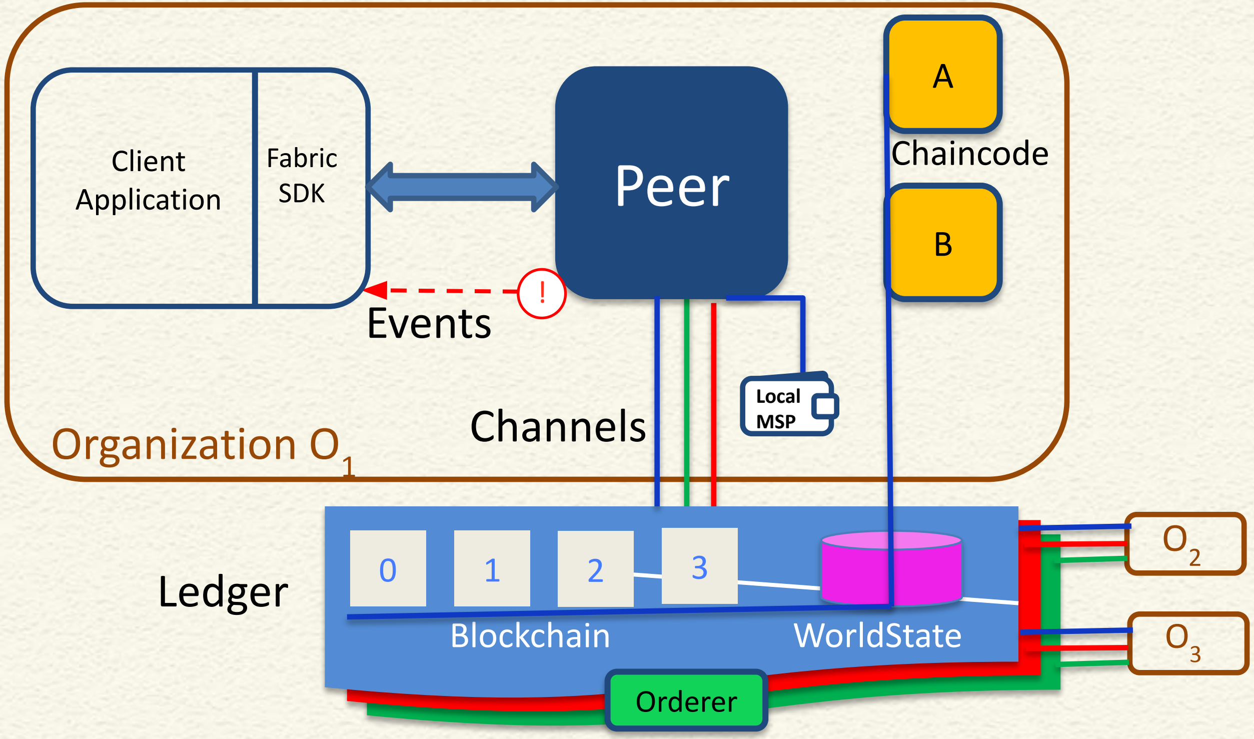
- **Go to the folder where you want to keep the fabric repository folder**
- **Open the terminal**
- **Type**

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

    ```
    Example: curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.4.6 1.5.3
    ```
  - This will clone the fabric repository from github and install all other required files
  - A folder will get created with name “fabric-samples”

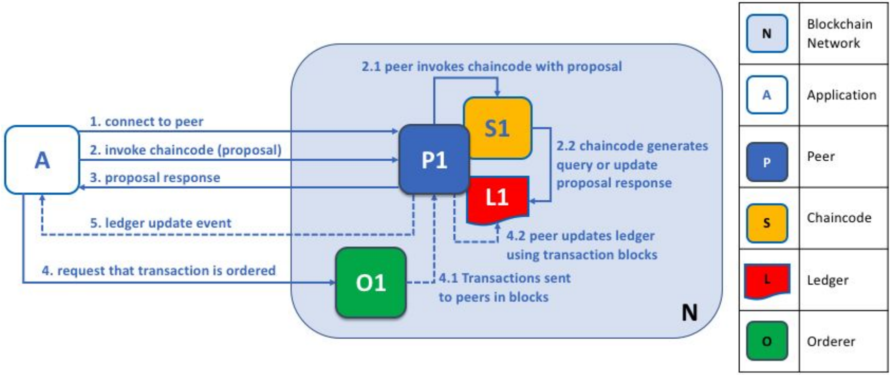
https://hyperledger-fabric.readthedocs.io/en/release-2.4/install.html

Fabric Architecture



Fabric Test Network

- There is a folder “test-network” inside your “fabric-samples” folder.
- This folder contains a script “network.sh” which will bring up the test network.
- **We will use this test network to explore Fabric by running nodes on our local machine.**


2.1 peer invokes chaincode with proposal
1. connect to peer
2. invoke chaincode (proposal)
3. proposal response
A
P1
S1
L1
2.2 chaincode generates query or update proposal response
5. ledger update event
4.2 peer updates ledger using transaction blocks
4. request that transaction is ordered
O1
4.1 Transactions sent to peers in blocks
N
N Blockchain Network
A Application
P Peer
S Chaincode
L Ledger
O Orderer

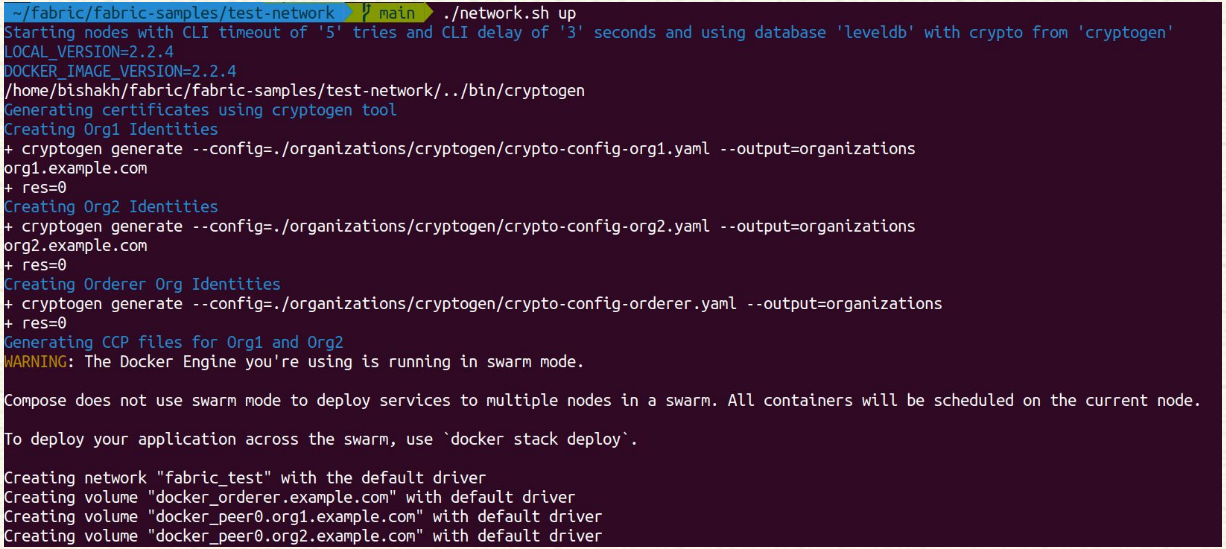
https://hyperledger-fabric.readthedocs.io/en/release-2.4/test_network.html

Start Test Network

Navigate to the directory where you have installed fabric samples.

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

– To tear down the network.

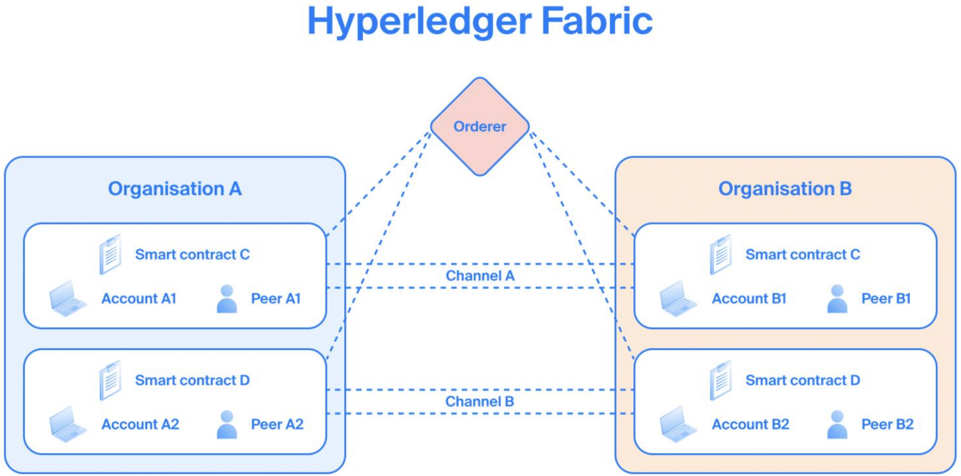


```
~/fabric/fabric-samples/test-network  main  ./network.sh up
Starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'leveldb' with crypto from 'cryptogen'
LOCAL_VERSION=2.2.4
DOCKER_IMAGE_VERSION=2.2.4
/home/bishakh/fabric/fabric-samples/test-network/../bin/cryptogen
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
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Creating network "fabric_test" with the default driver
Creating volume "docker_orderer.example.com" with default driver
Creating volume "docker_peer0.org1.example.com" with default driver
Creating volume "docker_peer0.org2.example.com" with default driver
```

Components of Hyperledger Fabric

Components of Hyperleder Fabric

- Ledgers (one per channel – comprised of the blockchain and the state database)
- Smart contract(s) (aka chaincode)
- *Organizations*
- *Peer nodes*
- *Ordering service(s)*
- *Channel(s)*
- *Fabric Certificate Authorities*



https://chainstack.com/picking-an-enterprise-blockchain-protocol-to-develop-on-hyperledger-fabric/

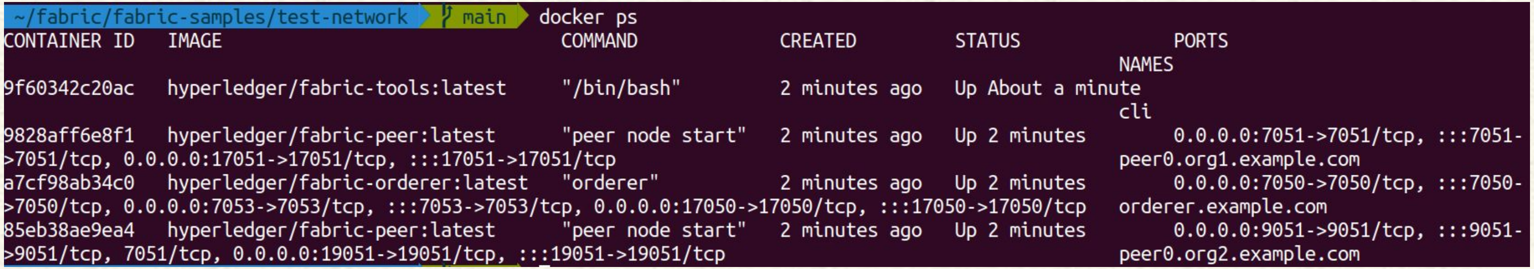
Fabric Test Network

- Real network consists of multiple organizations. Each maintain their own set of:
  - Peers
  - Client Applications
  - Optionally Orderers
  - MSP

- **Test Network**:
  - All organizations in a single system
  - Development and testing purposes
  - 2 Orgs, each having 1 peer and optionally one CA
  - 1 orderer
  - All components are containerized

Monitor Containers

```
docker ps
```



```
 ~/fabric/fabric-samples/test-network   main  docker ps
CONTAINER ID   IMAGE                               COMMAND              CREATED         STATUS              PORTS
                                                                                                     NAMES
9f60342c20ac   hyperledger/fabric-tools:latest     "/bin/bash"          2 minutes ago   Up About a minute
                                                                                                     cli
9828aff6e8f1   hyperledger/fabric-peer:latest      "peer node start"    2 minutes ago   Up 2 minutes        0.0.0.0:7051->7051/tcp, :::7051-
>7051/tcp, 0.0.0.0:17051->17051/tcp, :::17051->17051/tcp                                             peer0.org1.example.com
a7cf98ab34c0   hyperledger/fabric-orderer:latest   "orderer"            2 minutes ago   Up 2 minutes        0.0.0.0:7050->7050/tcp, :::7050-
>7050/tcp, 0.0.0.0:7053->7053/tcp, :::7053->7053/tcp, 0.0.0.0:17050->17050/tcp, :::17050->17050/tcp   orderer.example.com
85eb38ae9ea4   hyperledger/fabric-peer:latest      "peer node start"    2 minutes ago   Up 2 minutes        0.0.0.0:9051->9051/tcp, :::9051-
>9051/tcp, 7051/tcp, 0.0.0.0:19051->19051/tcp, :::19051->19051/tcp                                   peer0.org2.example.com
```

- 2 fabric-peer containers, - 1 per organization.
- 1 fabric-orderer container
- 1 fabric-tools container

Channels in Test Network

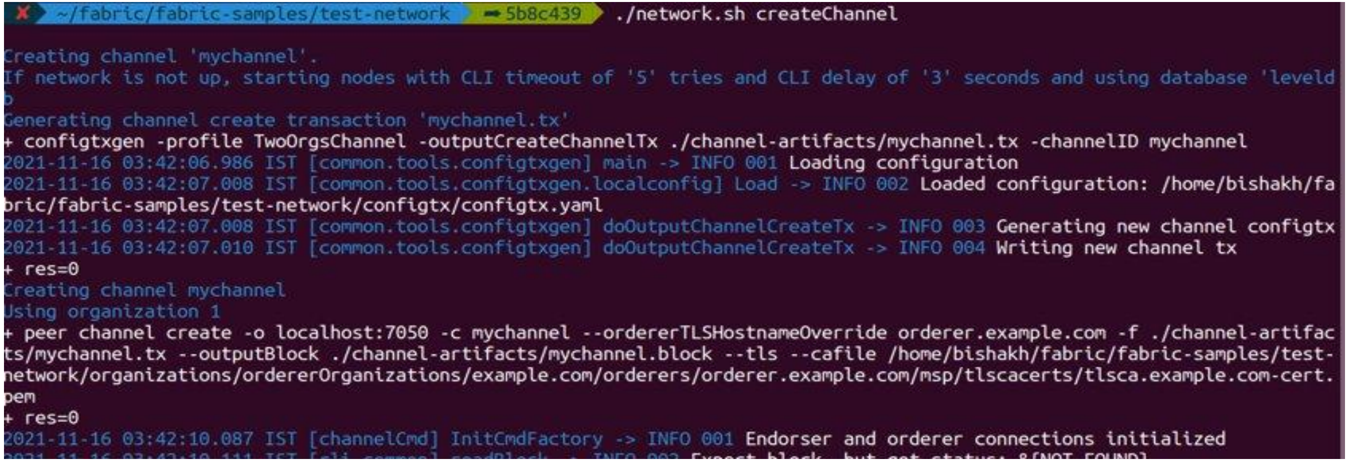
[Using the Fabric test network — hyperledger-fabricdocs master documentation]

- Channels are a private layer of communication between specific network members.
- Each channel has a separate blockchain ledger.
- To create a channel between Org1 and Org2 and join their peers to the channel, run
  - **./network.sh createChannel**
  - It will create a channel with the default name of '**mychannel**'

    ./network.sh createChannel -c <channel name>

Channels in Test Network

./network.sh createChannel



```
✘ ~/fabric/fabric-samples/test-network ➡ 5b8c439 ./network.sh createChannel

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 'leveld
b
Generating channel create transaction 'mychannel.tx'
+ configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/mychannel.tx -channelID mychannel
2021-11-16 03:42:06.986 IST [common.tools.configtxgen] main -> INFO 001 Loading configuration
2021-11-16 03:42:07.008 IST [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/bishakh/fa
bric/fabric-samples/test-network/configtx/configtx.yaml
2021-11-16 03:42:07.008 IST [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel configtx
2021-11-16 03:42:07.010 IST [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel tx
+ res=0
Creating channel mychannel
Using organization 1
+ peer channel create -o localhost:7050 -c mychannel --ordererTLSHostnameOverride orderer.example.com -f ./channel-artifac
ts/mychannel.tx --outputBlock ./channel-artifacts/mychannel.block --tls --cafile /home/bishakh/fabric/fabric-samples/test-
network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.
pem
+ res=0
2021-11-16 03:42:10.087 IST [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2021-11-16 03:42:10.111 IST [cli.common] readBlock -> INFO 002 Expect block, but got status: &{NOT_FOUND}
```

Deploying Chaincode

- Chaincode will be deployed on the channels to interact with the channel ledger.
- Chaincode contains the logic that governs the data on the ledger.
- Chaincode can be used for
  - Creating assets
  - Updating assets
  - Deleting assets
  - Transferring assets
  - Retrieving assets
- To start a chaincode on the channel, run
  - **./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go**
  - It will deploy a chaincode which is stored at **../asset-transfer-basic/chaincode-go**

https://hyperledger-fabric.readthedocs.io/en/release-2.4/test_network.html#starting-a-chaincode-on-the-channel

Setting CLI as Org1

To add binaries to your CLI Path, run

- **export PATH=${PWD}/../bin:$PATH**
- **export FABRIC_CFG_PATH=$PWD/../config/**

Set the environment variables that allow you to operate the CLI as 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**

https://hyperledger-fabric.readthedocs.io/en/release-2.4/test_network.html#interacting-with-the-network

Interacting with the Network

Using the Fabric test network — hyperledger-fabricdocs master documentation

- To invoke the “**InitLedger**” function of the deployed chaincode, run

  -
    ```
    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":[]}'
    ```
  - It will initialize the ledger with the entries there in the function.
- To query the content of the ledger, run

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

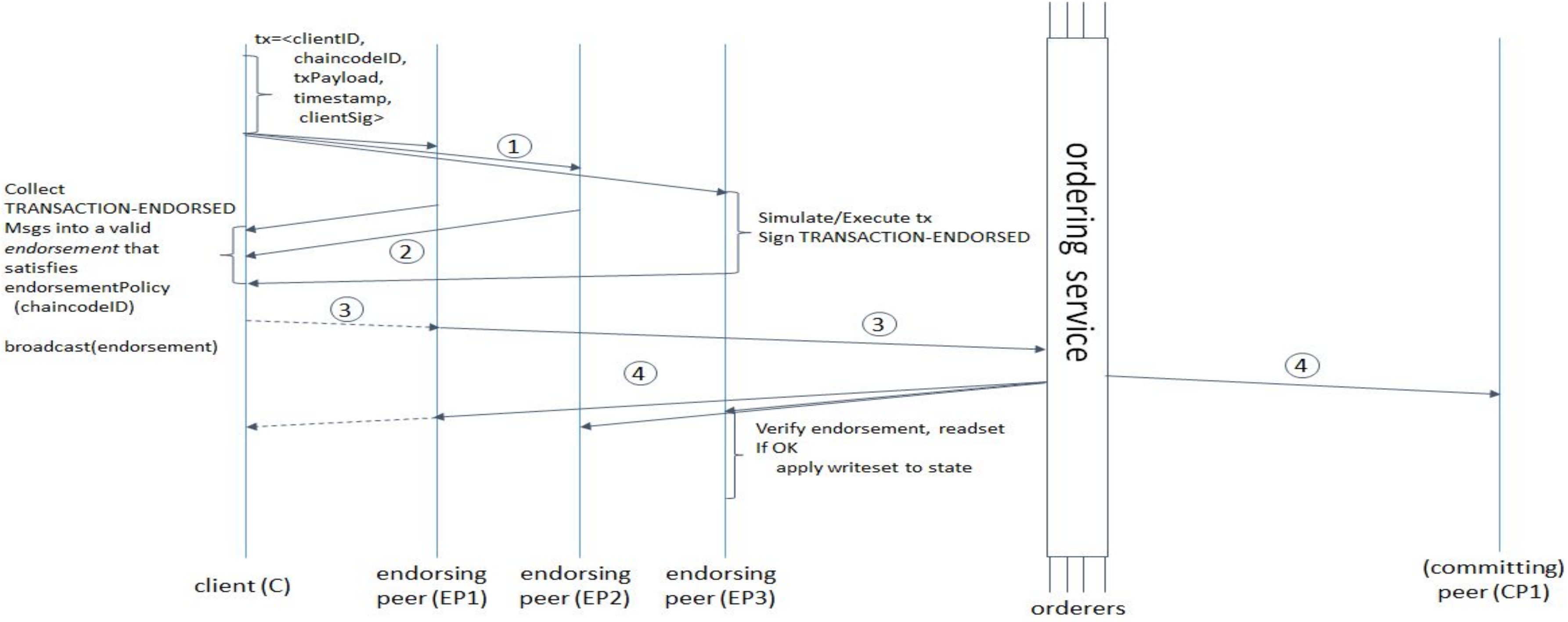
https://hyperledger-fabric.readthedocs.io/en/release-2.4/test_network.html#interacting-with-the-network

Invoke to change ledger state

```
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":"TransferAsset","Args":["asset6","Christopher"]}'
```

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

Transaction Flow



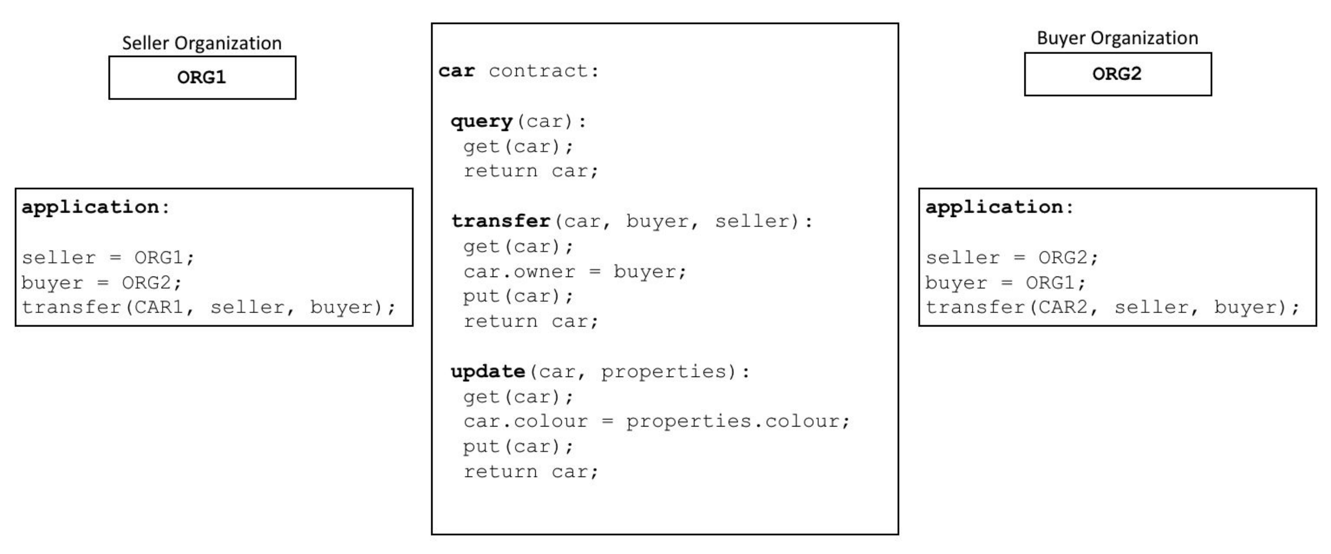
https://hyperledger-fabric.readthedocs.io/en/release-2.2/txflow.html

Developing Fabric Chaincodes

Using Go

Fabric Smart Contracts

- Defines common data, rules and processes for businesses to transact through the ledger.
- Smart contracts are packaged as **chaincodes**.



```
application:

seller = ORG1;
buyer = ORG2;
transfer(CAR1, seller, buyer);
```

```
car contract:

 query(car):
  get(car);
  return car;

 transfer(car, buyer, seller):
  get(car);
  car.owner = buyer;
  put(car);
  return car;

 update(car, properties):
  get(car);
  car.colour = properties.colour;
  put(car);
  return car;
```

```
application:

seller = ORG2;
buyer = ORG1;
transfer(CAR2, seller, buyer);
```

Writing Fabric Chaincode

- Written in Go, Node.js, or Java.

- Runs in a separate process from the peer.
  - separate container

- **fabric-contract-api** of Fabric SDK
  - contract interface, a high level API for implementing Chaincodes

  - Go: https://pkg.go.dev/github.com/hyperledger/fabric-contract-api-go/contractapi
  - Node.js: https://hyperledger.github.io/fabric-chaincode-node/release-2.2/api/
  - Java: https://hyperledger.github.io/fabric-chaincode-java/release-2.2/api/org/hyperledger/fabric/contract/package-summary.html

Writing Chaincode with Golang

- Fabric Contract API
  - Provides a high level API for application developers to implement Smart Contracts.
  - Each chaincode function has a transaction context "**ctx**" as an argument, from which you can to access the ledger (e.g. **GetState()**) and make requests to update the ledger (e.g. **PutState()**).
- **Steps**
  - Make a new Folder, go inside that folder
  - Create a **"go.mod"** file having a list of chaincode dependencies to be installed:

    ```
    go mod init <name>
    go get
    ```
  - Import necessary dependencies and define our Smart Contract
  - Create a structure to represent data on the ledger
  - Add functions in the contract to interact with the ledger
  - Run "**GO111MODULE=on go mod vendor"** to install dependencies into a vendor folder

https://hyperledger-fabric.readthedocs.io/en/release-2.4/chaincode4ade.html

Example Code Link

Define SmartContract

```
package main

import (
"fmt"
"github.com/hyperledger/fabric-contract-api-go/contractapi"
)


// SmartContract - provides functions for storing and
// retrieving keys and values from the world state
//
type SmartContract struct {
contractapi.Contract
}
```

InitLedger

```go
// InitLedger (optional in recent versions of fabric)
func (s *SmartContract) InitLedger(ctx
contractapi.TransactionContextInterface) error {
err := ctx.GetStub().PutState("testkey", []byte("testval"))

if err != nil {
return fmt.Errorf("Failed to put to world state. %s", err.Error())
}

return nil
}
```

GetState and PutState

```
// CreateKey
func (s *SmartContract) CreateKey(ctx
contractapi.TransactionContextInterface, key string, val string) error {
return ctx.GetStub().PutState(key, []byte(val))
}

// QueryKey
func (s *SmartContract) QueryKey(ctx contractapi.TransactionContextInterface,
key string) (string,error) {
val, err := ctx.GetStub().GetState(key)

if err != nil {
return "", fmt.Errorf("Failed to get from world state. %s", err.Error())
}

return string(val), nil
}
```

Start Chaincode

```
func main(){
chaincode, err := contractapi.NewChaincode(new(SmartContract))

if err != nil {
fmt.Printf("Error creating chaincode: %s", err.Error())
return
}

err = chaincode.Start();

if err != nil {
fmt.Printf("Error starting chaincode: %s", err.Error())
}

}
```

Install and invoke your chaincode

```
./network.sh deployCC -ccn <name of your chaincode> -ccp <relative path to chaincode>
-ccl go
```

```
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 samplechaincode --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":"CreateKey","Args":["x","2"]}'
```

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

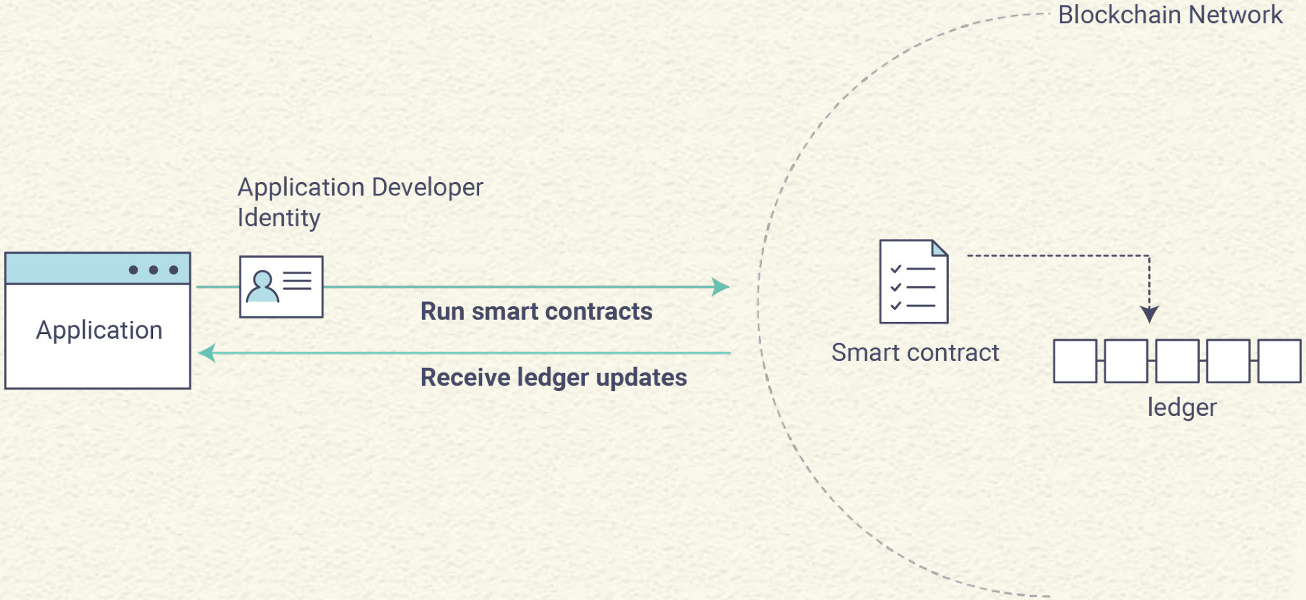
References

- HyperLedger Fabric Tutorial
- Smart contracts - Simply Explained
- Hyperledger Fabric : Overview
- Glossary — hyperledger-fabricdocs master documentation
- contractapi package - github.com/hyperledger/fabric-contract-api-go/contractapi

Fabric Applications

Using Javascript

Fabric Application



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

Prerequisites

- Fabric client applications can be developed in:
  - Node.js
  - Java
  - Go
  - Python

Make sure you start your network with CAs:

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

https://hyperledger-fabric.readthedocs.io/en/release-2.2/getting_started.html#hyperledger-fabric-application-sdks

Imports

```
const FabricCAServices = require('fabric-ca-client')
const {Wallets, Gateway} = require('fabric-network')

const fs = require('fs')
const path = require('path')


async function main(){
….
}

main()
```

https://hyperledger-fabric.readthedocs.io/en/release-2.2/getting_started.html#hyperledger-fabric-application-sdks

Connection Profile and CA

```
// Org1 connection profile
const ccpPath =
path.resolve('../organizations/peerOrganizations/org1.example.com/connection-org1.json')
const ccp = JSON.parse(fs.readFileSync(ccpPath, 'utf8'))

// Org1 Ca
const caInfo = ccp.certificateAuthorities['ca.org1.example.com']
const caTLSCACerts = caInfo.tlsCACerts.pem
const ca = new FabricCAServices(caInfo.url, { trustedRoots: caTLSCACerts,
verify: false }, caInfo.caName)
```

Connection Profile and CA

```
// Org1 connection profile
const ccpPath =
path.resolve('../organizations/peerOrganizations/org1.example.com/connection-org1.json')
const ccp = JSON.parse(fs.readFileSync(ccpPath, 'utf8'))

// Org1 Ca
const caInfo = ccp.certificateAuthorities['ca.org1.example.com']
const caTLSCACerts = caInfo.tlsCACerts.pem
const ca = new FabricCAServices(caInfo.url, { trustedRoots: caTLSCACerts,
verify: false }, caInfo.caName)
```

Configure CA admin

```
// Get admin identity

const enrollment = await ca.enroll({ enrollmentID: 'admin', enrollmentSecret:
'adminpw' });

const x509Identity = {
credentials: {
certificate: enrollment.certificate,
privateKey: enrollment.key.toBytes(),
},
mspId: 'Org1MSP',
type: 'X.509',
};

await wallet.put("admin", x509Identity)

console.log("Admin enrolled and saved into wallet successfully")

adminIdentity = await wallet.get("admin")
```

Register User

```
// Register user for this app
const provider = wallet.getProviderRegistry().getProvider(adminIdentity.type);
const adminUser = await provider.getUserContext(adminIdentity, 'admin');

const secret = await ca.register({affiliation: 'org1.department1', enrollmentID:
'appUser', role: 'client'}, adminUser);

const enrollment = await ca.enroll({enrollmentID: 'appUser',enrollmentSecret:
secret});

const x509Identity = {credentials: {certificate:
enrollment.certificate,privateKey: enrollment.key.toBytes()},
mspId: 'Org1MSP',
type: 'X.509',
};

await wallet.put('appUser', x509Identity)
console.log("Enrolled appUser and saved to wallet")

userIdentity = await wallet.get("appUser")
```

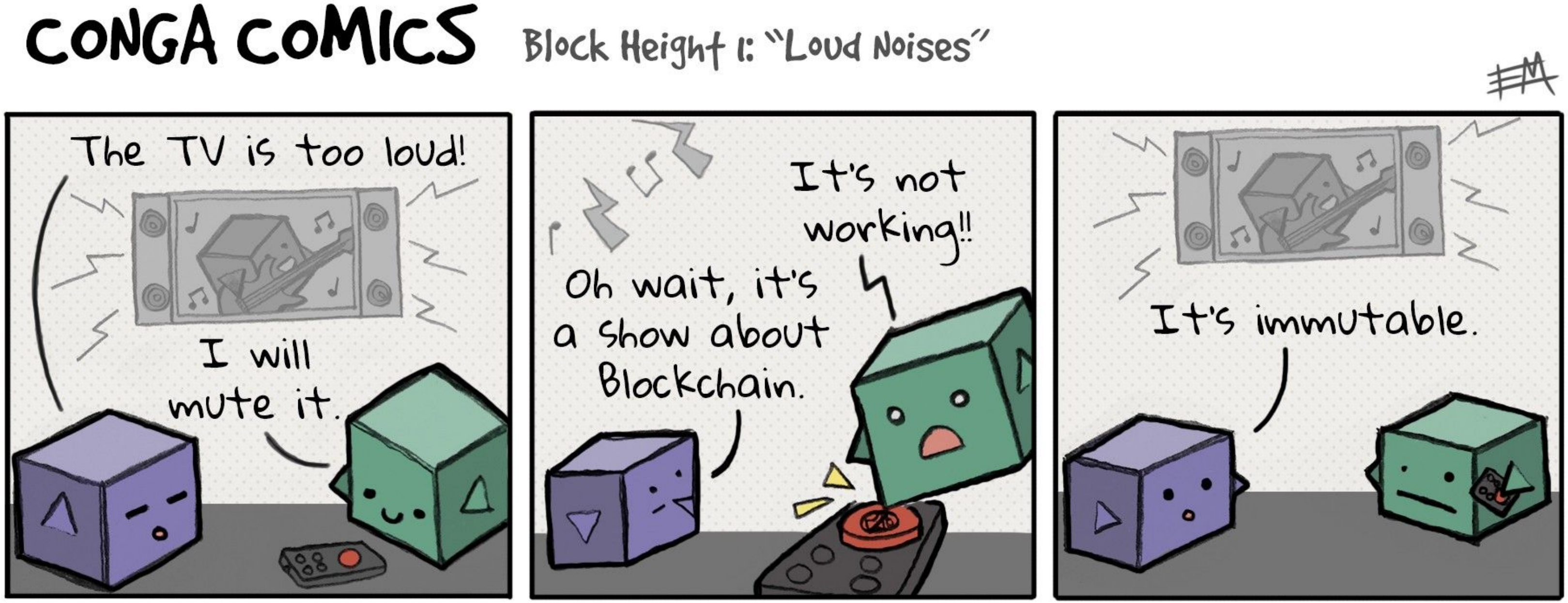
Configure Channel and Chaincode

```
// Connect to gateway
const gateway = new Gateway();

await gateway.connect(ccp, {wallet, identity:'appUser', discovery: {enabled:
true, asLocalhost: true}})

// connect to channel
const network = await gateway.getNetwork('mychannel')

// select the contract
const contract = network.getContract("keyvaluechaincode")
```

Query and Invoke Chaincodes

```
// Query and Invoke transactions

var result = await contract.evaluateTransaction("QueryKey", "nptel")
console.log("First query:", result.toString())

await contract.submitTransaction("CreateKey", "nptel", "a new value")

var result = await contract.evaluateTransaction("QueryKey", "nptel")
console.log("Second query:", result.toString())

// disconnect
await gateway.disconnect()
```

Thank You



https://medium.com/@edmoffat/know-any-good-blockchain-jokes-cea789b432b2