Hands-on session on Blockchain

Using  HYPERLEDGER FABRIC

Bishakh Chandra Ghosh

CNeRG

Permissioned Blockchains

For actual enterprise use, the following requirements must be satisfied by a blockchain platform:

- Participants must be **identifiable**
- Networks need to be **permissioned**
- **High** transaction **throughput** performance
- **Low latency** of transaction confirmation
- **Privacy** and **confidentiality** of transactions and data pertaining to business transactions

Hyperledger Fabric

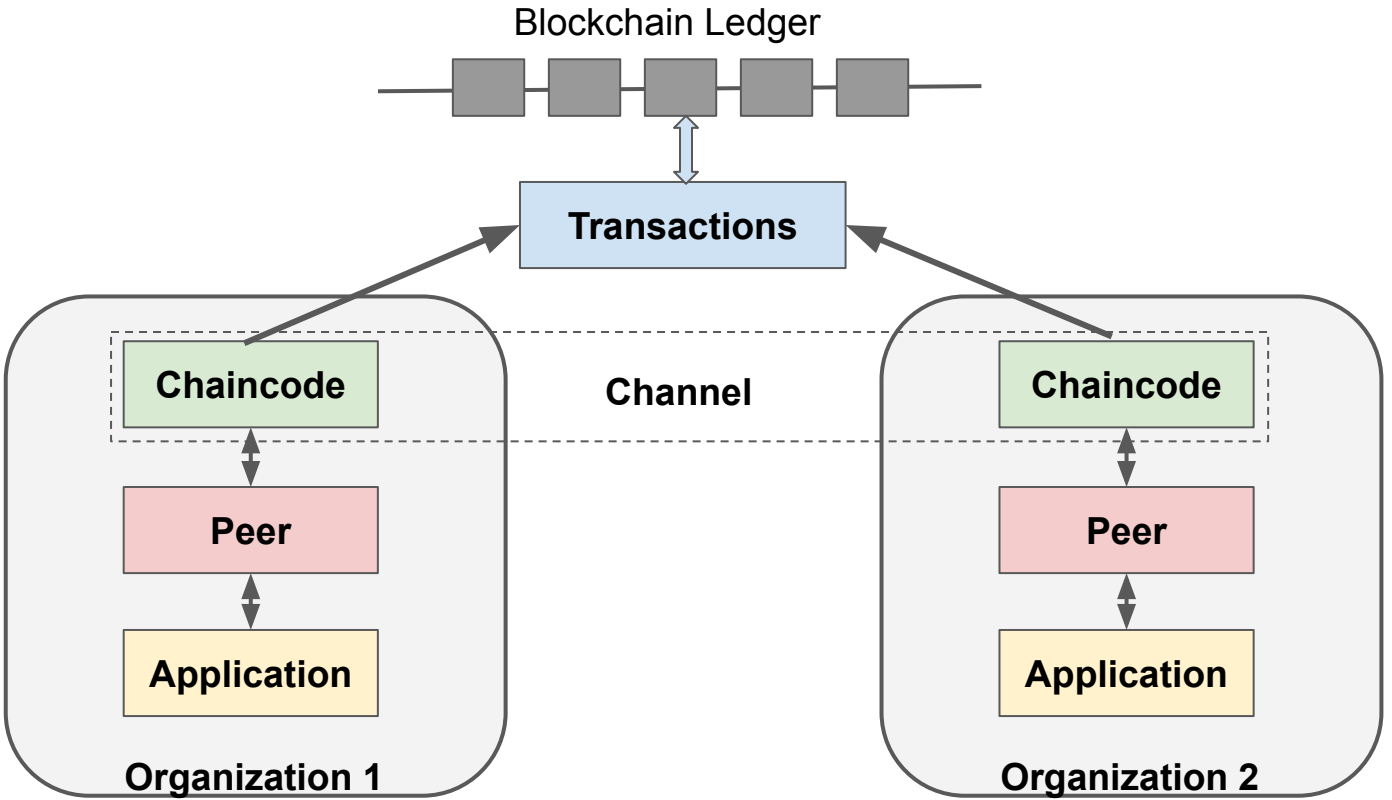
Hyperledger Fabric is an open source enterprise-grade **permissioned** distributed ledger technology (DLT) platform.

https://www.hyperledger.org/projects/fabric

Hyperledger Fabric Components

1. **Organizations:** The participants of the Blockchain network that form a *consortium* , “a group with a shared destiny”
2. **Peers:** A network entity that maintains a ledger and runs chaincode. Clients interact with the blockchain through the peers.
3. **Channels:** A channel is a primary communications mechanism between organizations. It allows for data isolation and confidentiality.
4. **Chaincode:** A smart contract, that manages access and modifications to a set of key-value pairs in the ***World State*** via ***Transactions***.

Hyperledger Fabric Components



Hyperledger Fabric Components

World State: The Hyperledger Fabric blockchain is similar to a database storing key-value pairs. The world state **represents the latest values** for all keys included in the chain transaction log.

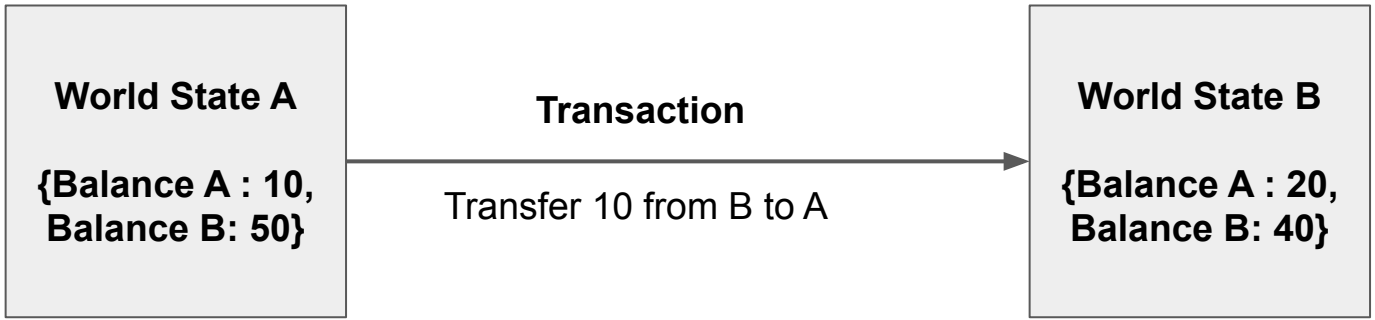
- World state provides direct access to the latest value of the keys
- Without it, the entire blockchain would have to be traversed again and again.
- Chaincodes execute transaction proposals against world state
- The world state will change every time the value of a key changes

Hyperledger Fabric Components

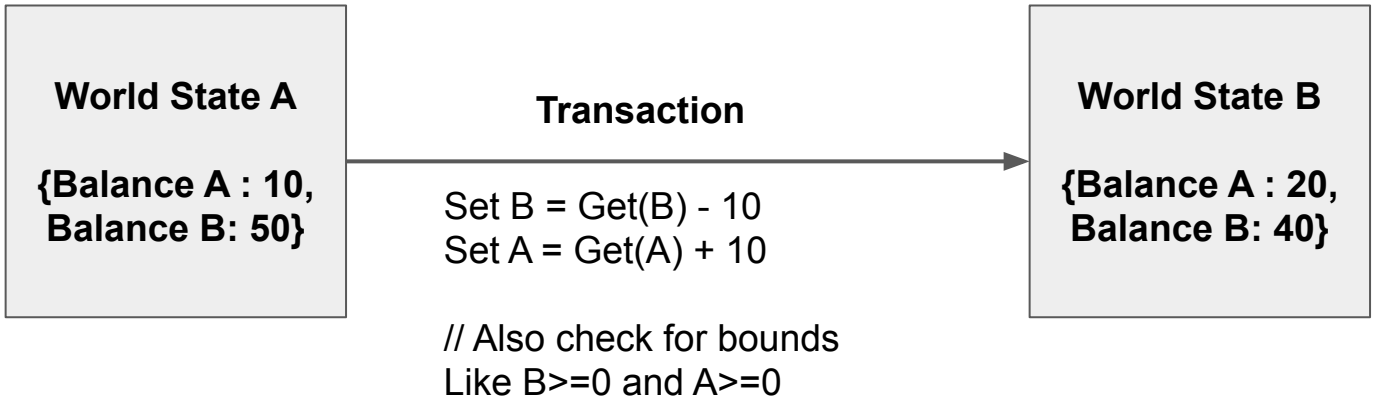
Transactions: Transactions are created when a chaincode is invoked from a client application to **read or write data from the ledger**.

- Transactions change the World State by changing the key-value pairs.
- Hyperledger Fabric uses ***execute-order-validate*** mode of transaction execution.
  - **execute** transactions, **check its correctness**, **endorse**
  - **order** transactions via a (pluggable) consensus protocol
  - **validate** transactions against an application-specific endorsement policy, **commit** to ledger.

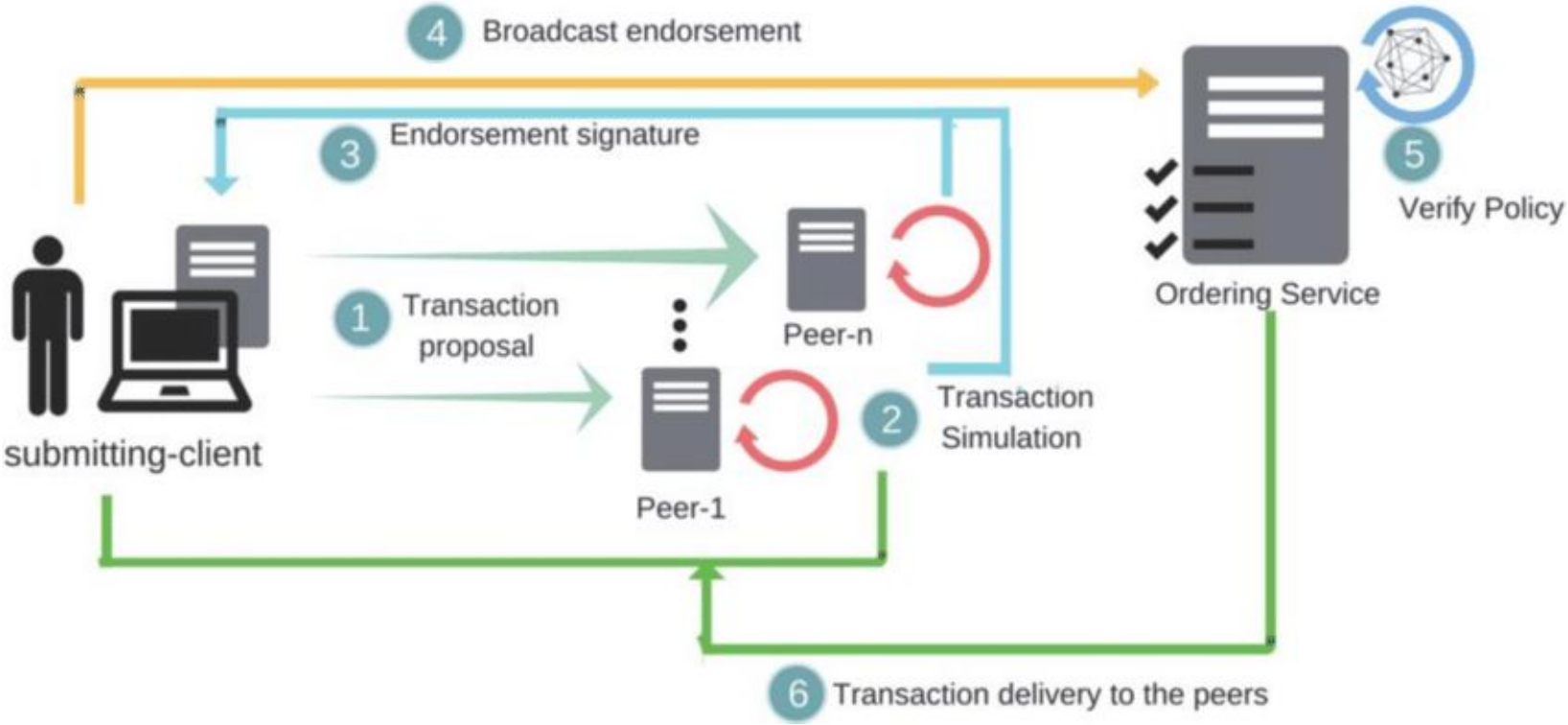
Hyperledger Fabric Components



Hyperledger Fabric Components



Transaction Flow



https://www.researchgate.net/publication/323973240_Enforcing_Human_Subject_Regulations_using_Blockchain_and_Smart_Contracts

Endorsement and Ordering

Endorsement: The process where specific peer nodes execute a chaincode transaction and return a proposal response.

The proposal response includes:

1. Output response message of chaincode

2. Results (read set and write set)

3. **Signature** to serve as **proof of the peer's chaincode execution**.

Endorsement and Ordering

Endorsement Policy: Specifies the required combination of endorsements to commit a transaction.

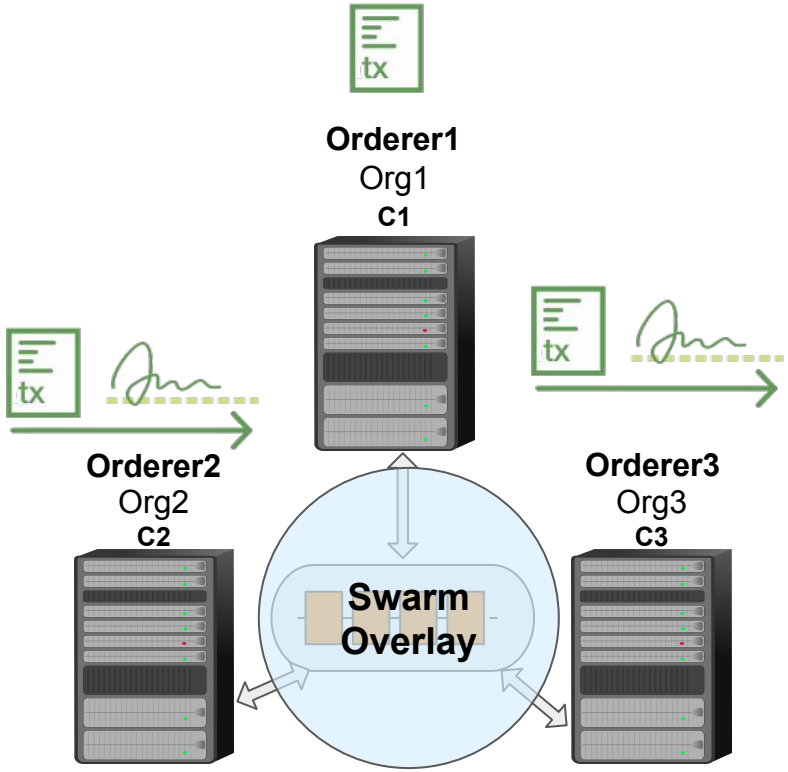
Eg: minimum number of endorsing peers, a minimum percentage of endorsing peers, all endorsing peers, etc..

Ordering: Orders transactions into a block and then distributes blocks to connected peers for validation and commit.

Orderer can use different pluggable BFT consensus protocols.

Our objective

- Write the simplest chaincode
- Test the chaincode in dev-mode
- Setup a multi organisation network using docker compose
- Test our chaincode in the network
- Deploy our network across different hosts



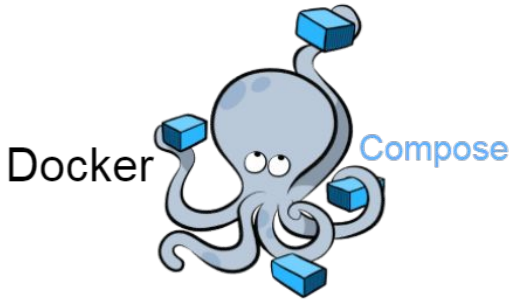
HYPERLEDGER
FABRIC

Other prerequisites

Containers: Containers are similar to virtual machines. They can can virtually package and isolate applications for deployment.

Docker: Docker is an open source software platform to create, deploy and manage virtualized application containers on a common operating system (OS).

Docker Compose: A tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services.



Setup Hyperledger Fabric

Step 1. Install Prerequisites:

Curl, Docker, Docker Compose, Go, NodeJS, NPM

```
sudo apt install curl docker docker-compose golang
```

```
sudo usermod -aG docker $USER
```

Step 2. Download binaries and samples:

This will clone a folder named **fabric-samples** in your current directory

```
curl -sSL http://bit.ly/2ysbOFE | bash -s 1.3.0
```

Setup Hyperledger Fabric

Step 3. Run Build your first network example [Link]

```
cd fabric-samples/first-network
./byfn.sh up
```

Stop the network:

```
./byfn.sh down
```

```
~/fabric-samples/first-network ➦ 94c21eb ./byfn.sh up
Starting for channel 'mychannel' with CLI timeout of '10' seconds and CLI delay of '3' seconds
Continue? [Y/n]
proceeding ...
LOCAL_VERSION=1.3.0
DOCKER_IMAGE_VERSION=1.3.0
Creating network "net_byfn" 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_peer1.org1.example.com" with default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating volume "net_peer1.org2.example.com" with default driver
Creating orderer.example.com    ... done
Creating peer0.org1.example.com ... done
Creating peer0.org2.example.com ... done
Creating peer1.org2.example.com ... done
Creating peer1.org1.example.com ... done
Creating cli                    ... done


 ____    _____      _      ____    _____
/ ___|  |_   _|    / \    |  _ \  |_   _|
\___ \    | |     / _ \   | |_) |   | |
 ___) |   | |    / ___ \  |  _ <    | |
|____/    |_|   /_/   \_\ |_| \_\   |_|

Build your first network (BYFN) end-to-end test

Channel name : mychannel
Creating channel...
+ peer channel create -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
+ res=0
+ set +x
2019-09-25 13:21:48.094 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2019-09-25 13:21:48.273 UTC [cli/common] readBlock -> INFO 002 Received block: 0
===================== Channel 'mychannel' created =====================

Having all peers join the channel...
+ peer channel join -b mychannel.block
+ res=0
+ set +x
2019-09-25 13:21:48.411 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
```

```
100
==================== Query successful on peer0.org1 on channel 'mychannel' ====================
Sending invoke transaction on peer0.org1 peer0.org2...
+ peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n mycc --peerAddresses peer0.org1.example.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --peerAddresses peer0.org2.example.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c '{"Args":["invoke","a","b","10"]}'
+ res=0
+ set +x
2019-09-25 13:23:07.576 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
==================== Invoke transaction successful on peer0.org1 peer0.org2 on channel 'mychannel' ====================

Installing chaincode on peer1.org2...
+ peer chaincode install -n mycc -v 1.0 -l golang -p github.com/chaincode/chaincode_example02/go/
+ res=0
+ set +x
2019-09-25 13:23:07.831 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2019-09-25 13:23:07.831 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2019-09-25 13:23:08.552 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
==================== Chaincode is installed on peer1.org2 ====================

Querying chaincode on peer1.org2...
==================== Querying on peer1.org2 on channel 'mychannel'... ====================
+ peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
Attempting to Query peer1.org2 ...3 secs
+ res=0
+ set +x

90
==================== Query successful on peer1.org2 on channel 'mychannel' ====================

========= All GOOD, BYFN execution completed ===========


 _____   _   _   ____
| ____| | \ | | |  _ \
|  _|   |  \| | | | | |
| |___  | |\  | | |_| |
|_____| |_| \_| |____/

~/fabric-samples/first-network ➡ 94c21eb
```

Download Sample Codes

Git Repo: https://github.com/ghoshbishakh/iiitgblockchain/

Or

ZIP: **http://bit.do/iiitg**

Chaincodes

A **smart contract**, is called “**chaincode**” in Fabric

It is the **business logic** of a blockchain application.

Functions as a trusted distributed application that gains its security/trust from the blockchain and the underlying consensus among the peers.

Developing Chaincodes

Can be written in *Go*, *NodeJS* and *Java*. **We will use Go**.

Setup Environment:

Get Hyperledger Fabric Chaincode Libraries

```
go get -u github.com/hyperledger/fabric/core/chaincode/shim
```

Create a chaincode file:

```
mkdir fabricdemo
```

```
cd fabricdemo
```

```
gedit iiitgchaincode.go
```

Developing Chaincodes

Import necessary dependencies and create a struct that implements the chaincode

```
package main

import (
    "fmt"

    "github.com/hyperledger/fabric/core/chaincode/shim"
    "github.com/hyperledger/fabric/protos/peer"
)

// SimpleAsset implements a simple chaincode to manage an asset
type SimpleAsset struct {
}
```

Developing Chaincodes

Every chaincode implements the ***'Chaincode'*** *interface* in particular, ***Init*** and ***Invoke*** functions.

Init is called during Instantiate transaction after the chaincode container has been established for the **first time**, allowing the chaincode to initialize its internal data.

Invoke is called to **update or query the ledger** in a proposal transaction. Updated state variables are not committed to the ledger until the transaction is committed.

Developing Chaincodes

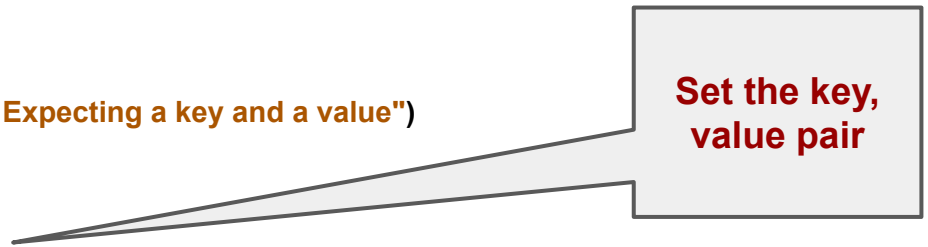
Implement ***Init*** function. - Takes input a **key** and a **value**

```
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {
    // Get the args from the transaction proposal
    args := stub.GetStringArgs()
    if len(args) != 2 {
        return shim.Error("Incorrect arguments. Expecting a key and a value")
    }
    // Store the key and the value on the ledger
    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))
    }
    return shim.Success(nil)
}
```

Developing Chaincodes

Implement ***Init*** function. - Takes input a **key** and a **value**

```
func (t *SimpleAsset) Init(stub shim.ChaincodeStubInterface) peer.Response {
    // Get the args from the transaction proposal
    args := stub.GetStringArgs()
    if len(args) != 2 {
        return shim.Error("Incorrect arguments. Expecting a key and a value")
    }
    // Store the key and the value on the ledger
    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        return shim.Error(fmt.Sprintf("Failed to create asset: %s", args[0]))
    }
    return shim.Success(nil)
}
```



Developing Chaincodes

Invoke function.

```
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
    // Extract the function and args from the transaction proposal
    fn, args := stub.GetFunctionAndParameters()

    if fn == "set" {

            // set the value

    } else {
             // assume 'get' even if fn is nil
            // get the value
    }

}
```

Developing Chaincodes

Invoke function.

```
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
    // Extract the function and args from the transaction proposal
    fn, args := stub.GetFunctionAndParameters()

    if fn == "set" {

            // set the value

    } else {
             // assume 'get' even if fn is nil
            // get the value
    }

}
```

The invocation function name

Developing Chaincodes

Set value method

```
// Set stores the asset (both key and value) on the ledger. If the key exists,
// it will override the value with the new one
func set(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 2 {
        return "", fmt.Errorf("Incorrect arguments. Expecting a key and a value")
    }

    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        return "", fmt.Errorf("Failed to set asset: %s", args[0])
    }
    return args[1], nil
}
```

Developing Chaincodes

Get value method

```
// Get returns the value of the specified asset key
func get(stub shim.ChaincodeStubInterface, args []string) (string, error) {
    if len(args) != 1 {
        return "", fmt.Errorf("Incorrect arguments. Expecting a key")
    }

    value, err := stub.GetState(args[0])
    if err != nil {
        return "", fmt.Errorf("Failed to get asset: %s with error: %s", args[0], err)
    }
    if value == nil {
        return "", fmt.Errorf("Asset not found: %s", args[0])
    }
    return string(value), nil
}
```

Developing Chaincodes

Final ***Invoke*** function.

```
func (t *SimpleAsset) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
    // Extract the function and args from the transaction proposal
    fn, args := stub.GetFunctionAndParameters()

    if fn == "set" {
            result, err = set(stub, args)
    } else { // assume 'get' even if fn is nil
            result, err = get(stub, args)
    }
    if err != nil {
            return shim.Error(err.Error())
    }

    // Return the result as success payload
    return shim.Success([]byte(result))
}
```

Testing the Chaincode

Terminal 1:

Copy chaincode source file to *fabric-samples/chaincode/myChaincode/*

```
cd fabric-samples/chaincode-docker-devmode
docker-compose -f docker-compose-simple.yaml up
```

Terminal2:

```
docker exec -it chaincode bash
cd myChaincode
go build
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:0 ./myChaincode
```

Testing the Chaincode

Terminal 3:

```
docker exec -it cli bash
```

Install Chaincode:

```
peer chaincode install -p chaincodedev/chaincode/myChaincode -n mycc -v 0
```

Instantiate Chaincode:

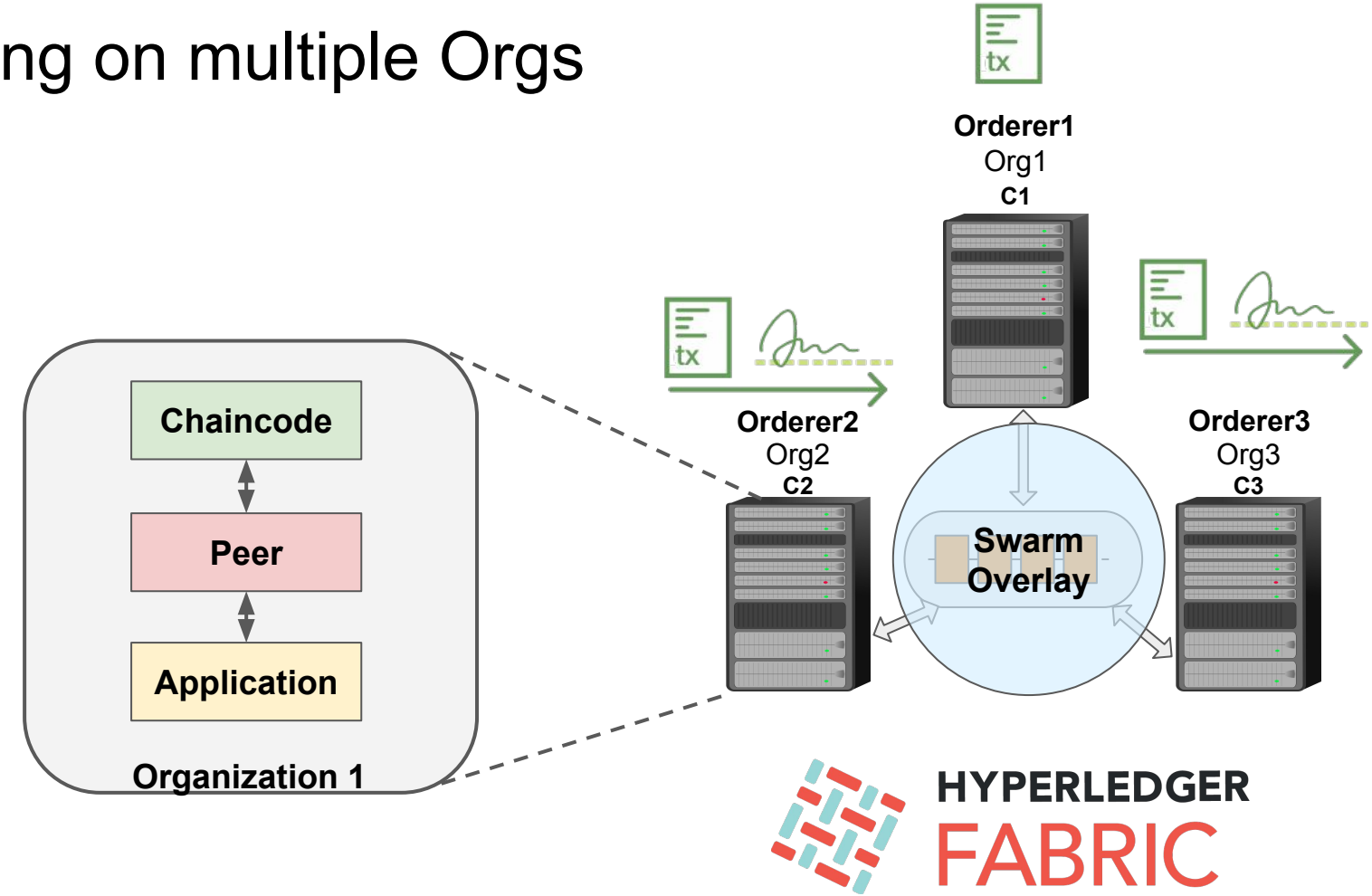
```
peer chaincode instantiate -n mycc -v 0 -c '{"Args":["a","10"]}' -C myc
```

Invoke and query chaincode:

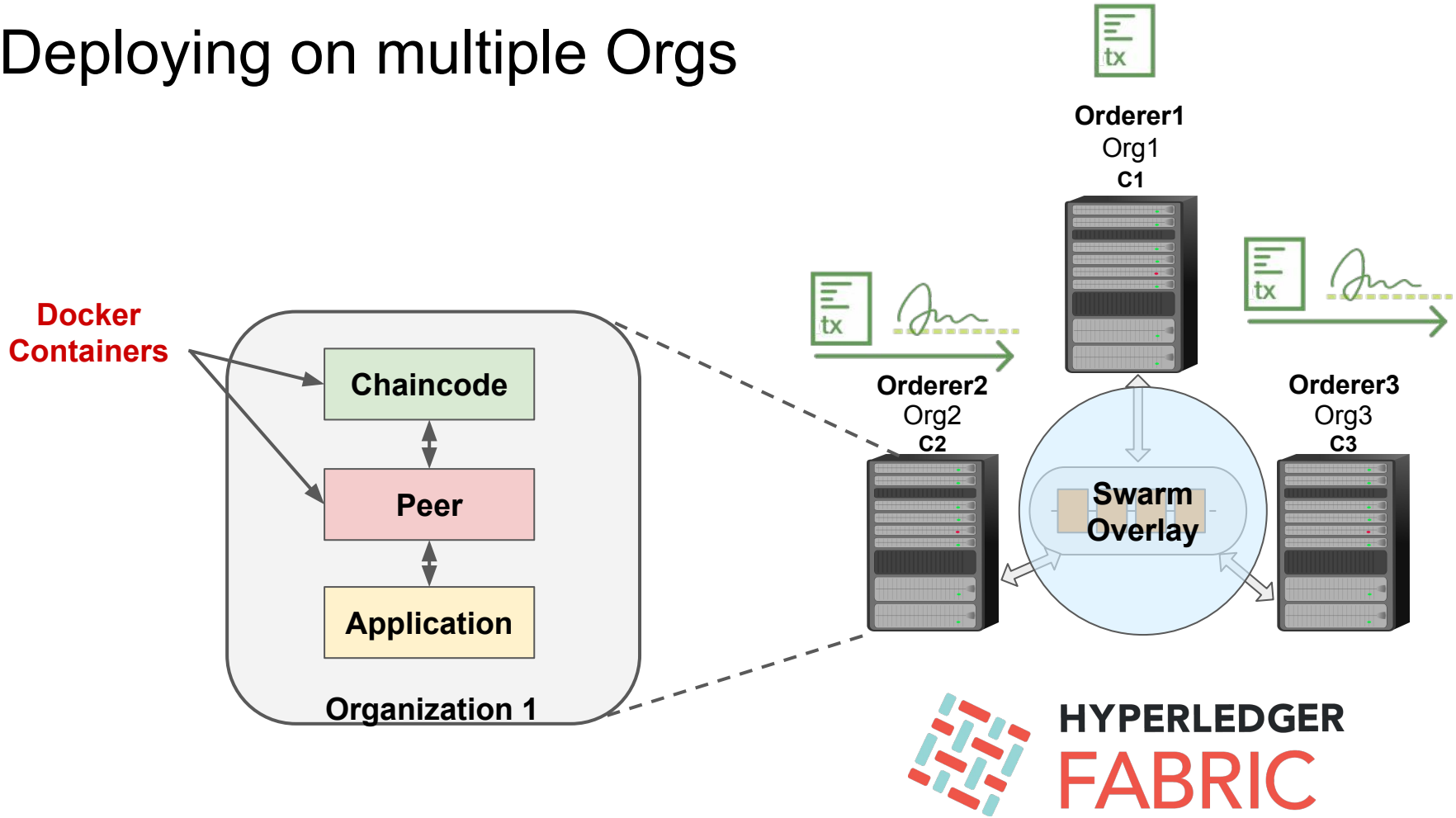
```
peer chaincode invoke -n mycc -c '{"Args":["set", "a", "20"]}' -C myc
peer chaincode query -n mycc -c '{"Args":["query","a"]}' -C myc
```

```
ng balancer to "pick_first"
2019-09-27 19:16:10.886 UTC [grpc] HandleSubConnStateChange -> DEBU 03a pickfirs
tBalancer: HandleSubConnStateChange: 0xc4204fd630, CONNECTING
2019-09-27 19:16:10.887 UTC [grpc] HandleSubConnStateChange -> DEBU 03b pickfirs
tBalancer: HandleSubConnStateChange: 0xc4204fd630, READY
2019-09-27 19:16:10.887 UTC [grpc] DialContext -> DEBU 03c parsed scheme: ""
2019-09-27 19:16:10.887 UTC [grpc] DialContext -> DEBU 03d scheme "" not registe
red, fallback to default scheme
2019-09-27 19:16:10.887 UTC [grpc] watcher -> DEBU 03e ccResolverWrapper: sendin
g new addresses to cc: [{peer:7051 0  <nil>}]
2019-09-27 19:16:10.887 UTC [grpc] switchBalancer -> DEBU 03f ClientConn switchi
ng balancer to "pick_first"
2019-09-27 19:16:10.888 UTC [grpc] HandleSubConnStateChange -> DEBU 040 pickfirs
tBalancer: HandleSubConnStateChange: 0xc4203501a0, CONNECTING
2019-09-27 19:16:10.888 UTC [grpc] HandleSubConnStateChange -> DEBU 041 pickfirs
tBalancer: HandleSubConnStateChange: 0xc4203501a0, READY
2019-09-27 19:16:10.888 UTC [msp] GetDefaultSigningIdentity -> DEBU 042 Obtainin
g default signing identity
2019-09-27 19:16:10.889 UTC [msp/identity] Sign -> DEBU 043 Sign: plaintext: 0AC
9070A6108031A0C08FABFB9EC0510...6D7963631A0A0A0571756572790A0161
2019-09-27 19:16:10.889 UTC [msp/identity] Sign -> DEBU 044 Sign: digest: 97EA36
7CA85569698907BDC566086A8A84ABF06AF3248E392099D4FF42B0290D
20
root@f777ec16a788:/opt/gopath/src/chaincodedev#
```

Deploying on multiple Orgs



Deploying on multiple Orgs



Deploying on multiple Orgs

Add Fabric binaries to executable path

Add “**PATH=$PATH:/home/iiitg/fabric-samples/bin**” To **~/.bashrc**

We will use the **cryptogen** tool to generate the cryptographic material (x509 certs and signing keys) for our various network entities. These certificates are representative of identities, and they allow for sign/verify authentication to take place as our entities communicate and transact.

The **configtxgen** tool is used to create four configuration artifacts:

- orderer genesis block,
- channel configuration transaction,
- and two anchor peer transactions - one for each Peer Org.

Deploying on multiple Orgs

Configure ***crypto-config.yml****:*

Specify the number of organizations, peers and users (applications).

Generate cryptographic assets like key pairs and certificates

```
cryptogen generate --config=crypto-config.yaml
```

The certs and keys (i.e. the MSP material) will be output into a directory - *crypto-config*

Deploying on multiple Orgs

Next, we need to generate the following:

1. Genesis Block
2. Channel Configuration Transactions
3. Anchor peers (In case of multiple peers per org)

Using ***configtxgen***

To configure the architecture, we define it in the the *configtx.yaml* file which is used by configtxgen to generate the above artifacts.

Deploying on multiple Orgs

Add configuration profiles in **configtx.yaml**

```
Profiles:
    ThreeOrgsOrdererGenesis:
        <<: *ChannelDefaults
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *Org1
                    - *Org2
                    - *Org3
    ThreeOrgsChannel:
        Consortium: SampleConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Org1
                - *Org2
                - *Org3
            Capabilities:
                <<: *ApplicationCapabilities
```

Deploying on multiple Orgs

Generate Genesis Block

```
mkdir channel-artifacts

export FABRIC_CFG_PATH=$PWD

configtxgen -profile ThreeOrgsOrdererGenesis -outputBlock
./channel-artifacts/genesis.block
```

Deploying on multiple Orgs

Create Channel Artifacts

```
export CHANNEL_NAME=mychannel
```

```
configtxgen -profile ThreeOrgsChannel -outputCreateChannelTx
./channel-artifacts/channel.tx -channelID $CHANNEL_NAME
```

Deploying on multiple Orgs

Configure anchor peers:

```
configtxgen -profile ThreeOrgsChannel -outputAnchorPeersUpdate
./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org1MSP

configtxgen -profile ThreeOrgsChannel -outputAnchorPeersUpdate
./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org2MSP

configtxgen -profile ThreeOrgsChannel -outputAnchorPeersUpdate
./channel-artifacts/Org3MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org3MSP
```

Deploying on multiple Orgs

Create a docker swarm (Will be helpful for our next step):

```
docker swarm init
```

On other hosts join it to the swarm using the command outputted with the last command.

Create an overlay network

Create network (on master node):

```
docker network create --attachable --driver overlay cloudExNet
```

Deploying on multiple Orgs

Configure the docker-compose files:

- Define three Orgs
- Each org will have two *peers*
- Each org will have one CA
- One CLI to access the peers

Main trick is to configure volumes and mount proper CA keys and certificates.

Deploying on multiple Orgs

Run the containers

[Copy the myChaincode and scripts directory first]

Use docker stack deploy with proper environment variables:

```
docker stack deploy -c docker-compose.yaml iiitg
```

Or

simply use the script:

```
./start-sh.sh
```

Deploying on multiple Orgs

Channel Setup:

Create channel:

```
peer channel create -o orderer.example.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/channel.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
```

Join peer into a channel:

```
peer channel join -b mychannel.block
```

Write a script for setting up channel for all users. (*peerChannelSetupScript.sh*)

Deploying on multiple Orgs

Chaincode Setup:

Install Chaincode:

```
peer chaincode install -n $CHAINCODE_NAME -v 1.0 -p github.com/chaincode/src/
```

Instantiate Chaincode:

```
peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/ord
erer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL_NAME -n
$CHAINCODE_NAME -v 1.0 -c '{"Args":["A","20"]}' -P "AND
('Org1MSP.peer','Org2MSP.peer','Org3MSP.peer')"
```

Write a script for setting up chaincodes (*peerChaincodeSetupScript.sh*)

Deploying on multiple Orgs

Chaincode Setup:

Install Chaincode:

```
peer chaincode install -n $CHAINCODE_NAME -v 1.0 -p github.com/chaincode/src/
```

Instantiate Chaincode:

```
peer chaincode instantiate -o orderer.example.com:7
/opt/gopath/src/github.com/hyperledger/fabric/peer/c
erer.example.com/msp/tlscacerts/tlsca.example.com
$CHAINCODE_NAME -v 1.0 -c '{"Args":["A","20"]}' -
('Org1MSP.peer','Org2MSP.peer','Org3MSP.peer')"
```

Endorsement Policies
How each transaction must be signed

Write a script for setting up chaincodes (*peerChaincodeSetupScript.sh*)

Deploying on multiple Orgs

Run the scripts for channel and chaincode setup

```
docker exec -it cli /bin/bash
```

```
sh scripts/peerChannelSetupScript.sh
```

```
sh scripts/peerChaincodeSetupScript.sh mycc1
```

Testing on multiple Orgs

```
peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.exam
ple.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n mycc1 --peerAddresses
peer0.org1.example.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.ex
ample.com/tls/ca.crt --peerAddresses peer0.org2.example.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.ex
ample.com/tls/ca.crt --peerAddresses peer0.org3.example.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.example.com/peers/peer0.org3.ex
ample.com/tls/ca.crt -c '{"Args":["set","b","10"]}'
```

Deploying on multiple hosts

Since all the materials have already been generated, we need to copy the same to different hosts.

We need to copy three things:

channel-artifacts

crypto-config

chaincode

In addition:

scripts

Deploying on multiple hosts

scp -r ./channel-artifacts REMOTE:/home/iiitg/fabricdemo

scp -r ./crypto-config REMOTE:/home/iiitg/fabricdemo

scp -r ./myChaincode REMOTE:/home/iiitg/fabricdemo

scp -r ./scripts REMOTE:/home/iiitg/fabricdemo

For simplicity, run ./distributesetup.sh

Deploying on multiple hosts

Add placement constraints to the docker compose files.

```
deploy:
  replicas: 1
  restart_policy:
    condition: on-failure
  placement:
    constraints:
      - node.hostname == HOSTNAME
```

Deploying on multiple hosts

sh start.sh

Deploying on multiple hosts

```
docker exec -it cli /bin/bash
```

```
sh scripts/peerChannelSetupScript.sh
```

```
sh scripts/peerChannelSetupScript.sh
```

Deploying on multiple hosts

```
peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.exam
ple.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n mycc --peerAddresses
peer0.org1.example.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.ex
ample.com/tls/ca.crt --peerAddresses peer0.org2.example.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.ex
ample.com/tls/ca.crt --peerAddresses peer0.org3.example.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.example.com/peers/peer0.org3.ex
ample.com/tls/ca.crt -c '{"Args":["set","b","10"]}'
```

Important References

In depth architecture of Fabric: https://hyperledger-fabric.readthedocs.io/en/latest/arch-deep-dive.html

Transaction Flow: https://hyperledger-fabric.readthedocs.io/en/latest/txflow.html

Build Your First Network: https://hyperledger-fabric.readthedocs.io/en/latest/build_network.html

Chaincode Development: https://hyperledger-fabric.readthedocs.io/en/latest/chaincode4ade.html

Key Level Endorsement Policies:
https://hyperledger-fabric.readthedocs.io/en/release-1.4/endorsement-policies.html#setting-key-level-endorsement-policies

Thank You