Table of Contents

CHAPTER 1: INTRODUCTION TO HYPERLEDGER FABRIC

Blockchain

Block chain is a continuously growing list of digital records called blocks, which are linked and secured using cryptography. Each block typically contains a cryptographic hash of the previous block, a timestamp and transaction data.

A blockchain is inherently resistant to modification of the data. It is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way.

Definition from wiki…

Hyperledger

Hyperledger is an open source and openly governed collaborative effort to advance cross-industry blockchain technologies for business, hosted by The Linux Foundation.

Hyperledger-Fabric

Hyperledger Fabric is a blockchain framework implementation and one of the Hyperledger projects, intended as a foundation for developing applications/solutions with a modular architecture

Hyperledger-Composer

Hyperledger Composer is a set of collaboration tools for building blockchain business networks that make it simple and fast for business owners and developers to create smart contracts and blockchain applications to solve business problems.

Increases understanding

Bridges simply from business concepts to blockchain

Saves time

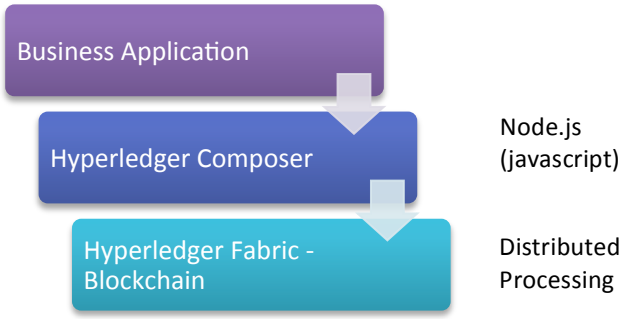
Develop blockchain applications more quickly and cheaply

Reduces risk

Well tested, efficient design conforms to best practice

Increases flexibility

Higher level abstraction makes it easier to iterate



Components in a Blockchain Solution

Ledger

A ledger is a channel's chain and current state data which is maintained by each peer on the channel.

Chaincode / Smart Contract

Software running on a ledger, to encode assets and the transaction instructions (business logic) for modifying the assets

Peer Network

A broader term overarching the entire transactional flow, which serves to generate an agreement on the order and to confirm the correctness of the set of transactions constituting a block

Membership

Membership Services authenticates, authorizes, and manages identities on a permissioned blockchain network

Events

Creates notifications of significant operations on the blockchain (e.g. a new block), as well as notifications related to smart contracts.

Systems Management

Provides the ability to create, change and monitor blockchain components

Wallet

Securely manages a user's security credentials

Systems Integration

Responsible for integrating Blockchain bi-directionally with external systems. Not part of blockchain, but used with it

Public vs. Private Blockchains

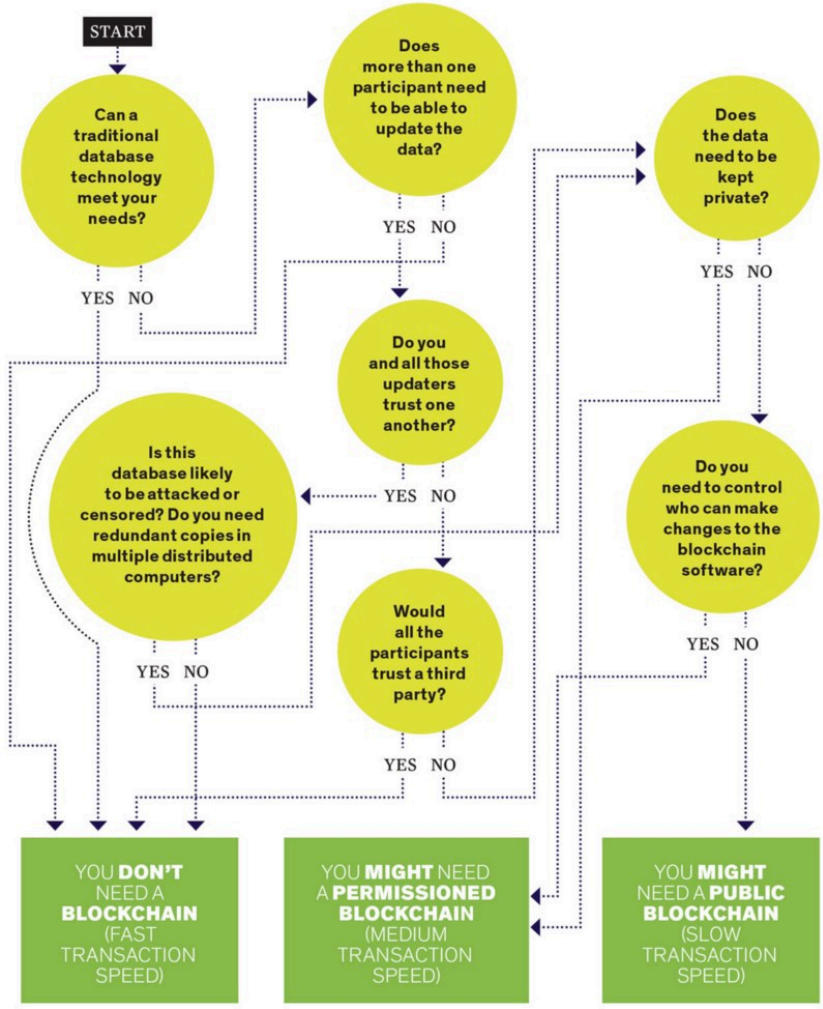
Public Blockchain

- Transactions are viewable by anyone
- Participant identity is more difficult to control
- For example: Bitcoin, Ethereum

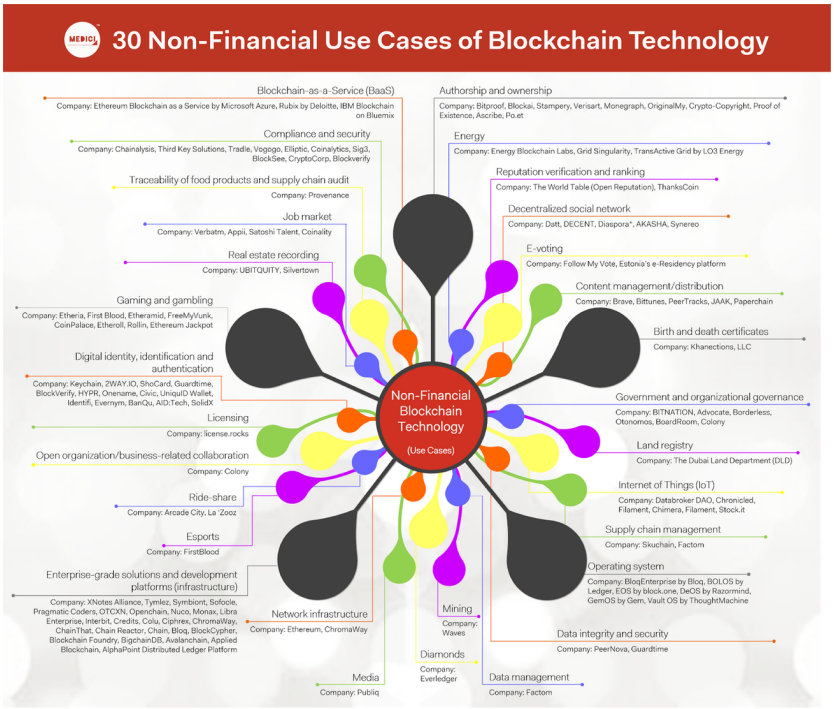
Private Blockchain

- Network members are known but transactions are secret
- Needs control who make the change to blockchain registry
- For example; Hyperledger Fabric

How to decide which blockchain to use?



Few Blockchain Usecases:



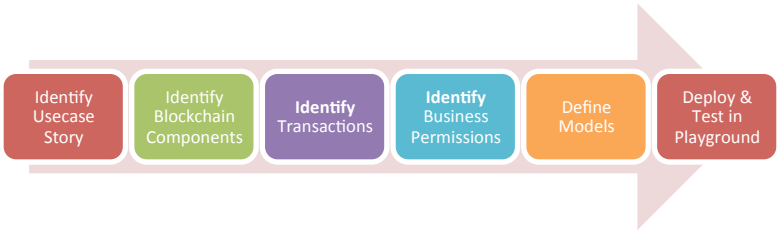
HyperLeger Fabric

AIM

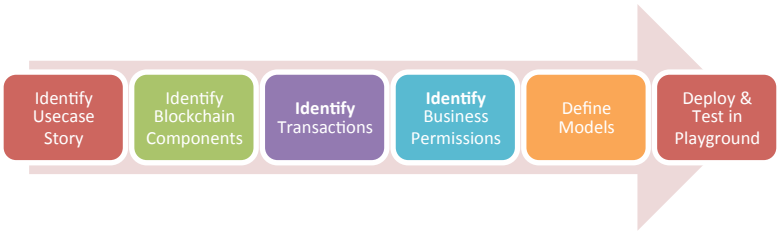
After you completed all the following chapters with lab exercises, you will be able to:

- Have Development Environment Setup for Hyperledger Fabric
- Understand on types of nodes & their functions
- Create permissioned blockchain network
- Use Authorizations and authentication
- Write chaincode and call transactions via composer tools and rest server
- Understand different Hyperledger architecture templates

Lab Exercise 1: Building a UseCase

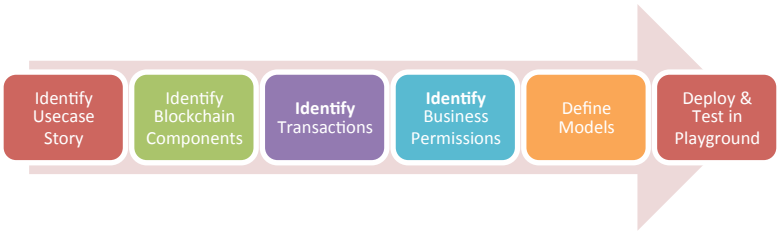


1. **Identify Usecase Story**
2. **Identify Blockchain Components**
3. **Identify Transactions**
4. **Identify Business Permissions**
5. **Define Models**
6. **Deploy & Test in Playground**

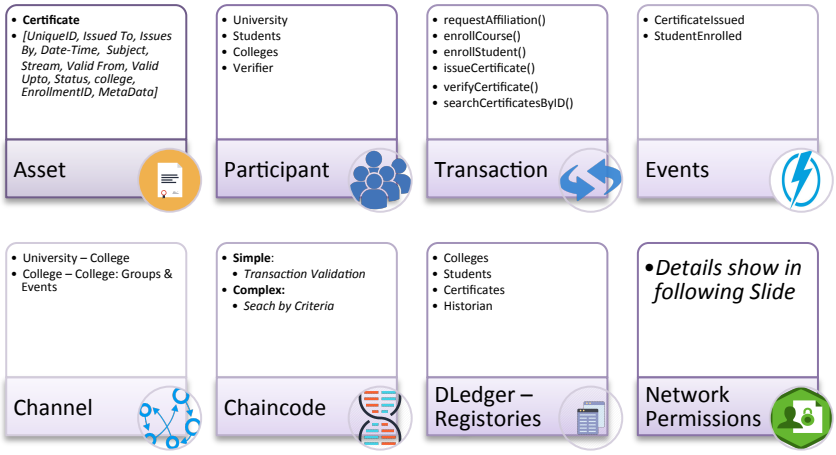


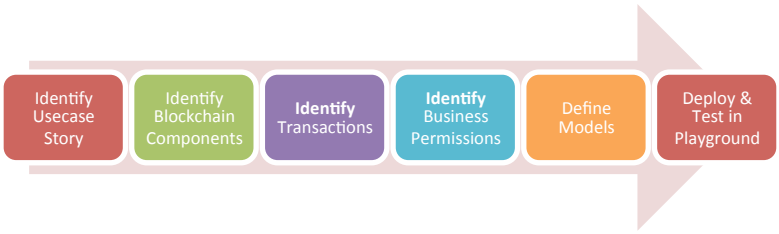
TASK#1: Private Blockchain Usecase & Stories

- **University**: As a University, I am an intitution for higher studies. I want to affiliate Colleges and cources. Enroll Students and Award certificates
- **Students**: As a student I want to take admissiom, enroll for a specific education program and get awarded with certification of successful completion
- **Colleges**: As a college, I provide students specialized courses as per the guidelines/affiliation with university.
- **Verifier**: As a verifier, I want to verify if the certificate ID (public address) is valid and its corressponding details (Issued to, Status, etc). I can also search all certificates assocaited with a Nation-ID (or tokenized ID)
- **Business Network Admin (BNA)**: As a network admin I want to deploy code and runtimes.



TASK#2: Identify Blockchain Components





TASK#3: Identify Transactions

Step 1:

requestAffiliation()
Input:
- College Detail
Output: *Status*
By Praticipant: *college*
On Praticipant: *University*

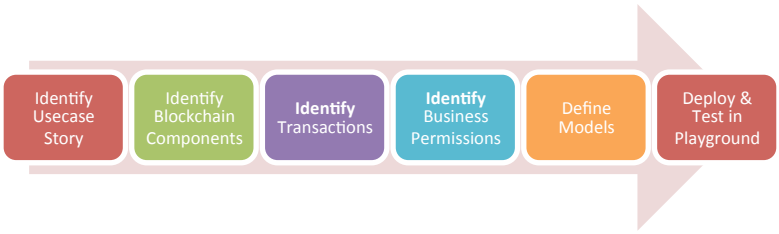
searchCertificateByID()
Input:
- National-ID / Enrollment-ID
Output: Certificate *List*
By Praticipant: *Verifier*
On Praticipant: *Blockchain*

enrollCourse()
Input:
- Course Detail
Output: *CourseID*
By Praticipant: *College*
On Praticipant: *University*

issueCertificate()
Input:
- Certificate Detail
Output: *Public Address*
By Praticipant: *College*
On Praticipant: *University*

enrollStudent()
Input:
- Student Detail
Output: *Student EnrollmentID*
By Praticipant: *College*
On Praticipant: *University*

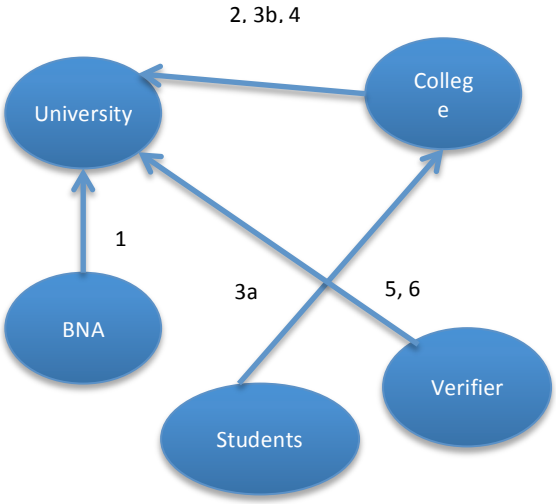
renewCertificate()
Input:
- Certificate Detail
Output: *Status-Success/fail*
By Praticipant: *College*
On Praticipant: *University*

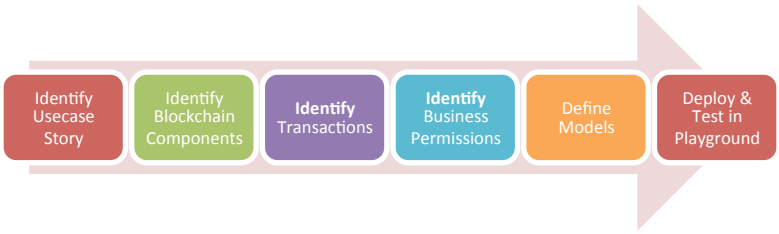


verifyCertificate()
Input: *Public Address*
Output: *Certifcate Details*
By Praticipant: *Verifier (anyone)*

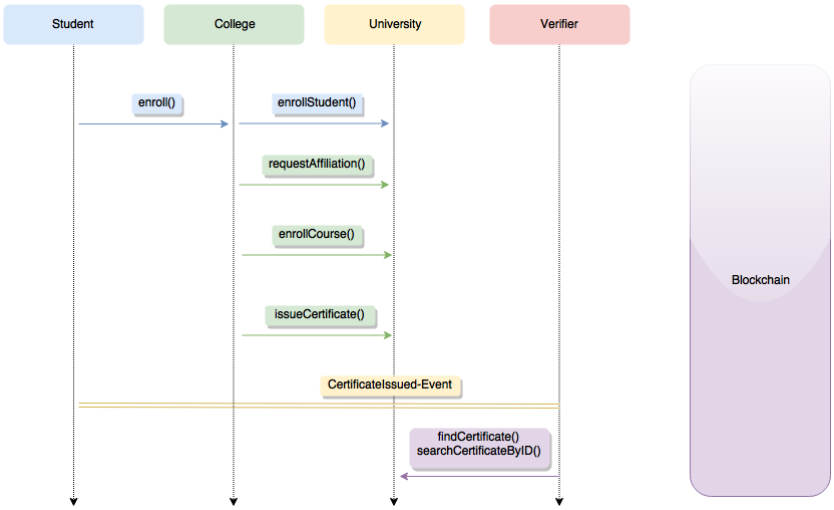
Step #3:

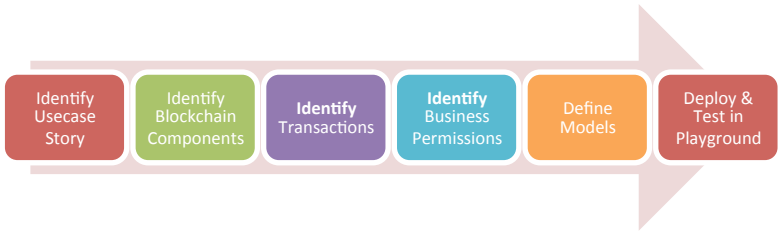
1. requestAffiliation()
2. enrollCourse()
3. enrollStudent()
4. issueCertificate()
5. verifyCertificate()
6. searchCertificatesByID()



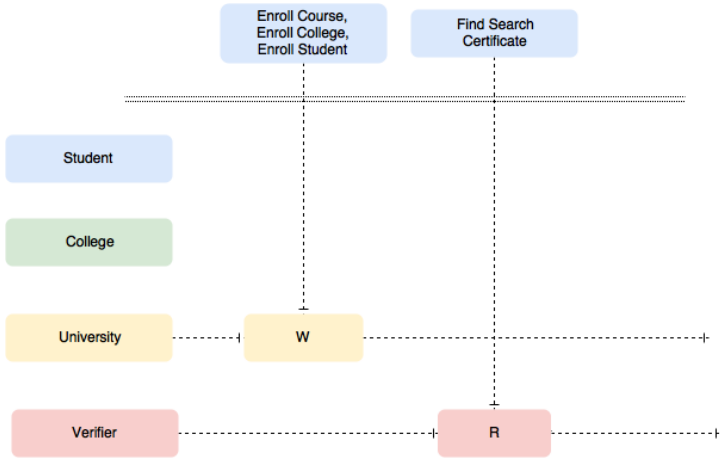

Identify Usecase Story
Identify Blockchain Components
Identify Transactions
Identify Business Permissions
Define Models
Deploy & Test in Playground

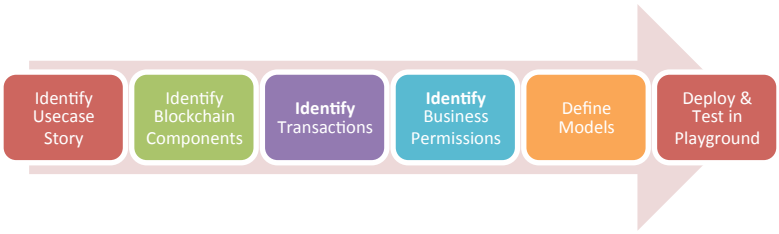
Step #4:


Student
College
University
Verifier
enroll()
enrollStudent()
requestAffiliation()
enrollCourse()
issueCertificate()
CertificateIssued-Event
findCertificate()
searchCertificateByID()
Blockchain



TASK#4: Identify Business Permissions





TASK#5: Define Models

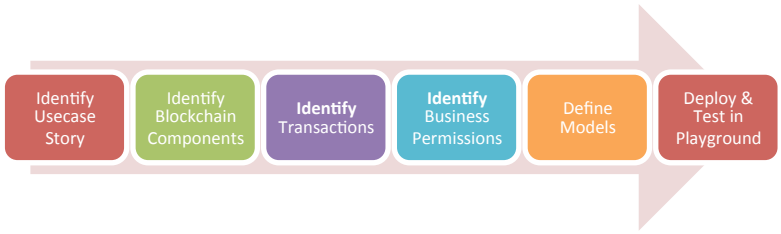
Step 1: Asset

- The main asset that we are dealing with is the Certificate that we will store in the blockchain and will be encrypted & digitally signed

```
/**
 * Digital Certificate Asset
 */
asset Certificate identified by certificateId {
  o String certificateId
  --> College college
  --> Student issuedTo
  --> Program program
  o String issuedBy
  o DateTime issuedDate
  o DateTime validUpto
  o String currentStatus
  o String metaData
}
```

- Note: (→) Defines the reference to other previously defined Network classes

```
/**
 * An abstract for all other intermediate assets
 */
abstract asset assetBase {
    o String memberId
    o String name
}

asset Program identified by memberId extends assetBase {
}
```



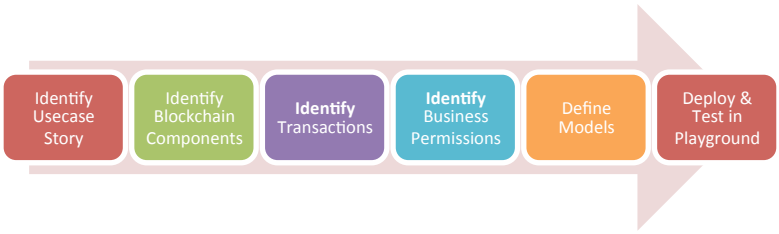
Step 2: Participants

- In the simple usecase of university we have Members with following properties
    - Identified by memberID
    - Has a Name
- As these properties and common within all participants let's encapsulate it in an abstract with name MemberParticipant

```
abstract participant MemberParticipant {
    o String memberId
    o String name
}
```

- We then extend all particiapnts with this base MemberParticipant abstract

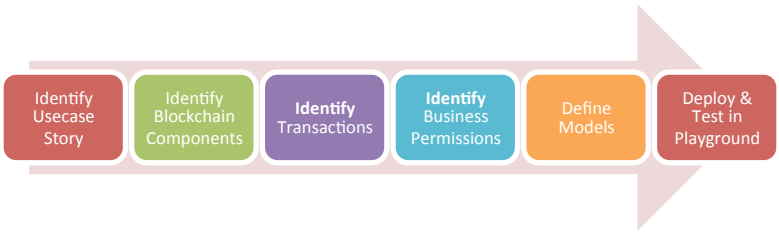
```
participant University identified by memberId extends MemberParticipant {
}

participant Student identified by memberId extends MemberParticipant {
    o String surname
    o DateTime dob
}

participant College identified by memberId extends MemberParticipant {
}

participant Verifier identified by memberId extends MemberParticipant {
}
```



Step 3: Transactions

- Transactions can be defined in a similar modeling language
- We need to define all the parameters that are needed to complete the requested transaction

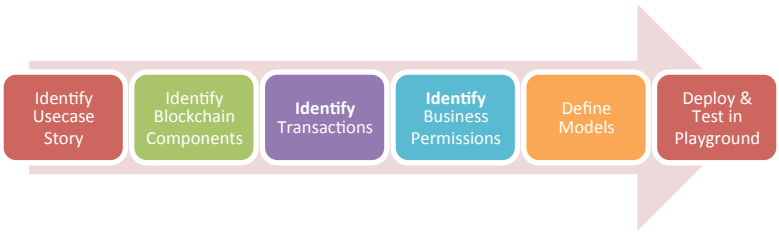
```
transaction issueCertificate {
    o Asset Certificate
}

transaction requestAffiliation {
    o String name
}

transaction enrollProgram {
    o String name
}

transaction enrollStudent {
    o String name
}

transaction verifyCertificate {
    o String certificateId
}

transaction searchCertificatesByID {
    o String enrollmentId
}
```



Step 4: Events

- Events are also declared similar to transactions and defines parameters that is broadcasted with the event.

```
event CerticateIssued {
    o String certificateId
}

event StudentEnrolled {
    o String studentID
}
```



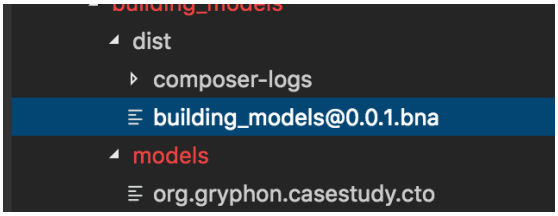
TASK#6: Deploy & Testing in Playground

Step 1: Create Archive File

- Copy and unzip chapter01.zip to a folder
- CD into the 'chapter1' folder
- Create a folder 'dist' and cd into it
- Now type the following command to create an archive file

```
> composer archive create -t dir -n ../
```

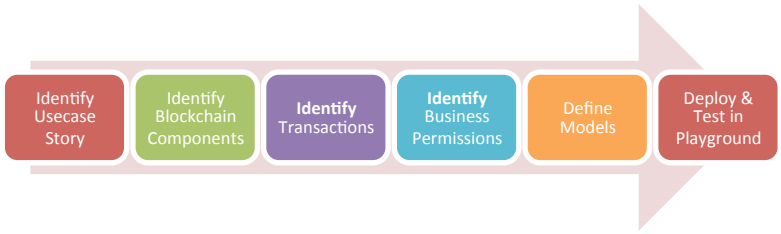
- This will create a .bna file in the dist folder which contain all the created model files in its binary (.cto : Business model files)



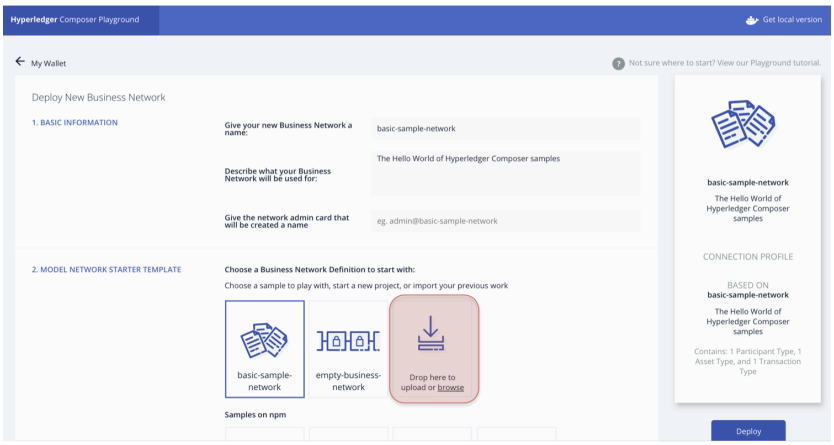
Step 2: Open your Internet browser and navigate to the following website:

`https://composer-playground.mybluemix.net/`

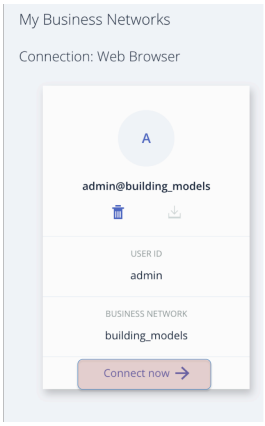
It will take a while to prepare your browser

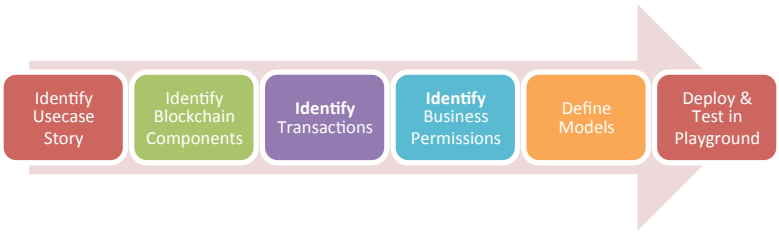


Use the deploy button as shown in figure below to deploy the archive



After deploying, start the application click the highlighted button as in figure below;

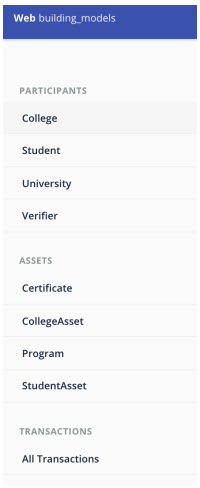




Step 3: Test using playground

Browse to the test panel as in figure below;

Left column will shows the entire asset, participants, which you can create and Test



SUMMARY

- Hyperledger is a Distributed Ledger Technology for the business
- Chaincode automates the Business processes
- Four Essential characteristics that make HyperLedger Suitable for Business are
    - Permissioned Network
    - Transaction confidentiality
    - No Mining needed compared to peers
    - Programmable using chaincode

REFERENCES

- http://hyperledger.org
- http://hyperledger.org/projects/fabric
- https://hyperledger-fabric.readthedocs.io/en/latest/build_network.html

CHAPTER 2: HYPERLEDGER FABRIC FUNDAMENTALS

Theory

In this chapter, we will cover the fundamental blocks of Hyperledger fabric, discuss about what constitutes of the distributed ledger in fabric, types of node in fabric, endorsement policies, membership cards and the certificate authority.

In the lab section we will create our first business network, model our first university use-case, write chaincode and its test.

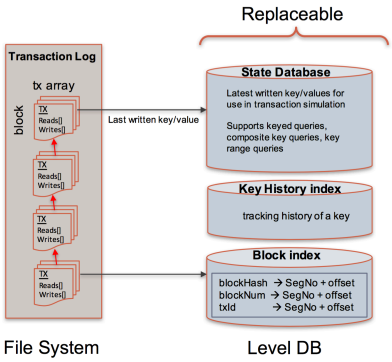
Finally we shall deploy the created business network and test it using chai and mocha framework. Also use explorer to add entries to the asset and validate the transaction history using the Historian registry and explorer.

Distributed Ledger

The ledger is the sequenced, tamper-resistant record of all state transitions. State transitions are a result of chaincode invocations ("transactions") submitted by participating parties. Each transaction results in a set of asset key-value pairs that are committed to the ledger as creates, updates, or deletes.

In Fabric Ledger has two parts:

- **State data**: Representation of current state of the assets
- **Transaction Logs**: Record of all the transactions which modified the state data



	Transaction Logs:	State data
Type	Is Immutable	Mutable
Operations	Create, Retrive	ALL – CRUD
DB	levelDB	levelDB / CouchDB
Behaviour	Embedded within Peers	Key-value paired (Json or Binary)
Query	Supports Simple Query	CouchDB Supports complex queries

Each transaction has a unique ID, its time-stamped and contains signatures of every endorsing peer and are submitted to ordering service

The ledger is comprised of a blockchain ('chain') to store the immutable, sequenced record in blocks, as well as a state database to maintain current fabric state. There is one ledger per channel. Each peer maintains a copy of the ledger for each channel of which they are member.

Nodes:

The concept of node is common in all blockchain technologies. Node becomes the communication end point in blockchain technology. Nodes connect to other nodes and that is how a blockchain is formed.

Nodes use some kind of peer-to-peer protocol for keeping the distributed ledger in sync across the network.

In public blockchain like Ethereum; anyone can participate as a node by downloading node client called wallet. But incase of permissioned Hyperledger network, things are quite different.

In Hyperledger, nodes need valid certificate to be able to communicate to the network and the participants use applications that connect to network by way of the nodes. Participant's identity is not the same as the nodes identity. The participant that executes or invokes a transaction their certificate is used for signing that transaction. The network to check if they should trust the node uses node's certificate or not. Incase the nodes certificate is revoked or has expired in that case the transaction thought signed by a valid certificate held by the participant is broadcasted to the network but the transaction will be rejected because the certificate that node is using has been expired or has been revoked.

In Hyperledger, all Nodes are NOT equal. There are three distinct types of nodes:

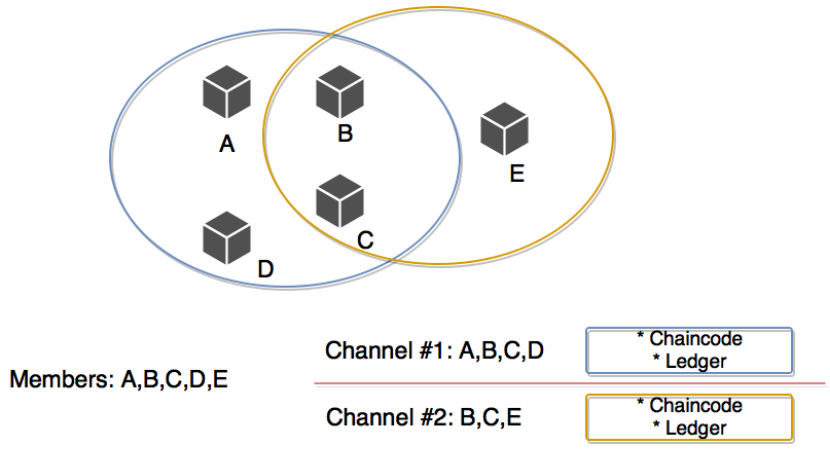
1. **Client Node:** That initiates the transaction
2. **Peer Nodes:** Commits Transaction & keeps the data in sync across the ledger

3. **Ordered:** They are the communication backbones and responsible for the distribution of the transactions

Channel:

Members can participate on multiple hyperledger blockchain networks. Transaction in each network is isolated and this is made possible by way of what is referred to the channels

Channel is a data partitioning mechanism to control transaction visibility only to stakeholders, other members on the network are not allowed to access the channel and will not see transactions on the channel



A chaincode may be deployed on multiple channels, each instance is isolated within its channel. Similarly each channel maintains their own ledger. [This is as of Hyperledger version 1.0 and may change].

Separation of the ledger, by defining the specific channel for each ledger and peer node memberships are defined in Chaincode configuration. It is stored in the Genesis block of the ledger, which also stores the members, policies, and anchor peers. The Genesis block defines the read/write access on a channel.

Peers are connected to the channel and can receive all the transactions that are broadcasted on that channel. Consensus takes place within a channel by members of the channel.

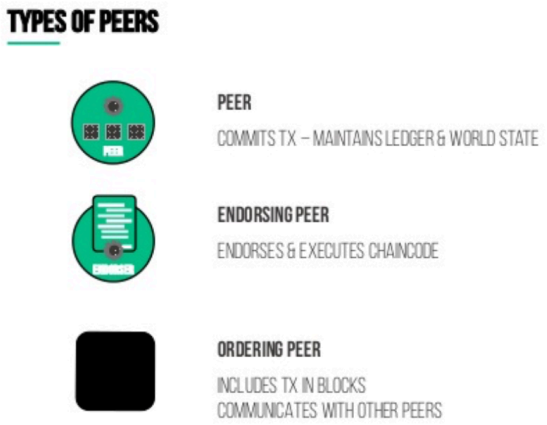
Node Types: Ordered, Anchor & Endorser

Client Node:

The client represents the entity that acts on behalf of an end-user. It must connect to a peer for communicating with the blockchain. The client may connect to any peer of its choice. Clients create and thereby invoke transactions.

Peer Node:

They are nodes that maintain the state and copy of a shared ledger. Peers are authenticated by certificates issued by MSP. In HyperLedger Fabric, there are three types of peer nodes depending upon the assigned roles:



Endorsing Peers (Endorsers) – An endorser executes and endorses transactions

The endorsing peers take up the role of endorsing transactions before it is ordered and committed as per the policy defined in Chaincode. The client application creating the transaction and sends it to endorsing peers as per the policy in chaincode. The endorsement policy is instantiated at the chaincode of the client application and forwarded to the endorsing peers. The endorsing peer evaluates and validates the transaction and produces an endorsement signature and then returns it to the application. There may be one or more pre-specified set of endorsing peers involved as per the endorsement policy. The transaction is evaluated and declared

valid only if it has been endorsed by the endorsing peers as per policy.

Ordering Service Nodes (Orderers) –

- Responsible for consistent ledger state across the network
    - Consensus Mechanism
    - Ensures order of Transactions
- Creates Blocks & Provides atomic delivery/broadcast
- Message Oriented Middleware options for orderer service in Hyperledger:
    - SOLO: Single Node (Good for Development)
    - Kafka: High throughput, scalable & Fault Tolerant

All transactions from the network are received by the orderer and it orders and groups them and then packages the transactions and creates blocks. The orderer service delivers blocks to the committing peers allowed to be part of a Channel. The orderer services do not review transaction information. The orderer makes guaranteed atomic delivery of blocks to committing peers on the channel. The orderer supports multiple channels using a publish/subscribe messaging system (based on Apache Kafka and Zookeeper). The ordered provides a practical Byzantine Fault tolerance for failures without a single-point of failure.

Ordering service nodes also provide the following services:

- Authentication of clients
- Maintenance of a system chain that defines ordering service configurations, root certs and MSP IDs for authenticated organizations and a grouping of profiles containing the various consortia within the network.
- Filtering and validation for configuration transactions that reconfigure or create a channel.

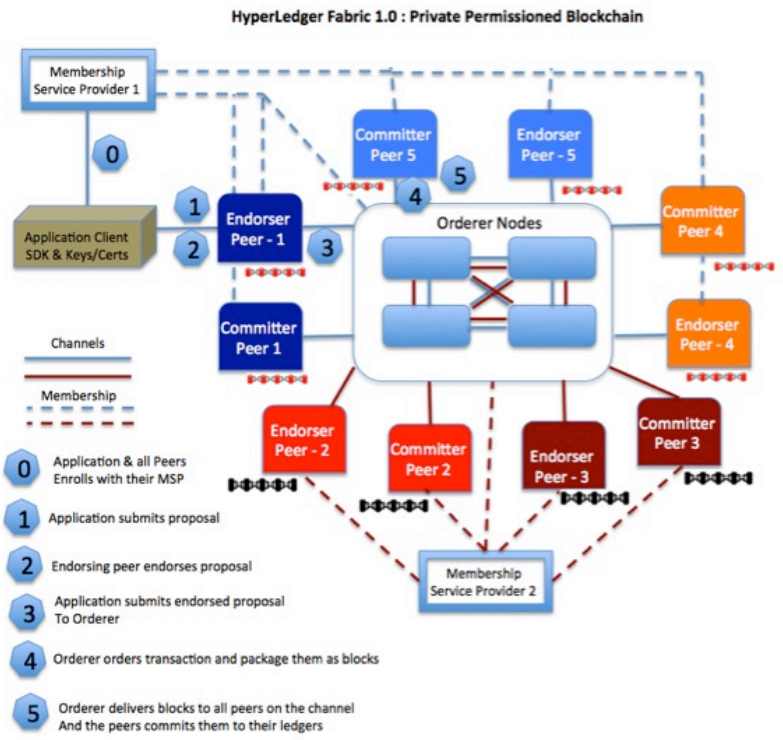
Committing Peers (Committers) – Verifies endorsements and validates transaction results

They receive blocks from Orderer service, which already endorsed by the endorsing peers. The Committing peers ultimately commit the transactional state by adding the blocks to the ledger. Before commital, the peers validate or invalidate the transaction by verifying if the endorsement policies are met, authenticate the signatures and also verify the version info (if there is any double spending).

HyperLedger Transaction Flow:

Assuming the HyperLedger Fabric 1.0 up and running, in a typical transaction flow of asset exchange:

- All application users and peer node members are registered in the MSP and issued with Keys/certificates from the CA for authenticating the network. The Chaincode representing the initial state is installed on the peers and the channels are active.
- The application client initiates a transaction (Client A makes a request to Client B to transfer an asset). The endorsement policy states that the request must be endorsed by Peer A and Peer B.
- The application submits a transaction proposal to the endorsing peers A and B.
- The endorsing peers receive and verify the transaction proposal and its signature, then executes the transaction and return a signed proposal response back to the application client.
- The application client verifies the responses from the endorsing peers. It assembles the response into a transaction and sends it to the Orderer service.
- The Orderer services order the transactions chronologically and package those transactions as blocks specific to a channel.
- The Orderer service delivers the blocks of the transactions to all the peers on the channel.
- The peers perform the validation of the blocks for endorsement policy, signatures verification and version info, and finally appends the block to the chain and commit the state database – Notifies the application client.



Endorsement Policies

An endorsement policy is a condition on what *endorses* a transaction. Blockchain peers have a pre-specified set of endorsement policies, which are referenced by a deploy transaction that installs specific chaincode. Endorsement policies can be parameterized, and these parameters can be specified by a deploy transaction.

To guarantee blockchain and security properties, the set of endorsement policies should be a set of proven policies with limited set of functions in order to ensure bounded execution time (termination), determinism, performance and security guarantees.

Dynamic addition of endorsement policies (e.g., by deploy transaction on chaincode deploy time) is very sensitive in terms of bounded policy evaluation time (termination), determinism, performance and security guarantees. Therefore, dynamic addition of endorsement policies is not allowed, but can be supported in future.

Transaction evaluation against endorsement policy

A transaction is declared valid only if it has been endorsed according to the policy. An invoke transaction for a chaincode will first have to obtain an *endorsement* that satisfies the chaincode's policy or it will not be committed. This takes place through the interaction between the submitting client and endorsing peers as explained in Section 2.

Formally the endorsement policy is a predicate on the endorsement, and potentially further state that evaluates to TRUE or FALSE. For deploy transactions the endorsement is obtained according to a system-wide policy (for example, from the system chaincode).

An endorsement policy predicate refers to certain variables. Potentially it may refer to:

1. Keys or identities relating to the chaincode (found in the metadata of the chaincode), for example, a set of endorsers;
2. Further metadata of the chaincode;
3. Elements of the endorsement and endorsement.tran-proposal;
4. And potentially more.

The above list is ordered by increasing expressiveness and complexity, that is, it will be relatively simple to support policies that only refer to keys and identities of nodes.

The evaluation of an endorsement policy predicate must be deterministic. An endorsement shall be evaluated locally by every peer such that a peer does *not* need to interact with other peers, yet all correct peers evaluate the endorsement policy in the same way.

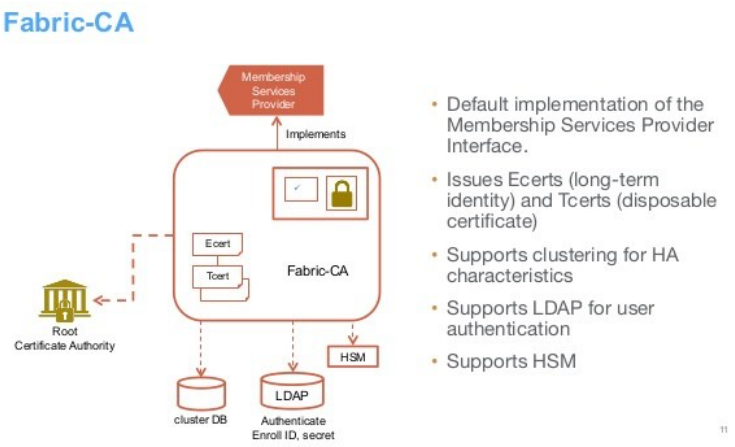
Membership Service Provider (MSP)

"Abstract component of the system that provides credentials to the clients, and the peers for then to participate in the Hyperledger Fabric network"

MSP implementation is based on the PKI (Public Key Infrastructure)

Service it Provides:

- Authorization Service
  - Role based
  - Examples:
    - Can this user further issue identity?
    - Can user deploy chaincode?
- Authentication Service
  - Where users Identity gets validated
  - Examples:
    - Is the user's/peer's certificate valid?
    - Is peer allowed to participate?



Certificate Authority

As a platform for **permissioned** blockchain networks, Hyperledger Fabric includes a modular **Certificate Authority (CA)** component for managing the network identities of all member organizations and their users. The requirement for a permissioned identity for every user enables ACL-based control over network activity, and guarantees that every transaction is ultimately traceable to a registered user.

- The CA (Fabric CA by default) issues a root certificate (rootCert) to each member (organization or individual) that is authorized to join the network.
- The CA also issues an enrollment certificate (eCert) to each member component, server side applications and occasionally end users.
- Each enrolled user is also granted an allocation of transaction certificates (tCerts). Each tCert authorizes one network transaction.

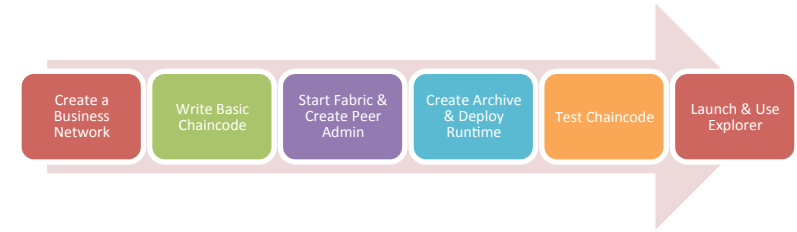
This certificate-based control over network membership and actions enables members to restrict access to private and confidential channels, applications, and data, by specific user identities.

AIM

After you completed this chapter with lab exercises, you will be able to:

- Have an understanding of Registries, distributed ledgers, types of nodes & their functions
- Create your first Hyperledger Business Network
- Start Hyperledger fabric, create archive, deploy runtimes and execute the tests
- Use explorer to add / update asset, run transactions and validate general system business methods like historian and identities

Lab Exercise 2: CHAINCODE BASICS



1. **Create a Business Network using Yo Scaffolding**
2. **Write Basic Model + Chaincode + Test**
    a. **npm install**
3. **Start Fabric and Create PeerAdmin Card**
4. **Deploy Runtime**
    a. **Create Archive**
    b. **Import Admin Card**
    c. **Deploy To Runtime**
5. **Testing Chaincode**
6. **Running Explorer**
    a. **Launch explorer**
    b. **Use Explorer to add a asset**
    c. **Run Transactions**
    d. **View Historian**

TASK#1: Create a Business Network Scaffolding

Step 1: Create a new folder and cd into it. Launch terminal window and use following commands

```
mkdir chapter2
```

```
cd chapter2
```

Step 2: Use Yo-Generator to create a business Scaffolding. Using the terminal window enter following commands

```
yo hyperledger-composer
```

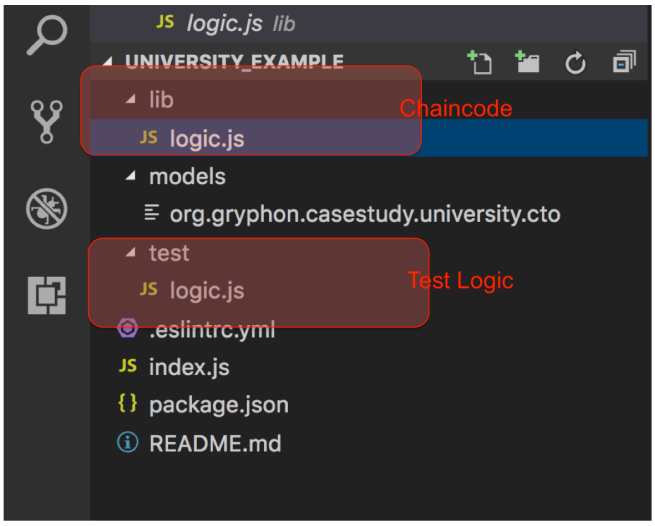
Step 3: Choose **'Business Network'** from the choice using your keyboard

```
Welcome to the Hyperledger Composer project generator
? Please select the type of project: (Use arrow keys)
  Angular
❯ Business Network
  Model
```

Step 4: After you have selected Business Network option you will be asked few details which you need to provide, shown below;


```
Welcome to the Hyperledger Composer project generator
? Please select the type of project: Business Network
You can run this generator using: 'yo hyperledger-composer:businessnetwork'
Welcome to the business network generator
[? Business network name: university_example
[? Description: Creating models for our university example
[? Author name:  Ernesto
[? Author email: university@example.com
[? License: GPL 1.0
[? Namespace: org.gryphon.casestudy.university
```

Step 5: Open the 'chapter02' folder in Visual Studio code to find the following directory structure



Task 1 is complete!

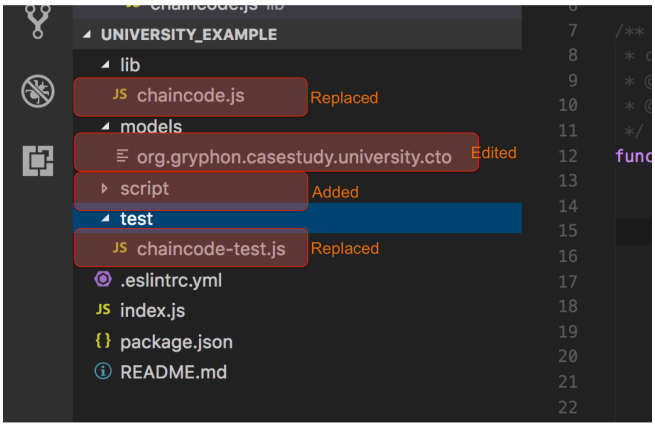
TASK#2: Write basic model, chaincode and test

**** We will use the code 'chapter02' provided with this lab as a sample chaincode**

Step 1: Open **'chapter02'** code folder supplied with this book using Visual Studio Code;

This folder is similar to the Business Network Scaffolding created earlier but with few changes as below;

- 'logic.js' under lib folder is replaced with our university example chaincode and renamed to 'chaincode.js'
- 'logic.js' under test folder is replaced with 'chaincode-test.js' file which will test our university sample chaincode
- 'org.gryphon.casestudy.university.cto' under the models folder is modified to use a single asset in our university example
- Additional 'script' folder added to speed up the development process



Step 2: Let us review what we have in our university **'basic'** example

- Model file: org.gryphon.casestudy.university.cto

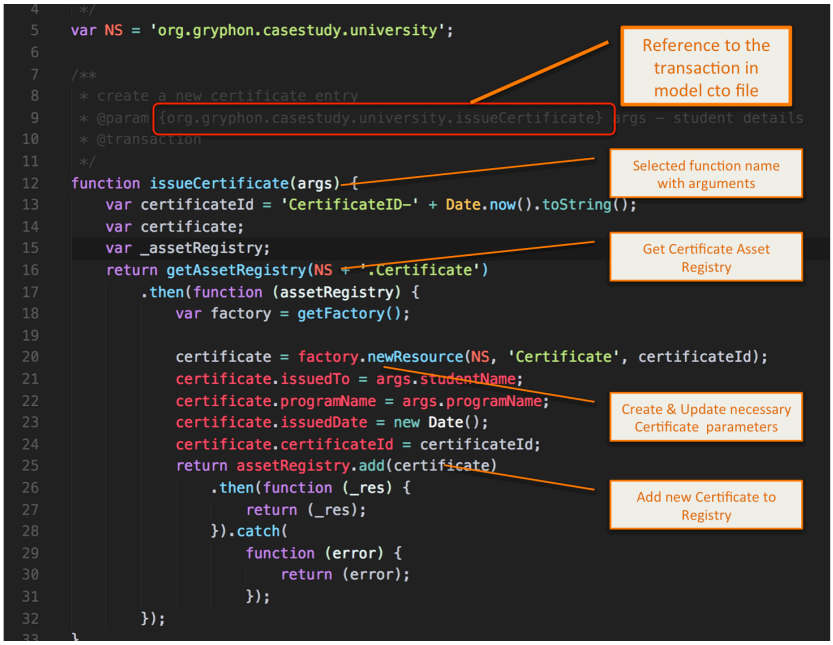
```
5
6   namespace org.gryphon.casestudy.university
7
8   /**
9    * Digital Certificate Asset
10   */
11  asset Certificate identified by certificateId {
12    o String certificateId
13    o String issuedTo
14    o String programName
15    o DateTime issuedDate
16  }
17
18  transaction issueCertificate {
19    o String studentName
20    o String programName
21  }
22
```

Name Space defination

Defining a Certificate Asset

Defining a Transaction which will create new asset in the registry

- Review Chaincode: 'chaincode.js'



- Review Test code: 'chaincode-test.js'


```
const BusinessNetworkConnection = require('composer-client').BusinessNetworkConnection;

require('chai').should();

const _timeout = 90000;
const NS = 'org.gryphon.casestudy.university';

describe('Connect Network', function () {

    this.timeout(_timeout);
    let businessNetworkConnection;
    before(function () {
        businessNetworkConnection = new BusinessNetworkConnection();
        return businessNetworkConnection.connect('admin@university_example');
    });
```

Use composer client to connect to the Business Network

Using 'CHAI' framework for testing

Before any test connect to the network

Use admin card to connect to the Business network

'chaincode-test.js' => continued…

```
describe('#issueCertificate', () => {

    const PROGRAM_NAME = 'MBA';
    const STUDENT_NAME = 'John';

    it('should be able to issue Certificate', () => {
        const factory = businessNetworkConnection.getBusine

        // create the stepUp transaction
        const issueCertificate = factory.newTransaction(NS, 'issueCertificate');
        issueCertificate.studentName = STUDENT_NAME;
        issueCertificate.programName = PROGRAM_NAME;
        return businessNetworkConnection.submitTransaction(issueCertificate)
            .then((_res) => {
                return businessNetworkConnection.getAssetRegistry(NS + '.Certificat
            })
            .then((assetRegistry) => {
                // re-get the asset registry
                return assetRegistry.getAll();
            })
            .then((allCertificates) => {
                // the owner of the commodity should no
                let index = allCertificates.length - 1;
                console.log('Number of Certificates: ' + allCertificates.length);
                console.log('Certifcate ID: ' + allCertificates[index].certificateI
                allCertificates[index].issuedTo.should.equal(STUDENT_NAME);
                allCertificates[index].programName.should.equal(PROGRAM_NAME);
            });
    });
});
```

Start 1st Test in 'it' block

Create Transaction

Define inputs to transaction function

Submit Transaction

Get Certificate Registry for validation

All Items returned

Display last certificate entry

Validate the entry using chai test

Task 2 is complete!

TASK #3: Start fabric and create peer admin card

Step 1: Launch terminal window and use following commands to change the directory to the installed fabric-tools folder

```
cd ~/fabric-tools/
```

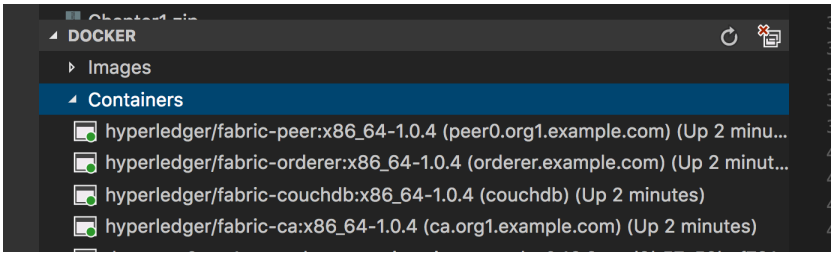
Step 2: Use the startFabric.sh script to start the fabric environment.

```
./startFabric.sh
```

This command will create and launch the necessary docker containers

```
ARCH=$ARCH docker-compose -f "${DIR}"/composer/docker-compose.yml up -d
Creating couchdb ... done
Creating peer0.org1.example.com ... done
Creating couchdb ...
Creating orderer.example.com ...
Creating peer0.org1.example.com ...
```

You can see containers in your Visual Studio Code (Docker Plugin installed)



Step 3: Create Peer Admin Card to be able to deploy business runtimes.

From the fabric-tools directory use the createPeerAdmin.sh script

```
cd ~/fabric-tools/
```

```
./createPeerAdminCard.sh
```

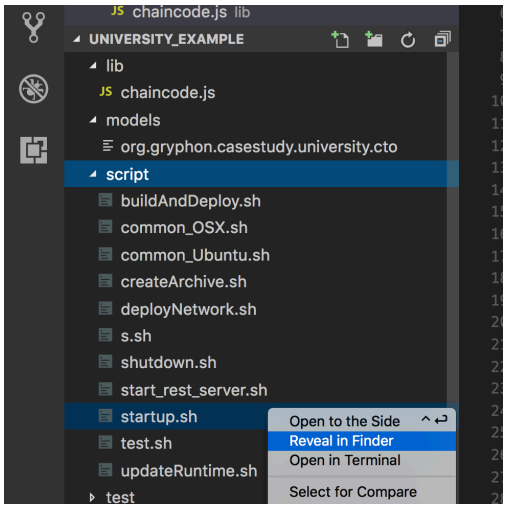
```
Successfully created business network card file to
        Output file: /tmp/PeerAdmin@hlfv1.card

Command succeeded


Successfully imported business network card
        Card file: /tmp/PeerAdmin@hlfv1.card
        Card name: PeerAdmin@hlfv1

Command succeeded

Hyperledger Composer PeerAdmin card has been imported
The following Business Network Cards are available:

Connection Profile: hlfv1
```

PeerAdmin@hlfv1	PeerAdmin

```
Issue composer card list --name <Card Name> to get details a specific carc

Command succeeded
```

Step 4: To speed up the development process few scripts are available. You can use script in the 'chapter02' code folder to start fabric and create peer admin card

***** From next time we will be using this script only to start fabric and create peer card before deploying hence follow these steps *****



Change the path below in startup.sh script corresponding to your OS

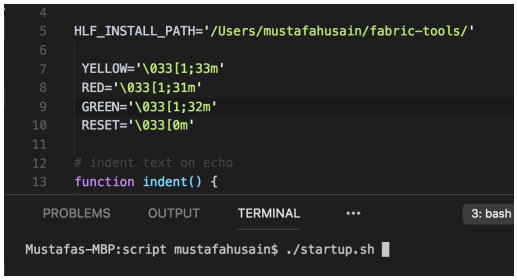
- **For MAC USERS**

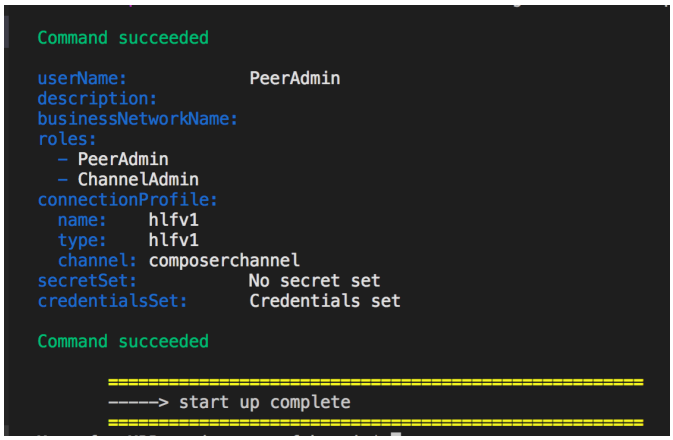
For Ubuntu Users:

```
HLF_INSTALL_PATH='~/(username)/fabric-tools/'
```

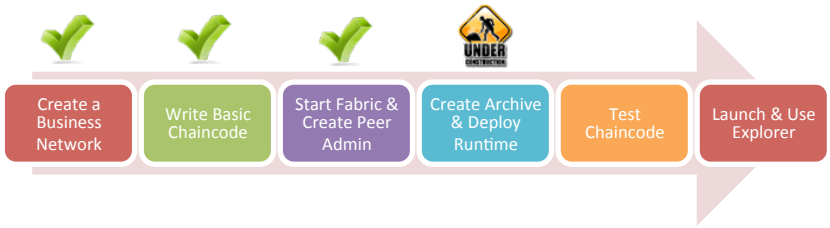
After this change has been done you can run this script from the visual studio terminal window directly

```
username-MBP:script username $ ./startup.sh
```





Task 3 is complete!



TASK #4: Create Archive & Deploy Runtime

Step 1: Open 'chapter02' code provided in Visual Studio Code. Open terminal window and cd into the 'chapter02' directory

Step 2: Create a 'dist' directoty

```
mkdir dist
```

Step 3: Type the following command to create the archive

```
composer archive create -t dir -n . -a
./dist/university_example.bna
```

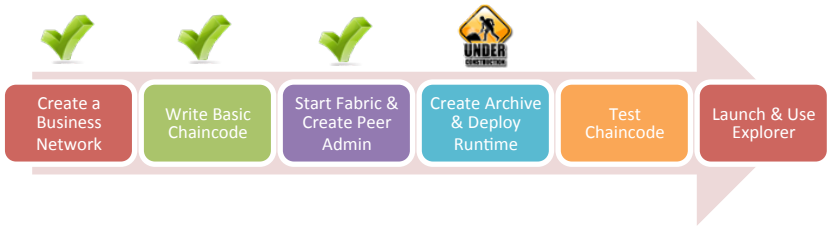
```
/university_example.bna
Creating Business Network Archive

Looking for package.json of Business Network Definition
        Input directory: /Users/mustafahusain/Desktop/university_example

Found:
        Description: Creating models for our university example
        Name: university_example
        Identifier: university_example@0.0.1

Written Business Network Definition Archive file to
        Output file: ./dist/university_example.bna

Command succeeded
```

This creates the archive **'university_example.bna'** in the ./dist folder created earlier.

Step 4: Now install the created peer admin card

First cd into the created 'dist' directory

```
cd dist
```



Type the following command in the terminal window

```
composer runtime install --card PeerAdmin@hlfv1 --
businessNetworkName university_example
```

This will take sometime and deploy the runtime

```
Mustafas-MBP:university_example mustafahusain$ cd dist/
Mustafas-MBP:dist mustafahusain$ composer runtime install --card PeerAdmin@hlfv1 --bu
etworkName university_example
✔ Installing runtime for business network university_example. This may take a minute.

Command succeeded
```

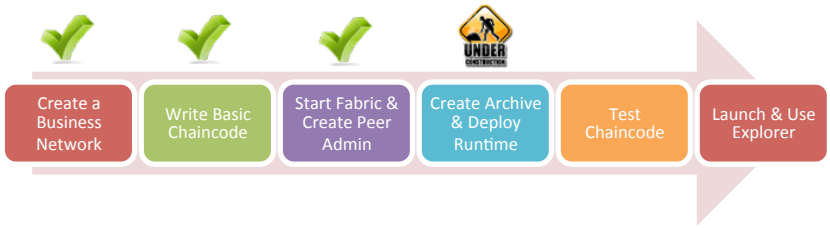
Step 5: Start Business network and create a network admin card to handle all network related operations

Type the following command in the terminal window and press enter:

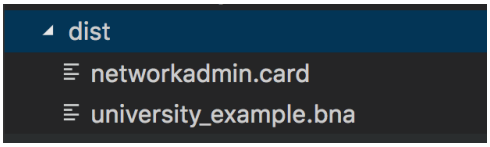
```
composer network start -c PeerAdmin@hlfv1 -A admin -S
adminpw -a university_example.bna --file
networkadmin.card
```

This uses the Peer Admin card to create the Network Admin cards

```
Starting business network from archive: university_example.bna
Business network definition:
        Identifier: university_example@0.0.1
        Description: Creating models for our university example

Processing these Network Admins:
        userName: admin

✔ Starting business network definition. This may take a minute...
Successfully created business network card:
        Filename: networkadmin.card

Command succeeded
```



This command will create a network admin card **'networkadmin.card'** in dist folder



Step 6: Import Business network admin card just created. Type the following command in terminal window

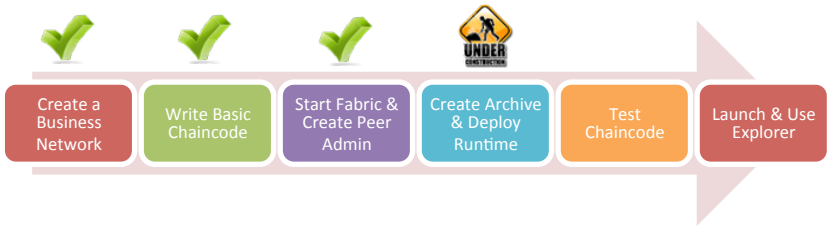
```
composer card import --file networkadmin.card
```

```
Successfully imported business network card
        Card file: networkadmin.card
        Card name: admin@university_example

Command succeeded
```

Step 7: To verify everything has been completed successfully we need to ping the network and check for success. Type the following command in terminal window

```
composer network ping --card admin@university_example
```

```
The connection to the network was successfully tested: university_example
        version: 0.16.3
        participant: org.hyperledger.composer.system.NetworkAdmin#admin

Command succeeded
```



[Alternative]: Doing all the steps one by one is cumbersome, hence to speedup the development process, a script has been written.

a. Move into the scripts folder of 'chapter02'

```
cd script
```

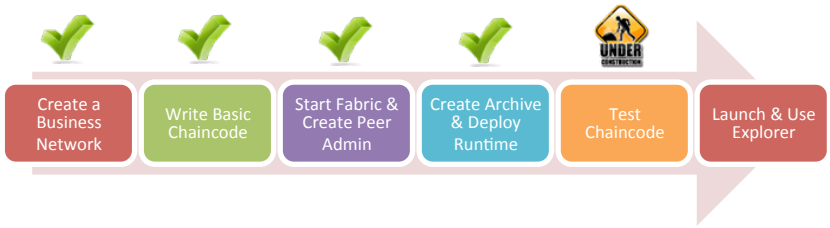
b. Run buildAndDeploy.sh script

```
./buildAndDeploy.sh
```

What does the script do?

- Everything done in Task #3 and #4 is done via this script
- Please ensure to have the correct path of hyperledger folder setup according to your operating system

Task 4 is complete!



TASK#5: Running chaincode Test

Step 1: Open 'chapter02' code provided in Visual Studio Code. Open terminal window and cd into the 'chapter02' directory

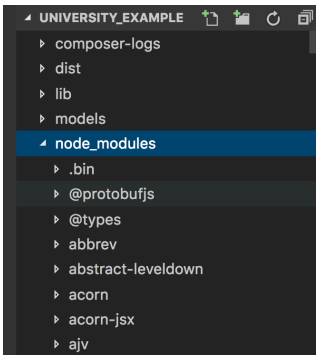
Step 2: Install the node modules by typing the following command into the terminal window

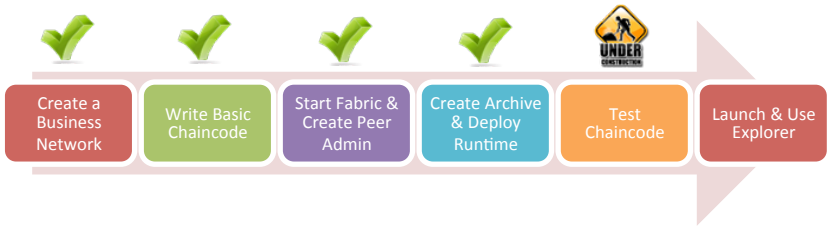
```
npm install
```

This will take a bit of time and install all the required node modules in the ./node_module folder

```
Mustafas-MBP:university_example mustafahusain$ npm install
npm WARN deprecated fs-promise@1.0.0: Use mz or fs-extra^3.0 with Promise Support
npm WARN deprecated crypto@0.0.3: This package is no longer supported. It's now a built-in Node modu
le. If you've depended on crypto, you should switch to the one that's built-in.
(                                ) ⁝ extract:core-js: sill extract check-error@1.0.2
```

It may show some warnings but that should be OK.



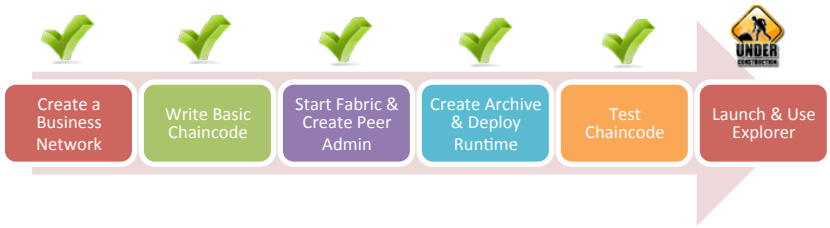


Step 3: After the node modules are installed type the following command to run the test

```
npm test
```

```
> university_example@0.0.1 test /Users/mustafahusain/Desktop/university_example
> mocha --recursive


  Connect Network
    #issueCertificate
Number of Certificates: 1
Certifcate ID: CertificateID-1521369285345
      ✓ should be able to issue Certificate (3101ms)


  1 passing (5s)
```

Task 5 is complete!



TASK#6: Launch & use explorer

Step 1: Open 'chapter02' code provided in Visual Studio Code. Open terminal window and cd into the 'chapter02' directory

Step 2: Launch explorer using the following command

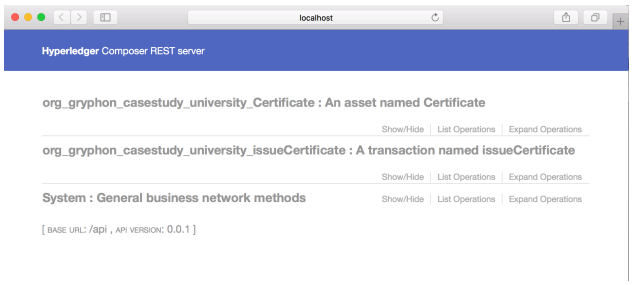
```
composer-rest-server -c "admin@university_example"
```

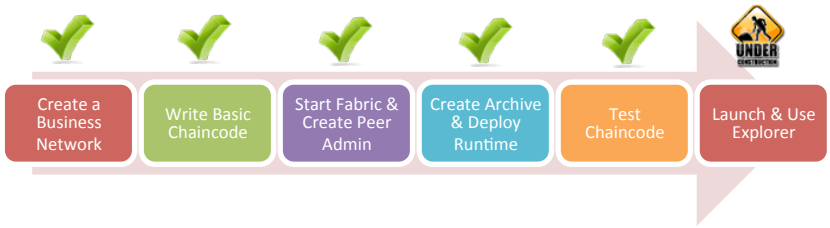
On successful completion of the command explorer rest server will be launched address as below;

```
Discovering types from business network definition ...
Discovered types from business network definition
Generating schemas for all types in business network definition ...
Generated schemas for all types in business network definition
Adding schemas for all types to Loopback ...
Added schemas for all types to Loopback
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

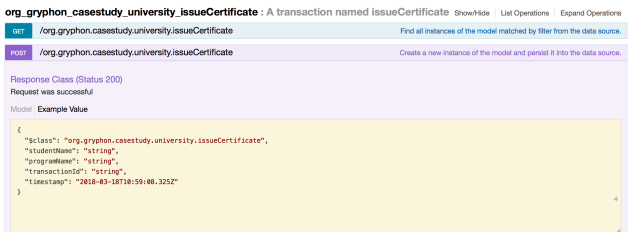
Step 3: Launch internet explorer and enter the following url:

http://localhost:3000/explorer/

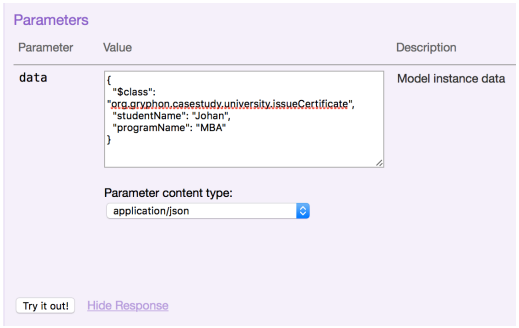




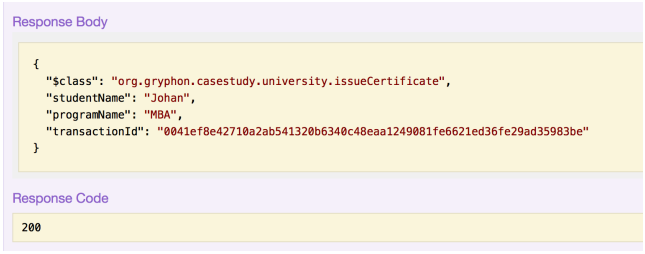
Step 4: Click on “org_gryphon_casestudy_university_issueCertificate : A transaction named issueCertificate”

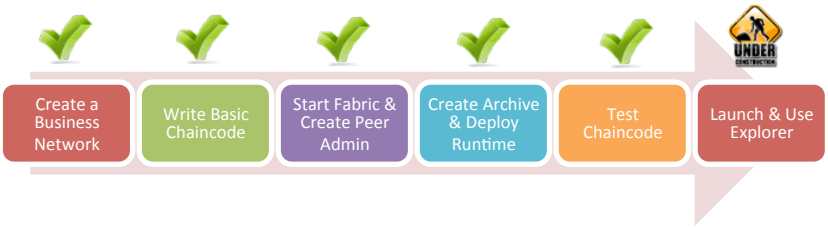


Using the example modify the transaction parameters and click “Try it out” as below

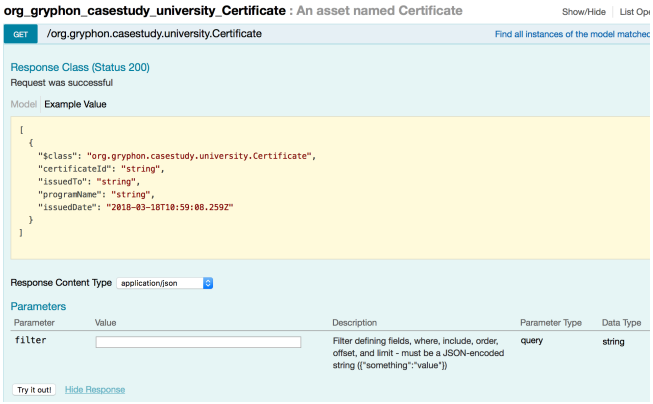


On clicking “Try it out” button you will response similar to below;

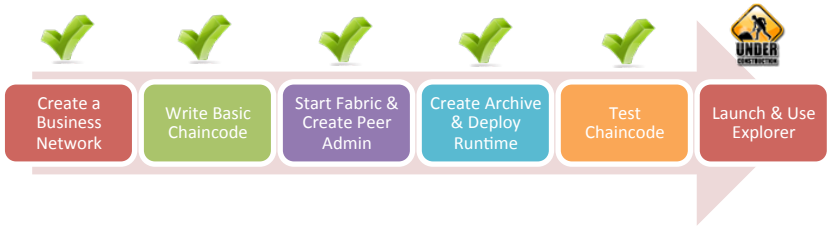




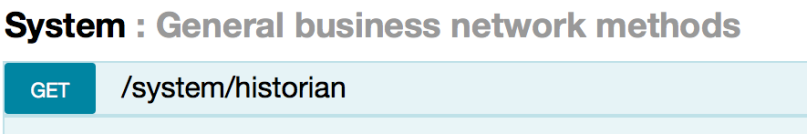
Step 5: Click on “org_gryphon_casestudy_university_Certificate : An asset named Certificate”



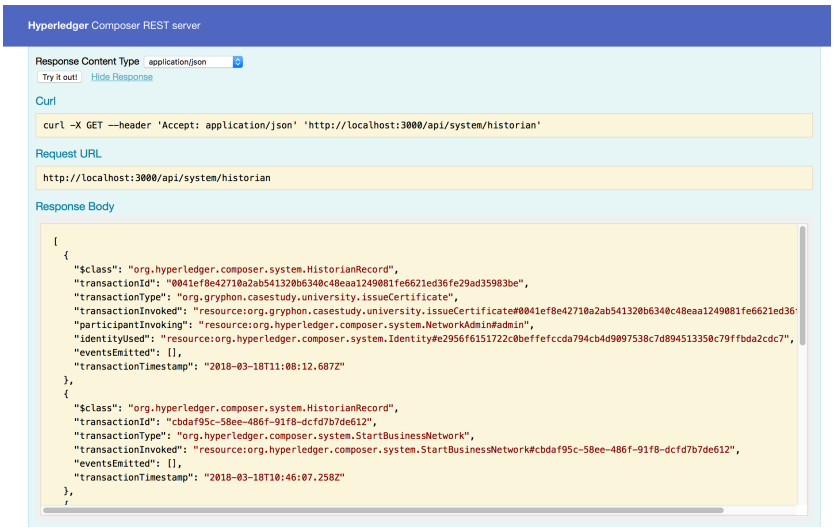
On clicking “Try it out” button you will receive the list of all the ‘Certificate’ Assets as below;



Step 6: Click on "System : General business network methods". Click "Get" method and then click "Try it out!" button at the bottom;



Historian will list all the transactions done to date



Task 5 is complete!

SUMMARY

Hyperledger is a permissioned network. To be able to deploy chaincode and runtime network business cards are needed.

In this chapter we have learnt the following:

- How to start fabric composer
- Create Peer Admin Card and the Network Admin Card
- How chaincode is written and tested using chai framework
- Create archive and deploy the runtime
- How to use scripts for deploying and running during development process
- How use explorer to validate transactions, assets and the system methods

REFERENCES

- http://hyperledger.org
- http://hyperledger.org/projects/fabric
- https://hyperledger-fabric.readthedocs.io/

CHAPTER 3: PARTICIPANT, IDENTITIES & ACCESS CONTROL

Theory

In this chapter, we will cover access control and authorization, which are a very important part of Hyperledger and the security architecture of a business network. Hyperledger enables an administrator control what resources or data a participant, or indeed participant role - is authorized to see or do, in a business network.

In the lab section we will explore and define an ACL file for our University example use-case and provide access permissions to different participants.

Access control rules (the language that defines ACLs) fall into two main areas:

- Authority to access system, network or administrative resources and operations in the System namespace (governing Network and System operations)
- Authority to access resources or perform operations within a given business network itself (like Create, Read, Update assets)

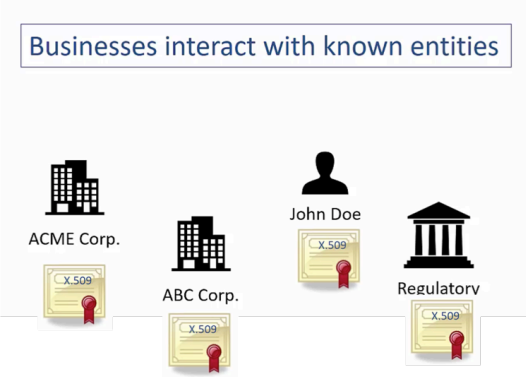
Finally, we will use the online Playground to try out some simple and conditional access rules. In doing so, we will interact with our University use-case network as various identities - ultimately, it is the users of the blockchain that we want to apply access control to.

Participants and identities

A **Participant** is an *actor* in a business network. A participant might represent an individual or an organization. A participant can create assets, and also share assets with other participants. A participant can interact with assets by submitting transactions.

A participant has a set of Identity that can be validated to prove the identity of that participant.

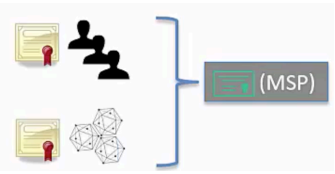
Hyperledger Fabric is a Private Permissioned Network for Businesses. Businesses interact with only known identities.

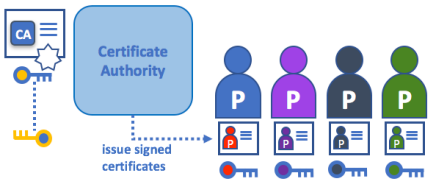


In Hyperledger, participants are separated from the set of identities that they can use to interact with a business network.

In order for a new participant to join a business network, a new instance of that participant must be created in the business network. The participant instance stores all of the required information about that participant, but it does not give that participant access to interact with the business network.

In order to grant the participant access to interact with the business network, an identity must then be issued to that participant. The new participant can then use that identity document to interact with the business network.



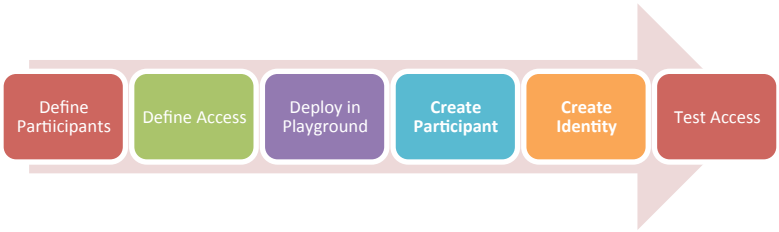


Identity usually expire after a set period of time. Identity may also be lost or stolen. If the identity expires, or if it needs to be replaced, then it must be Revoked so it can no longer be used to interact with the business network.

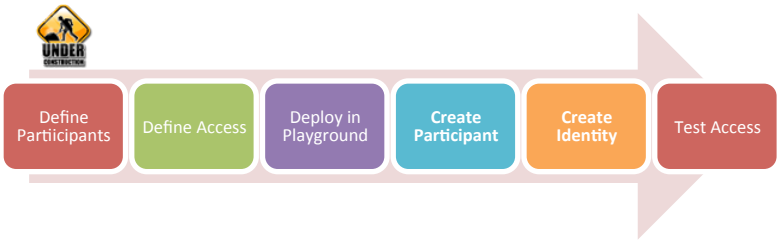
However, revoking an identity document does not remove the information about that participant and any assets that they own. Revoking the identity document simply removes the participant's ability to interact with the business network using that identity. Access to the business network can be restored by issuing the participant with a new identity.

These participant and identity management actions are performed by an existing participant in the business network, for example a regulatory body, or a participant in the same organization who has been trusted to manage participants/identities in that organization.

Lab Exercise 3: Adding Participants, Identities & Access Controls



1. **Define Participants and associated Transactions**
2. **Define Access (ACL File)**
3. **Deploy in Playground**
4. **Create Participant**
5. **Create Identity**
6. **Login To Business Network & Test Access**



TASK#1: Define Participants & Transactions

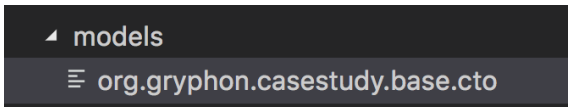
Step 1: Open 'chapter03' code provided in Visual Studio Code.

*#**Info**: 'chapter03' is a Business Network Scaffolding created as in Chapter#2-Task 1 with following changes:*

1. *Chaincode 'logic.js' is replaced with 'chaincode.js'*
2. *Test 'logic.js' is replaced with 'chaincode-test.js'*
3. *'script' folder is added*

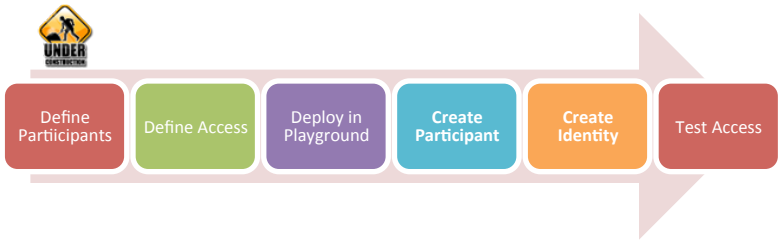
Step 2: Adding Participants:
We will segregate all the participants in different namespaces that helps in defining access.

Step 2.1: Open *'models/org.gryphon.casestudy.base.cto'* file in Visual Studio Code



Step 2.1: Defining abstract for class for all participants with mandatory common attributes

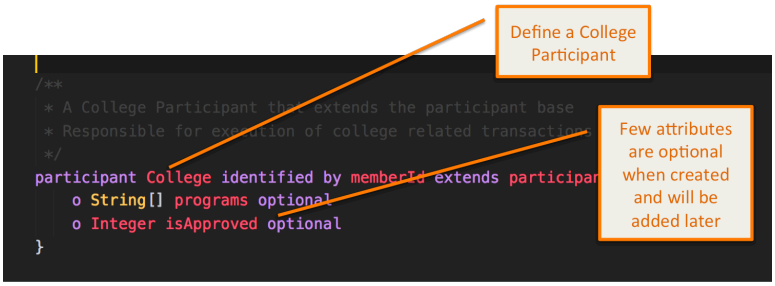
```
/**
 * Partipant Base
 */

namespace org.gryphon.casestudy.base

abstract participant participantBase {
    o String memberId
    o String name
}
```



Step 2.2: Open *'models/org.gryphon.casestudy.college.cto'* file in Visual Studio Code. Define Namespace for college related participant and transactions.

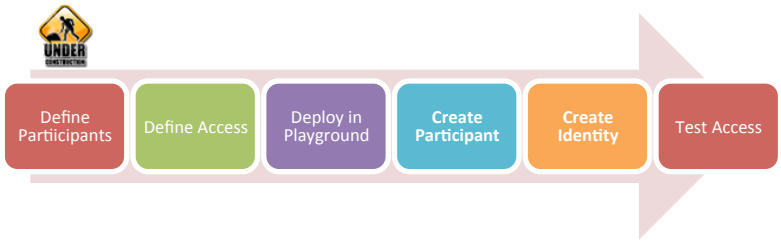
```
namespace org.gryphon.casestudy.college
import org.gryphon.casestudy.base.*
```

Step 2.3: Define 'College' Participant



Step 2.4: Define Transactions that 'College' Participant can carry out

```
/**  College Specific transactions **/

/**
* Request affiliation to a University
*/
transaction requestAffiliation {
    o String name
}

/**
* Enroll new Programs and cources
*/
transaction enrollProgram {
    o String collegeId
    o String programName
}
```



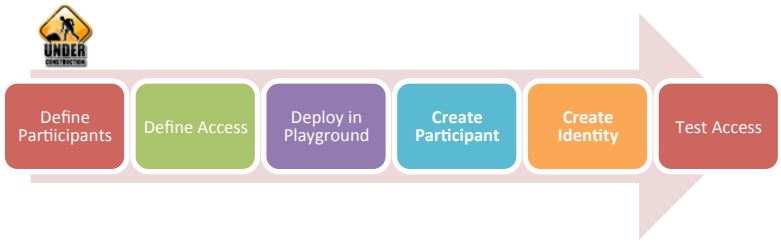
Step 2.5: Similarly Open *'models/org.gryphon.casestudy.university.cto'* file in Visual Studio Code and define university Participant and Transactions

** **Note**: We have already seen Certificate Asset in previous chapter.

```
/**
 * A University Participant is the governing authority
 */
participant University identified by memberId extends participantBase {
}

/**  University Specific transactions **/

/**
 * Issue a certificate to a student
 * A transaction that can only be executed by the University Participant
 */
transaction issueCertificate {
  o String studentName
  o String programName
}

/**
 * Approve affiliation of College to the University
 * A transaction that can only be executed by the University Participant
 */
transaction approveAffiliation {
    o String memberId
}
```

Step 2.6: Also define student & verifier Participant and Transactions in corresponding .cto model files:

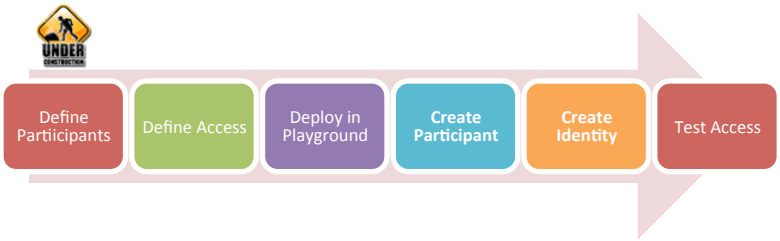
'org.gryphon.casestudy.student.cto'




```
namespace org.gryphon.casestudy.student
import org.gryphon.casestudy.base.*

/**
 * A College Participant that extends the participant base
 */
participant Student identified by memberId extends participantBase {
    o DateTime dob
    o String collegeName
    o String programName
    o String certificateId optional
}

/**  Student Specific transactions **/

/**
 * Student enrolls to a college and program
 */
transaction enrollStudent {
    o String name
    o DateTime dob
    o String collegeName
    o String programName
}
```

'org.gryphon.casestudy.verifier.cto'

```
namespace org.gryphon.casestudy.verifier
import org.gryphon.casestudy.base.*

/**
 * A Verifier who has only readonly attribute to Certificate Asset
 */
participant Verifier identified by memberId extends participantBase {
}

/**  Verifier Specific transactions **/

/**
 * Anyone can verify if the authenticity of the certificate by its ID
 */
transaction verifyCertificate {
    o String certificateId
}
```



Step 3: Define chaincode corresponding to the respective transactions defined.

** **Note:** Writing chaincode was shown in previous chapter. Here we define chaincode for remaining transactions

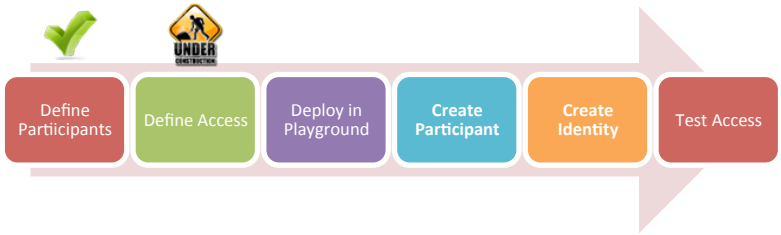
```
 * @transaction
 */
function approveAffiliation(args) {
    var registry;
    return getParticipantRegistry(NS_college + '.College')
        .then(function (assetRegistry) {
```

```
 */
function enrollProgram(args) {
    var registry;
    return getParticipantRegistry(NS_college + '.College')
        .then(function (assetRegistry) {
            registry = assetRegistry;
```

```
 */
function enrollStudent(args) {
    return getParticipantRegistry(NS_student + '.Student')
        .then(function (assetRegistry) {
            var factory = getFactory();
            var studentId = 'Student-' + Date.now().toString();
```

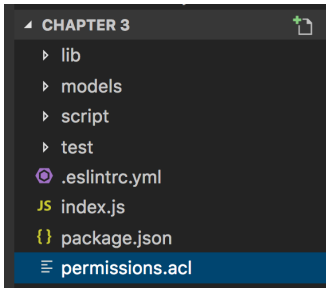
```
 * @transaction
 */
function requestAffiliation(args) {
    return getParticipantRegistry(NS_college + '.College')
        .then(function (assetRegistry) {
            var factory = getFactory();
            var collegeId = 'College-' + Date.now().toString();
```

Task 1 is complete!



TASK#2: Defining Access Control

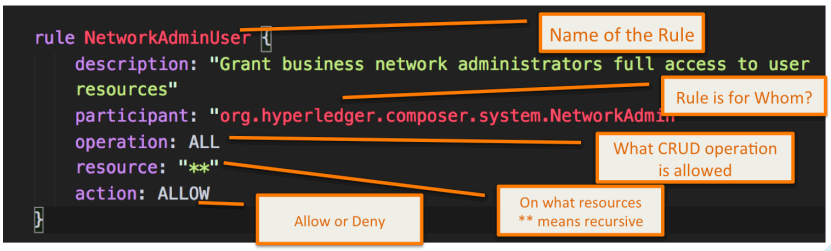
Step 1: Open 'chapter03' code provided in Visual Studio Code and Open *'permissions.acl'* file

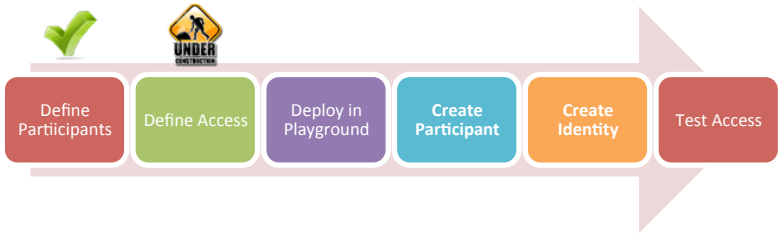


Step 1: Define permissions for NetworkAdmin User.

#Info: *If '.acl' is NOT defined all users have full permissions, however once '.acl' is defined all permissions are revoked and you need to define permissions even for the NetworkAdmin in order for the Admin to perform deployment and administrative functions.*

Step 1.1: Grant business network administrators full access to user resources





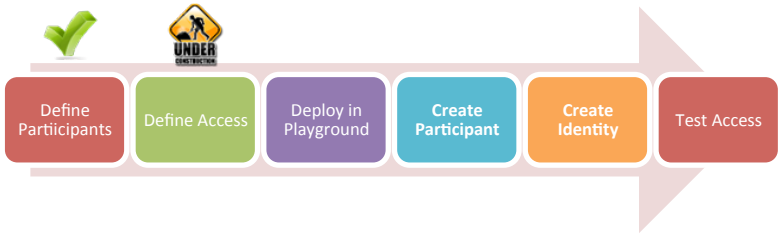
Step 1.2: Grant business network administrators full access to system resources

```
rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system
    resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

Step 2: Now that the NetworkAdmin access to all system and user resources are provided, we provide access to the Participants in our example. The intension here is to provide isolation, so that the Participant can carry out operations / transactions related to their authority only. All other operations/transactions should be restricted.

Step 2.1: Grant College Participant full access to resources in college namespace and restrict access to resources in other namespace

```
/*** Define College Participant Access ***/
rule CollegeResourceAccess {
    description: "Grant College Participant full access to resources in
    college namespace"
    participant: "org.gryphon.casestudy.college.College"
    operation: ALL
    resource: "org.gryphon.casestudy.college.**"
    action: ALLOW
}
```

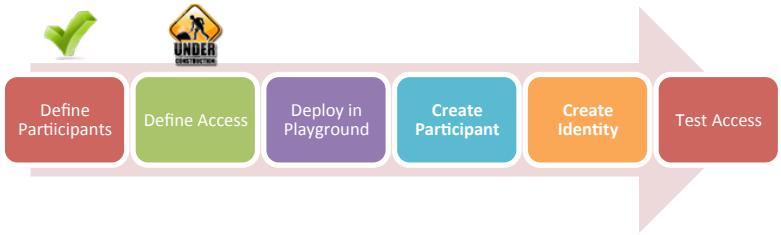


Step 2.2: Grant College Participant readonly access to system resources. This will ensure we are able to use Playground to read status of all resources and we can test our ACL file

```
/*** Required for Playgroud to view all resources ***/
rule CollegeSystemReadOnly {
    description: "Grant College Participant readonly access to system
    resources"
    participant: "org.gryphon.casestudy.college.College"
    operation: READ
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

Step 2.3: Grant access to perform transactions. Any transaction will create an entry in the historian, as transaction record is immutable. We need to provide this access to our College Participant so that it can perform appropriate transactions '*requestAffiliation*' and '*enrollProgram*'.

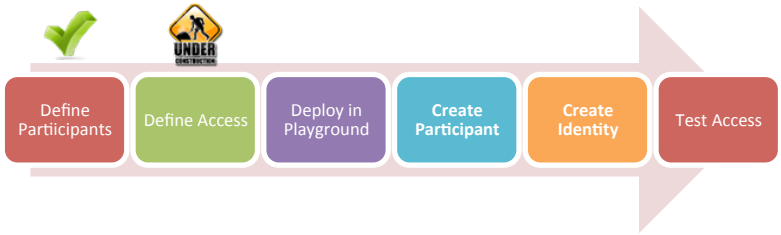
```
/*** Required to execute any Transaction ***/
rule CollegeSystemHistorianCreate {
    description: "Grant access to perform transactions"
    participant: "org.gryphon.casestudy.college.College"
    operation: CREATE
    resource: "org.hyperledger.composer.system.HistorianRecord"
    action: ALLOW
}
```



Step 3: Similarly we define access to other Participants:

Step 3.1: For Student Participant

```
/*** Define Student Participant Access ***/
rule StudentResourceAccess {
    description: "Grant Student Participant full access to resources in
    college namespace"
    participant: "org.gryphon.casestudy.student.Student"
    operation: ALL
    resource: "org.gryphon.casestudy.college.**"
    action: ALLOW
}

/*** Required for Playgroud to view all resources ***/
rule StudentSystemReadOnly {
    description: "Grant Student Participant readonly access to system
    resources"
    participant: "org.gryphon.casestudy.student.Student"
    operation: READ
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}

/*** Required to execute any Transaction ***/
rule StudentSystemHistorianCreate {
    description: "Grant access to perform transactions"
    participant: "org.gryphon.casestudy.student.Student"
    operation: CREATE
    resource: "org.hyperledger.composer.system.HistorianRecord"
    action: ALLOW
}
```



Step 3.1: For University Participant, we need to update College Registry and also Student Registry. Hence for simplicity we will provide them access to all user resources. In actual we should explicitly define what operations, on what resources needs to be granted.

```
/*** Define University  Access ***/
rule UniversityResourceAccess {
    description: "Grant University Participant full access to resources in
    all namespaces"
    participant: "org.gryphon.casestudy.university.University"
    operation: ALL
    resource: "org.gryphon.casestudy.**"
    action: ALLOW
}

/*** Required for Playgroud to view all resources ***/
rule UniversitySystemReadOnly {
    description: "Grant University Participant readonly access to system
    resources"
    participant: "org.gryphon.casestudy.university.University"
    operation: READ
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}

/*** Required to execute any Transaction ***/
rule UniversitySystemHistorianCreate {
    description: "Grant access to perform transactions"
    participant: "org.gryphon.casestudy.university.University"
    operation: CREATE
    resource: "org.hyperledger.composer.system.HistorianRecord"
    action: ALLOW
}
```

Task 2 is complete!

TASK#3: Deploy in Playground

Step 1: Open 'chapter03' code provided in Visual Studio Code. Open terminal window and cd into the script' directory

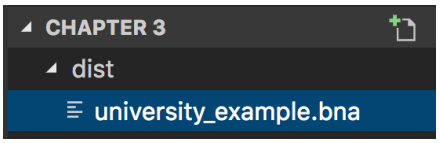
```
cd script
```

Step 2: Create Business Network Archive using the provided scripts; Type the following command in terminal window to run the archive script

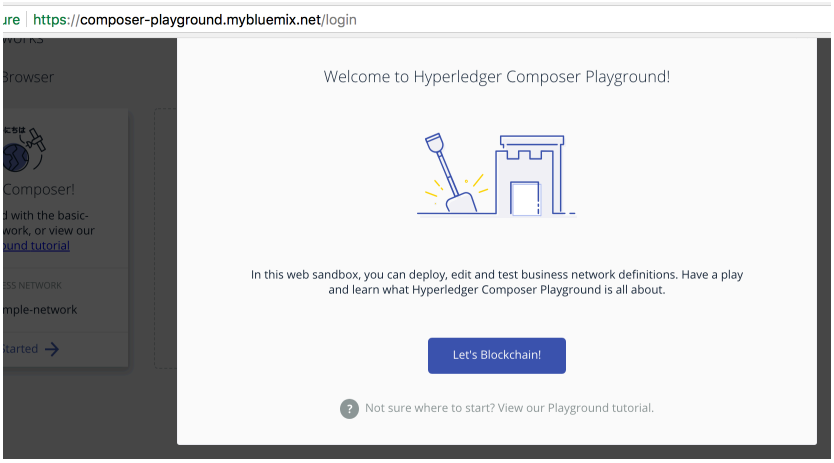
```
./createArchive.sh
```

```
==================================================
Creating Business Network Archive


Looking for package.json of Business Network Definition
        Input directory:                                    /Chapter 3

Found:
        Description: Creating models for our university example
        Name: university_example
        Identifier: university_example@0.0.1

Written Business Network Definition Archive file to
        Output file: ./dist/university_example.bna

Command succeeded
```

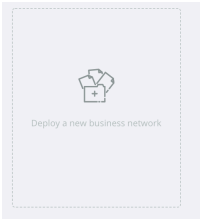
On successful completion of the command a *'university_example.bna'* is created in the 'dist' folder



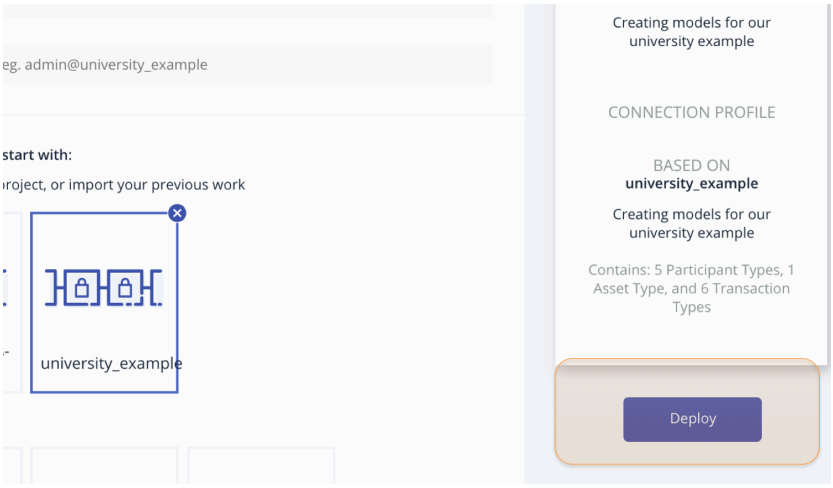
Step 3: Launch internet browser and goto the following URL:
https://composer-playground.mybluemix.net



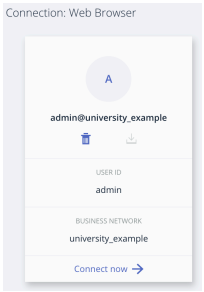
Step 4: Click 'Deploy Network' as below



Step 5: Choose 'Drop File to Upload' option and upload the recently created ***'university_example.bna'*** file from the ***'dist'*** folder.



Step 6: Choose 'Deploy' button to deploy the ***'.bna'*** file

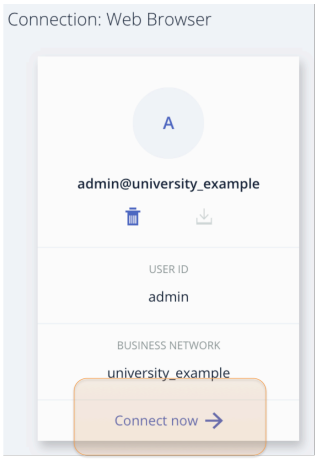


NetworkAdmin User is created on deployment of business network.

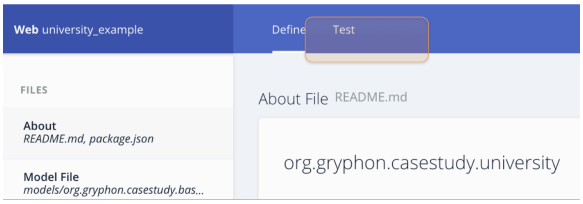
Task 3 is complete!

TASK#4: Create Participants

Step 1: Business Network Archive was deployed in previous task to Playground. Use the NetworkAdmin User created on deployment to connect to the network.

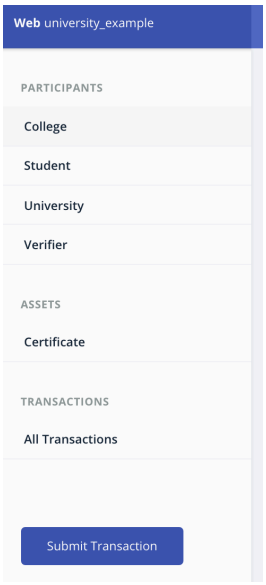


Step 2: Click on 'Test' tab to start interacting with the business network

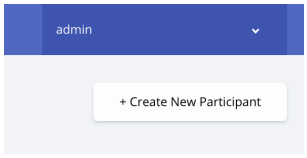


Step 3: Creating College participants:

Step 3.1: To create College Participant, click on 'College' Tab under participant list

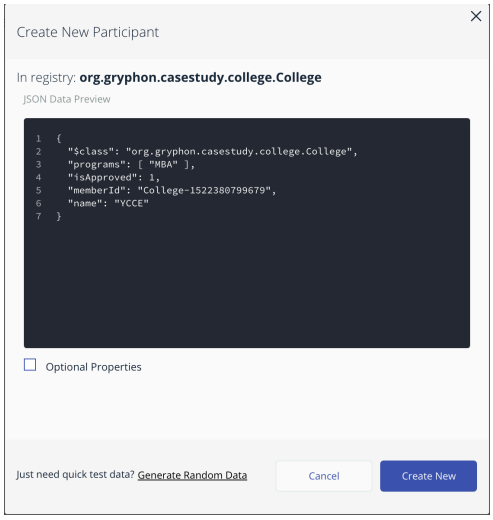


Step 3.2: Then use 'Create new Participant' button on the top right to add participant details



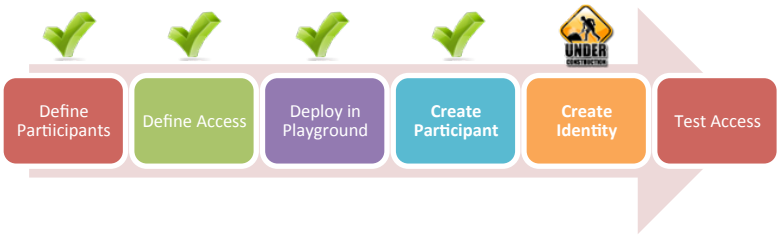
Step 3.3: Use following details to create a new participant

```
{
  "$class": "org.gryphon.casestudy.college.College",
  "programs": [ "MBA" ],
  "isApproved": 1,
  "memberId": "College-1522380799679",
  "name": "YCCE"
}
```



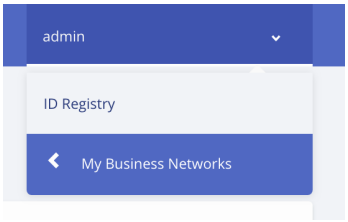
On completion a new College Participant is created.

Task 4 is complete!

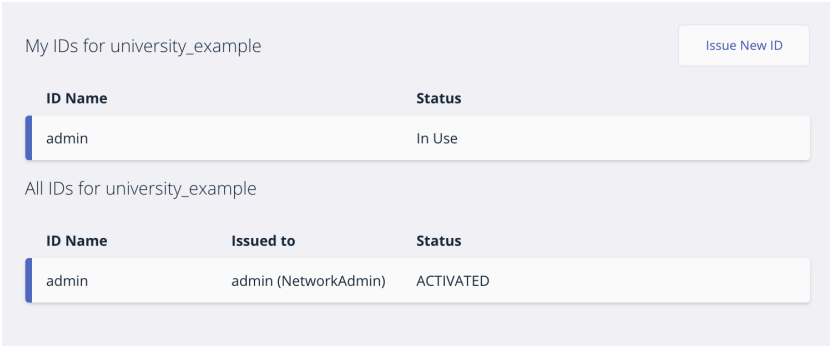


TASK#5: Create Identity

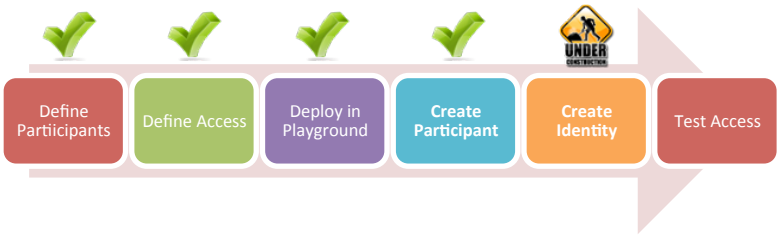
Step 1: Business Network Archive was deployed and 'College' Participant was created in previous task. Use the dropdown button on top right with the admin label and click on 'ID Registry'

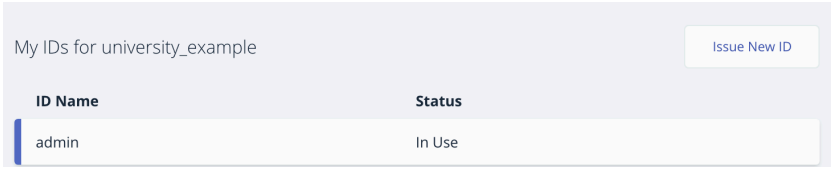


'ID Registry' contains the entries of all the Participants and associated Identities. New Identities for the participant can be created or earlier identities can be revoked from here.

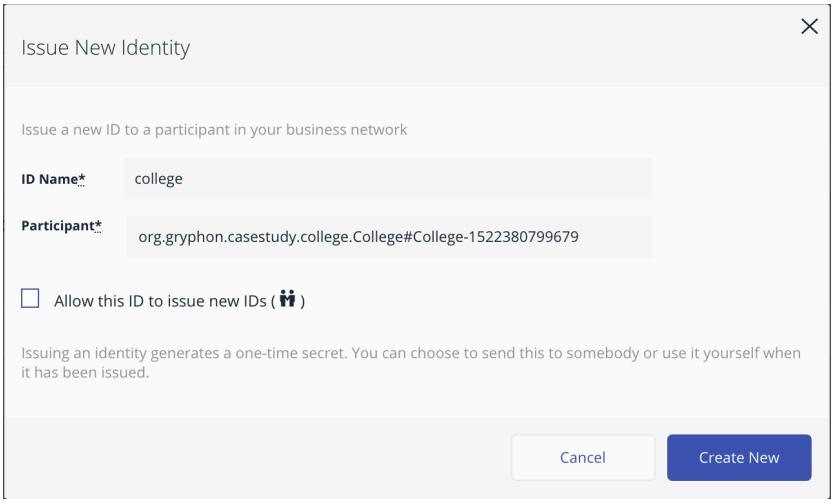


Step 2: Click on 'Issue New ID'

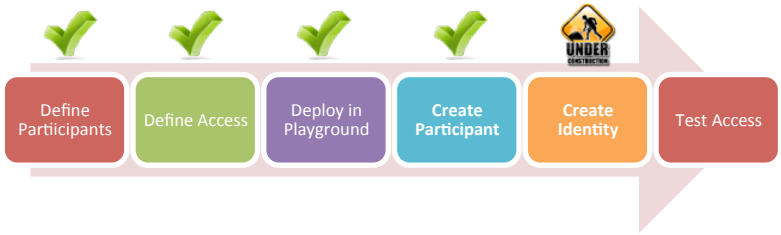


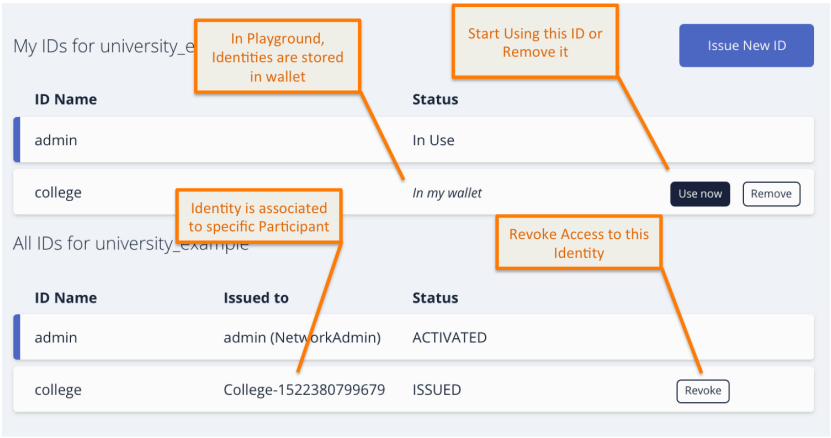


Step 3: Enter a name to the identity and also specify to which Participant identity has to be issued. In our case a college participant with ID: ***College-1522380799679***

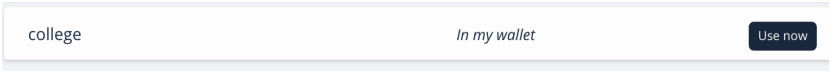


A new Identity has been created for the College Participant. Listing as shown below;

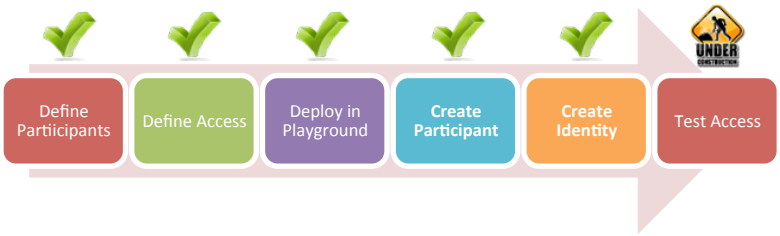




Step 4: To use the newly created identity for College participant, click “Use now” button against the identity.

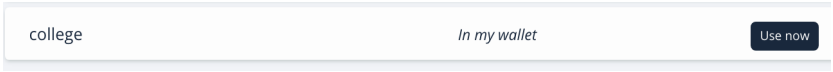


Task 5 is complete!

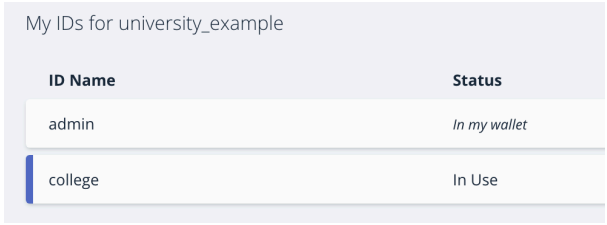


TASK#6: Test Access

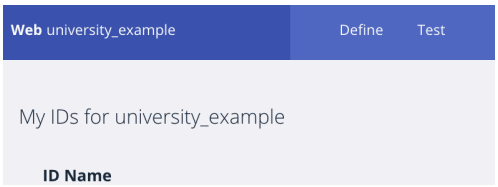
Step 1: Business Network Archive was deployed, 'College' Participant and its associated identity was created in previous task. We need to use this newly created Identity to test the access. Click 'Use now'.

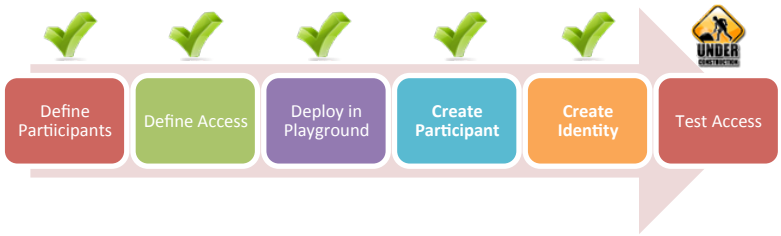


'admin' ID's status will start showing as 'In my wallet' and college ID will be 'In Use', indicating that our new Identity is at work.



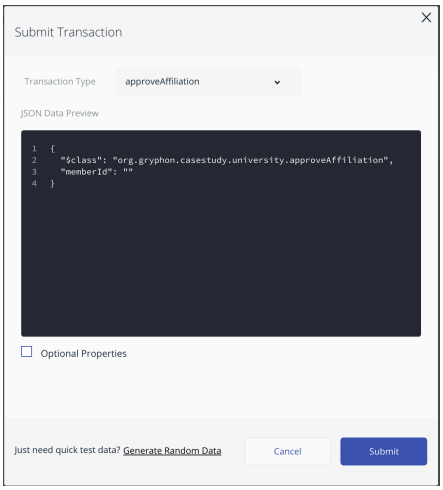
Step 2: Click on the 'Test' tab to navigate to the Playground





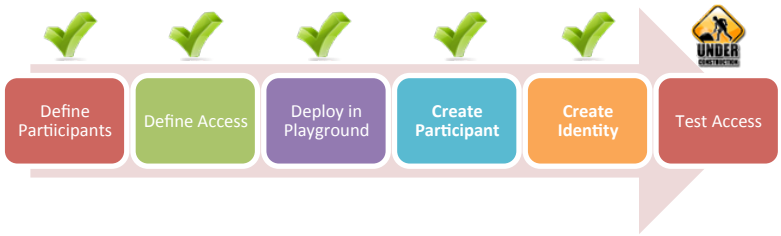
Step 3: We can invoke transaction using the 'Submit Transaction' button on left bottom of the 'Test' page

On clicking the button, transaction dialog shall appear;

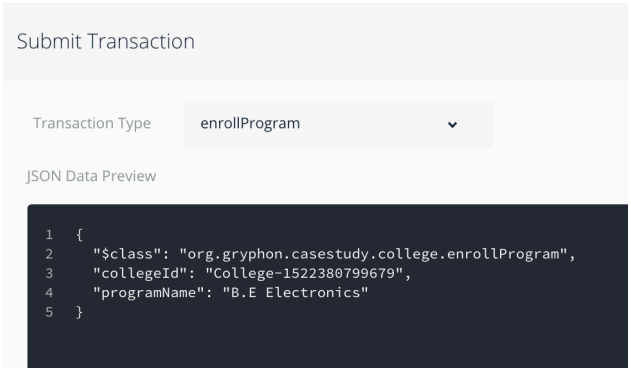


Step 4: Testing a **permissioned** transactions [***'requestAffliation'*** & ***'enrollProgram'***]

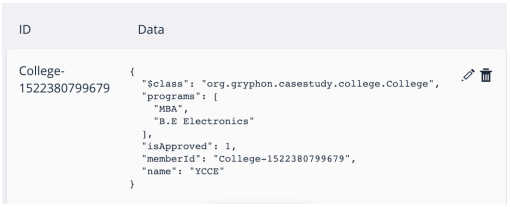
Step 4.1: In submit transaction dialog select transaction type as ***'enrollProgram'*** and use the following json to add "B.E Electronics" program to the existing college 'YCCE' with ID: College-1522380799679

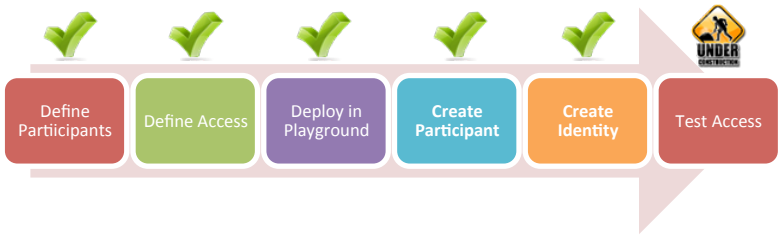



```
{
  "$class": "org.gryphon.casestudy.college.enrollProgram",
  "collegeId": "College-1522380799679",
  "programName": "B.E Electronics"
}
```



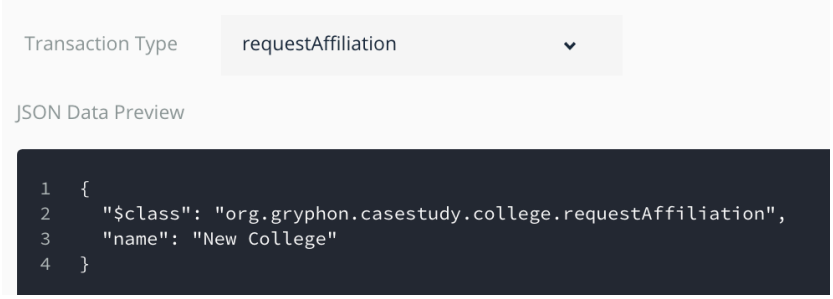
Step 4.2: Click Submit. On successful completion, we should be able to see a second program 'B.E Electronics' in the list of programs for the specific college



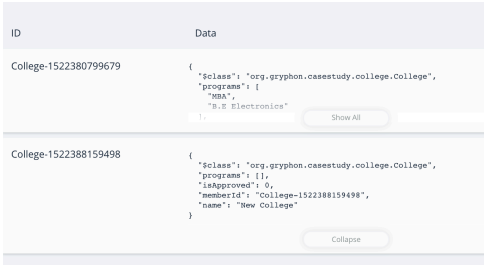


Step 4.3: Similarly Submit a new ***'requestAffliation'*** transaction using argument data as below;

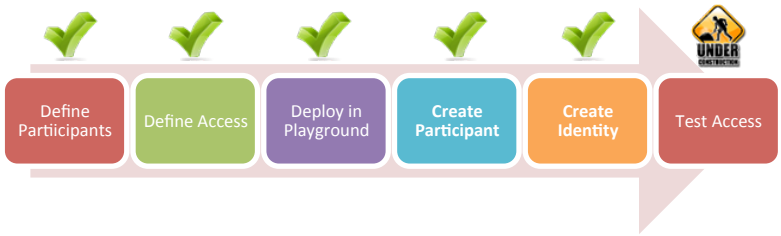
```
{
 "$class": "org.gryphon.casestudy.college.requestAffiliation",
 "name": "New College"
}
```



On successful execution a new college participant will be created.



#Result: *We are able to successfully execute chaincode transactions within namespace org.gryphon.casestudy.college and access as specified in 'permissions.acl' file*

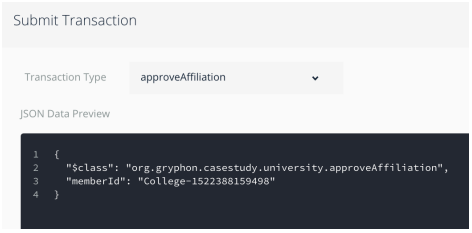


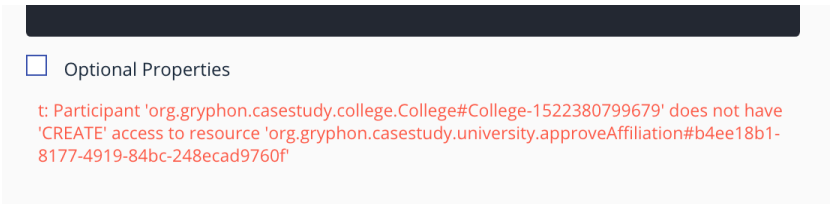
Step 5: Testing a **restricted permissioned** transactions [***'approveAffliation'*** & ***'issueCertificate'***]

Step 5.1: Let us try to execute ***'approveAffliation'*** transaction which only University is authorized as per our 'permissions.acl' file and see if permission are granted as expected;

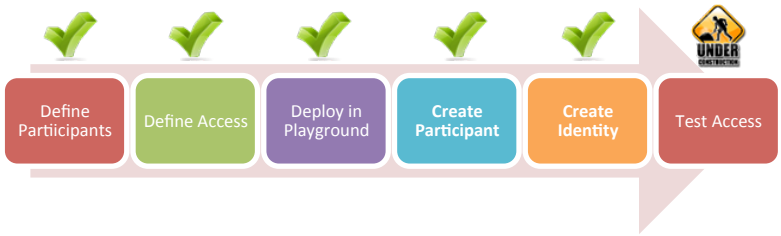
Lets approve 'New College' participant;

***** Note:*** *Please use college ID that is generated by your system for 'New College' participant*

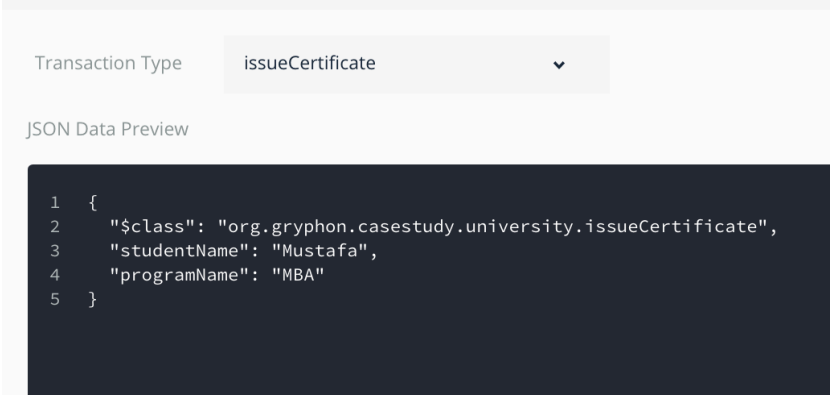




Transaction fails, showing that college participant does not have appropriate permissions



Step 5.2: Similarly Let us try to execute ***'issueCertificate'*** transaction which only University is authorized as per our 'permissions.acl' file and validate our permissions;



On submission of transaction we get permissions error as below;

#Result: *College Participant access to chaincode transactions are restricted only within namespace org.gryphon.casestudy.college. Hyperledger is a private permissioned network and access to the network can be provided/revoked using identities and selective permissions are defined in '.acl' file.*

Task 6 is complete!

Test Your Knowledge

In Hyperledger Fabric business network application, access is granted to both known and unknown entities?
- TRUE
- FALSE

In which file do you define access permissions?
- .ACL file
- .CTO file
- .json file
- .yml file

Access to Hyperledger Fabric Business Network Application is granted based on the _____ of the participant
- Role
- Identity
- Documents

Participants are assigned an identity using:
- User Name / Password
- X509 Certificate
- SSO

Apart from Participants, are Organization and Hyperledger Infrastructure also issued a X509 certificate?
- True
- False

SUMMARY

Hyperledger is a permissioned network. To be able to access resources or perform transactions network business cards/identities are needed.

In this chapter we have learnt the following:

- Participant & and their identities
- Adding a new participant to a participant registry
- Issuing a new identity to a participant
- Binding an existing identity to a participant
- Defining permissions to participant users using ACL (Access control Language)

CHAPTER 4: HYPERLEDGER – CLIENT APP

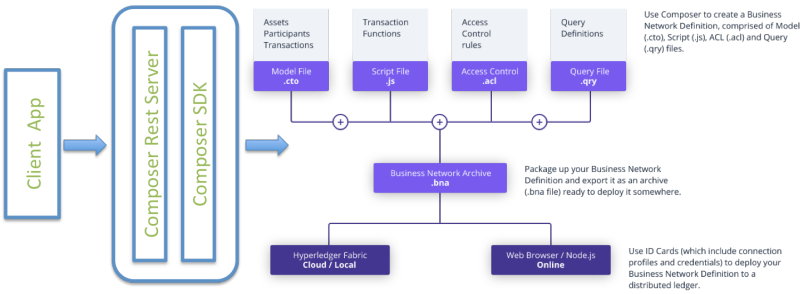
Theory

This chapter will be more hands-on. It will cover how we can build the client app, which can interaction with the fundamental blocks of Hyperledger blockchain i.e;

- Connecting to the Business Network
- Access Participant and Asset registries
- Invoke Transactions
- Query Resources (Static & Dynamic Queries)
- Handling events

High Level Architecture

Model file, Transaction functions (chaincode), access control file and the static query file make the Business Network Archive (Package) for the Hyperledger Fabric.



We will use Node.js and composer SDK to connect to the Fabric network and access/interaction with the blockchain resources.

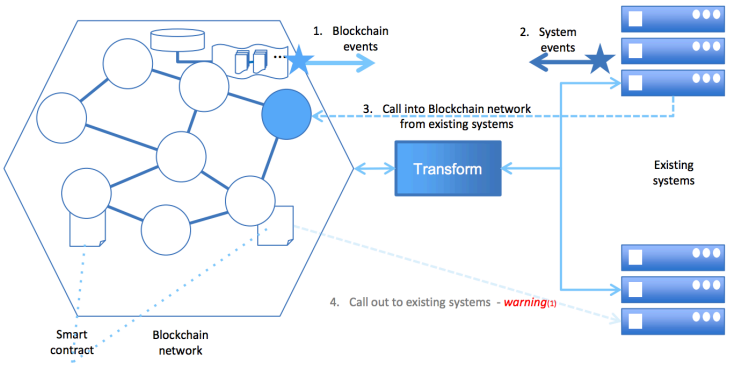
Queries:

The native query language can filter results returned using criteria and can be invoked in transactions to perform operations, such as updating or removing assets on result sets.

Queries are defined in a query file (.qry) in the parent directory of the business network definition. Queries contain a WHERE clause, which defines the criteria by which assets or participants are selected.

Events

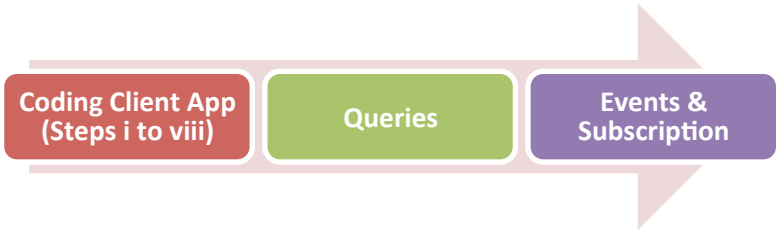
Event creates notifications of significant operations on the Blockchain (e.g. a new block), as well as notifications related to milestone achieved while processing a smart contract/chaincode. Does not include event distribution.



The client app can subscribe to this event and take appropriate business actions. It's an important part of any system to provide an insight ask for immediate attentions.

In the lab section we will generate an event in the chaincode and client app will subscribe for this event. We can then utilize this event to either send an email or appropriately notify the participant.

Lab Exercise 4: Coding client App, Queries & Events



1. **Coding Client App**
    i. **requestAffiliation()**
    ii. **getCollegeList()**
    iii. **approveAffiliation()**
    iv. **enrollProgram()**
    v. **takeAdmission()**
    vi. **getStudentList()**
    vii. **issueCertificate()**
    viii. **verifyCertificate()**
2. **Querying Registries**
3. **Events & Subscription**

TASK#1: Coding Client App

Step 1: Open 'chapter04' code provided in Visual Studio Code.

*#**Info**: 'chapter04' is a Business Network Scaffolding created as in Chapter#2-Task 1 with following changes:*

1. *Chaincode 'logic.js' is replaced with 'chaincode.js'*
2. *Test 'logic.js' is replaced with 'chaincode-test.js'*
3. *'script' folder is added*
4. *'permissions.acl' added*
5. *'client_app' folder added*

Step 2: Open terminal window of Visual Studio code and type run the following command

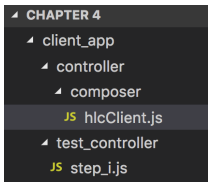
```
npm install
```

Step 3: Cd in the scripts folder and run buildAndDeploy.sh script to deploy the business network [Models and chaincode]

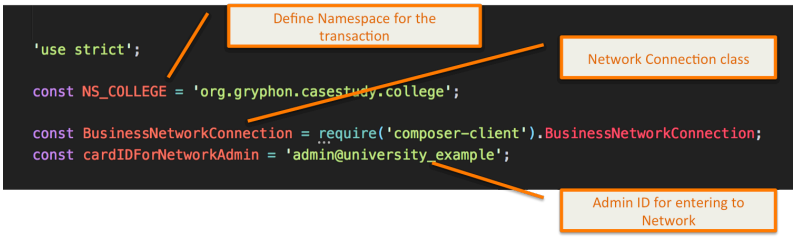
```
cd script
./buildAndDeploy.sh
```

TASK#1.1: Coding requestAffiliation()

Step 1: Browse to directory 'client_app/controller/composer' and open file 'hlcClient.js

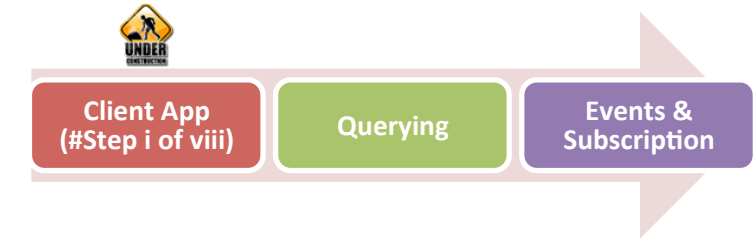


Step 2: Coding requestAffiliation() method



Actions:

- Create Business Network object
- Connect to network using permissioned card name (Participant or Admin) depending on usecase
- Create Transaction Object
- Set Transaction Arguments
- Submit Transaction
- Handle failure conditions by logging error info




```
exports.requestAffiliation = function (req, res,
    console.log('..... requestAffiliation .....

    let college_name = req.body.college_name;
    console.log("req: " + college_name);
    let businessNetworkConnection = new BusinessNetworkConnection();
    return businessNetworkConnection.connect(cardIDForNetworkAdmin)
        .then(() => {
            let factory = businessNetworkConnection.getBusinessNetwork().getFactory();

            const requestAffiliation = factory.newTransaction(NS_COLLEGE, 'requestAffiliation');
            requestAffiliation.name = college_name;
            return businessNetworkConnection.submitTransaction(requestAffiliation)
                .then(() => {
                    res.send({ "result": "Success" });
                })
                .catch((error) => {
                    console.log('requestAffiliation Transaction failed: text', error.message);
                    res.send({ 'result': 'failed',
                    'error': ' failed on requestAffiliation transaction ' + error.message });
                });
        })
        .catch((error) => {
            console.log('Business network connection failed: text', error.message);
            res.send({ 'result': 'failed',
            'error': ' requestAffiliation failed on on business network connection ' + error.message });
        });
};
```


Step 3: To validate if requestAffiliation() works, open terminal in visual studio code and cd into following directory

```
cd client_app/test_controller/
```

Step 4: Run the test script using nodejs as below

```
node step_i.js
```

The result success signifies that our transaction was successful

```
..... requestAffiliation .......
req: #CollegeNAME
{ result: 'Success' }
```

Task 1 #i is complete!

TASK#1.2: Coding getCollegeList()

Step 1: Browse to directory 'client_app/controller/composer' and open file 'hlcClient.js

```
exports.getCollegeList = function (req, res, next) {
    console.log('..... College List .......');
    let allColleges = new Array();
    let businessNetworkConnection = new BusinessNetworkConnection();
    return businessNetworkConnection.connect(cardIDForNetworkAdmin)
        .then(() => {
            return businessNetworkConnection.getParticipantRegistry(NS_COLLEGE + '.College')
                .then(function (registry) {
                    return registry.getAll()
                        .then((collegeList) => {
                            for (let each in collegeList) {
                                (function (_idx, _arr) {
                                    let _jsn = _arr[_idx];
                                    let jsn = { "id": _jsn.memberId,
                                    "name": _jsn.name,
                                    "is_approved": _jsn.isApproved,
                                    "programs": _jsn.programs };
                                    allColleges.push(jsn);
                                })(each, collegeList);
                            }
                            res.send({ 'result': 'success', 'college_lis
                        })
                        .catch((error) => {
                            console.log('error with getAll Colleges', error);
                            res.send({ 'result': 'false', 'college_list': [] });
                        });
                })
                .catch((error) => {
                    console.log('error with getCollegeList', error);
                    res.send({ 'result': 'false', 'college_list': [] });
                });
        })
        .catch((error) => { console.log('error with business network Connect', error); });
```

Connect to the Business Network

Get College Participant Registry

Contruct json array for return

Actions:

- Create Business Network object
- Connect to network using permissioned card name (Participant or Admin) depending on usecase
- Get College Participant Registry

- Get All participants
- On success, build a json and return
- On error Handle failure conditions by logging error info

Step 2: To validate if getCollegeList() works, open terminal in visual studio code and cd into following directory

```
cd client_app/test_controller/
```

Step 3: Run the test script using nodejs as below

```
node step_ii.js
```

The result success with list of colleges signifies that our transaction was successful

```
..... College List .......
{ result: 'success',
  college_list:
   [ { id: 'College-1523857299940',
       name: '#CollegeNAME',
       is_approved: 0,
       programs: [] },
     { id: 'College-1523882073365',
       name: '#CollegeNAME',
       is_approved: 0,
       programs: [] } ] }
```

Task 1 #ii is complete!

TASK#1.3: Coding approveAffiliation()

Step 1: Browse to directory 'client_app/controller/composer' and open file 'hlcClient.js


```
exports.approveAffiliation = function (req,
    console.log('..... approveAffiliation .

    let college_id = req.body.college_id;
    console.log("college_id: " + college_id);

    let businessNetworkConnection = new BusinessNetworkConnection();
    return businessNetworkConnection.connect(cardIDForNetworkAdmin)
        .then(() => {
            let factory = businessNetworkConnection.getBusinessNetwork().getFactory();

            const approveAffiliation = factory.newTransaction(NS_UNIVERSITY, 'approveAffiliation');
            approveAffiliation.memberId = college_id;
            return businessNetworkConnection.submitTransaction(approveAffil
                .then(() => {
                    res.send({ "result": "Success" });
                })
                .catch((error) => {
                    console.log(' approveAffiliation Transaction failed: te
                    res.send({ 'result': 'failed',
                    'error': ' failed on requestAffiliation transaction ' + error.message });
                });
        })
        .catch((error) => {
            console.log('Business network connection failed: text', error.
            res.send({ 'result': 'failed',
            'error': ' approveAffiliation failed on on business network connection ' + error.message });
        });
};
```

Actions:

- Create Business Network object
- Connect to network using permissioned card name (Participant or Admin) depending on usecase
- Create Transaction & assign input arguments
- Handle failure conditions by logging error info

Step 2: To validate if approveAffiliation() works, open terminal in visual studio code and cd into following directory

```
cd client_app/test_controller/
```

Step 3: Run the test script using nodejs as below to FETCH list of all colleges

```
node step_iii.js
```

The result success with list of colleges signifies that our transaction was successful

```
...... College List .......
{ result: 'success',
  college_list:
   [ { id: 'College-1523857299940',
       name: '#CollegeNAME',
       is_approved: 0,
       programs: [] },
     { id: 'College-1523882073365',
       name: '#CollegeNAME',
       is_approved: 0,
       programs: [] } ] }
```

Step 4: Note the **CollegeID** in red above and run the test below script using nodejs to approve affiliation of college

```
node step_iii.js ReplaceWithYourCollegeID
```

The result success with approved flag of specified college ID signifies that our transaction was successful

```
..... approveAffiliation .......
college_id: College-1523857299940
{ result: 'Success' }
..... College List .......
{ result: 'success',
  college_list:
   [ { id: 'College-1523857299940',
       name: '#CollegeNAME',
       is_approved: 1,
       programs: [] },
     { id: 'College-1523882073365',
       name: '#CollegeNAME',
       is_approved: 0,
       programs: [] } ] }
```

Task 1 #iii is complete!

TASK#1.4: Coding enrollProgram()

Step 1: Browse to directory 'client_app/controller/composer' and open file 'hlcClient.js

```
exports.enrollProgram = function (req, res, next) {
    console.log('...... enrollProgram .......');

    let college_id = req.body.college_id;
    let program_name = req.body.program_name;
    console.log("college_id: " + college_id);
    console.log("program_name: " + program_name);

    let businessNetworkConnection = new BusinessNetworkConnection();
    return businessNetworkConnection.connect(cardIDForNetworkAdmin)
        .then(() => {
            let factory = businessNetworkConnection.getBusinessNetwork().getFactory();

            const enrollProgram = factory.newTransaction(NS_COLLEGE, 'enrollProgram');
            enrollProgram.programName = program_name;
            enrollProgram.collegeId = college_id;
            return businessNetworkConnection.submitTransaction(enrollProgra
                .then(() => {
                    res.send({ "result": "Success" });
                })
                .catch((error) => {
                    console.log('enrollProgram Transaction failed: text', e
                    res.send({
                        'result': 'failed',
                        'error': ' failed on enrollProgram transaction ' + error.message
                    });
                });
        })
        .catch((error) => {
            console.log('Business network connection failed: text', error.m
            res.send({
                'result': 'failed',
                'error': ' enrollProgram failed on on business network connection ' + error.message
            });
        });
};
```

Fetch input arguments

Connect to the Business Network

getFactory() object for access to chaincode

Create Transaction & Assign arguments

Submit Transactions

Actions:

- Create Business Network object
- Connect to network using permissioned card name (Participant or Admin) depending on usecase
- Create Transaction & assign input arguments
- Handle failure conditions by logging error info

Step 2: To validate if enrollProgram() works, open terminal in visual studio code and cd into following directory

```
cd client_app/test_controller/
```

Step 3: Run the test script using nodejs as below

**Note: Use CollegeID that was generated in you usecase

```
  college_list:
   [ { id: 'College-1523857299940',
       name: '#CollegeNAME',
```

```
node step_iv.js College-1523857299940 M.B.A
```

The result success with list of program for the specified college signifies that our transaction was successful

```
...... enrollProgram .......
college_id: College-1523948141134
program_name: MBA
{ result: 'Success' }
...... College List .......
Programs Enrolled for College-1523948141134: MBA
```

Task 1 #iv is complete!

TASK#1.5: Coding takeAdmission()

Step 1: Browse to directory 'client_app/controller/composer' and open file 'hlcClient.js

```
exports.takeAdmission = function (req, res, next) {

    let student_name = req.body.student_name;
    let student_dob = req.body.student_dob;
    let college_name = req.body.college_name;
    let program_name = req.body.program_name;

    console.log("student_name: " + student_name);
    console.log("student_dob: " + student_dob);
    console.log("college_name: " + college_name);
    console.log("program_name: " + program_name);

    let businessNetworkConnection = new BusinessNetworkConnection();
    return businessNetworkConnection.connect(cardIDForNetworkAdmin)
        .then(() => {
            let factory = businessNetworkConnection.getBusinessNetwork().get

            const enrollStudent = factory.newTransaction(NS_STUDENT, 'enroll
            enrollStudent.name = student_name;
            enrollStudent.dob = new Date();
            enrollStudent.collegeName = college_name;
            enrollStudent.programName = program_name;
            return businessNetworkConnection.submitTransaction(enrollStudent
                .then(() => {
                    res.send({ "result": "Success" });
                    console.log(' enrollStudent Transaction Success......');
                })
                .catch((error) => {
                    console.log('enrollStudent Transaction failed: text', error.message);
                    res.send({ 'result': 'failed',
                    'error': ' failed on enrollStudent transaction ' + e
                });
        })
        .catch((error) => {
            console.log('Business network connection failed: text', error.message);
            res.send({ 'result': 'failed',
            'error': ' enrollStudent failed on on business network connection ' + error.message });
        });
};
```

Fetch input arguments

Connect to the Business Network

getFactory() object for access to chaincode

Create Transaction & Assign arguments

Submit Transactions

Actions:

- Create Business Network object
- Connect to network using permissioned card name (Participant or Admin) depending on usecase
- Create Transaction & assign input arguments
- Handle failure conditions by logging error info

Step 2: To validate if takeAdmission() works, open terminal in visual studio code and cd into following directory

```
cd client_app/test_controller/
```

Step 3: Run the test script using nodejs as below

**Note: In actual scenario ID for Student, College and Program Name should be taken. However this is just a demo usecase hence we are simply taking names

```
node step_v.js "John Doe" CollegeNAME MBA
```

The result success signifies that our transaction was successful

```
  student_name: John Doe
  student_dob: Tue Apr 17 2018 12:30:36 GMT+0530 (IST)
  college_name: CollegeNAME
  program_name: MBA
  { result: 'Success' }
   enrollStudent Transaction Success......
```

Task 1 #v is complete!

TASK#1.6: Coding getStudentList()

Step 1: Browse to directory 'client_app/controller/composer' and open file 'hlcClient.js

```
exports.getStudentList = function (req, res, next) {
    console.log('..... Student List .......');

    let allStudents = new Array();
    let businessNetworkConnection = new BusinessNetworkConnection();
    return businessNetworkConnection.connect(cardIDForNetworkAdmin)
        .then(() => {
            return businessNetworkConnection.getParticipantRegistry(NS_STUDENT + '.Student')
                .then(function (registry) {
                    return registry.getAll()
                        .then((studentList) => {
                            for (let each in studentList) {
                                (function (_idx, _arr) {
                                    let _jsn = _arr[_idx];
                                    let jsn = {
                                        "id": _jsn.memberId,
                                        "name": _jsn.name,
                                        "certificateId": _jsn.certificateId,
                                        "program": _jsn.programName
                                    };
                                    allStudents.push(jsn);
                                })(each, studentList);
                            }
                            res.send({ 'result': 'success', 'student_list': allStudents });
                        })
                        .catch((error) => { console.log('error with get
                })
                .catch((error) => { console.log('error with getStudentList
        })
        .catch((error) => { console.log('error with business network Connect', error); });
};
```

Connect to the Business Network

Fetch Participant Registry

Make return json object

Send Response Data

Actions:

- Create Business Network object
- Connect to network using permissioned card name (Participant or Admin) depending on usecase
- Create Transaction & assign input arguments
- Handle failure conditions by logging error info

Step 2: To validate if getStudentList() works, open terminal in visual studio code and cd into following directory

```
cd client_app/test_controller/
```

Step 3: Run the test script using nodejs as below

```
node step_vi.js
```

The result success signifies that our transaction was successful

```
..... Student List .......
{ result: 'success',
  student_list:
   [ { id: 'Student-1523948446529',
       name: 'John Doe',
       certificateId: undefined,
       program: 'MBA' } ] }
```

Task 1 #vi is complete!

TASK#1.7: coding issueCertificate()

Step 1: Browse to directory 'client_app/controller/composer' and open file 'hlcClient.js

```
exports.issueCertificate = function (req, res, next) {
    console.log('..... issueCertificate .......');

    let student_id = req.body.student_id;
    console.log("student_id: " + student_id);

    let businessNetworkConnection = new BusinessNetworkConnection();
    return businessNetworkConnection.connect(cardIDForNetworkAdmin)
        .then(() => {
            return businessNetworkConnection.getParticipantRegistry(NS_STUDENT + '.Student')
                .then(function (registry) {
                    return registry.get(student_id)
                        .then((student) => {

                            let factory = businessNetworkConnection.getBusinessNetw

                            const issueCertificate = factory.newTransaction(NS_UNIVERSITY, 'issueCertificate');
                            issueCertificate.studentID = student_id;
                            issueCertificate.studentName = student.name;
                            issueCertificate.programName = student.programName;
                            console.log(student.name);
                            console.log(student.programName);
                            return businessNetworkConnection.submitTransaction(issueCertificate)
                                .then(() => {
                                    res.send({ "result": "Success" });
                                })
                                .catch((error) => {
                                    console.log('issueCertificate Transaction failed: text', error.message);
                                    res.send({ 'result': 'failed',
                                    'error': ' failed on issueCertificate transaction ' + error.message });
                                });
                        })
                        .catch((error) => {
                            console.log('issueCertificate Transaction failed: text'
                            res.send({ 'result': 'failed',
                            'error': ' failed on getParticipant registry' + error.message });
                        });
                })
                .catch((error) => {
                    console.log('issueCertificate Transaction failed: text', error.message);
```

Fetch input arguments

Connect to the Business Network

Get Student Participant Registry

Create Transaction & Assign arguments

Submit issueCertificate Transactions

Actions:

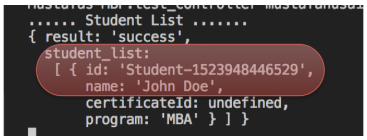
- Create Business Network object
- Connect to network using permissioned card name (Participant or Admin) depending on usecase
- Fetch Student Participant Registry
- Get Student Records
- Create issueCertificate Transaction & assign input arguments
- Handle failure conditions by logging error info

Step 2: To validate if issueCertificate() works, open terminal in visual studio code and cd into following directory

```
cd client_app/test_controller/
```

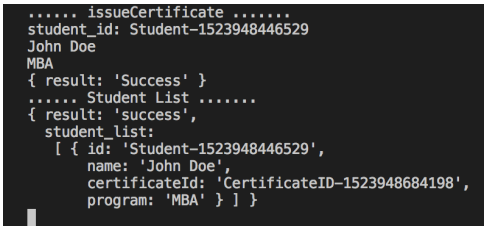
Step 3: Run the test script using nodejs as below

****Note**: Please use Student ID generated on your system




```
node step_vii.js Student-1523948446529
```

The result success signifies that our transaction was successful

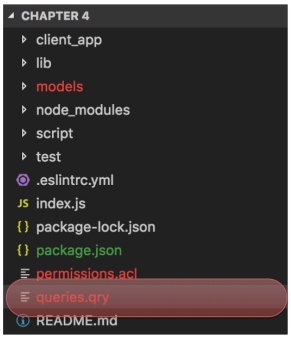


Task 1 #vii is complete!

TASK#2: Using Queries

Step 1: Open 'chapter04' code provided in Visual Studio Code.

Step 2: Open file 'queries.qry'



Step 3: Define query to find student by its ID

```
/**  quering the Certificate
*/

query selectCertificate {
  description: "Fetch a specific Certificate"
  statement:
      SELECT org.gryphon.casestudy.university.Certificate
          WHERE (certificateId == _$id)
}
```

Name of Query we will use it to call from client App

Description can be used for readability & maintenance

Resource to query

WHERE clause for filter

Step 4: Browse to directory 'client_app/controller/composer' and open file 'hlcClient.js

Step 5: Coding getCertificateById() client App method

```
exports.getCertificateById = function (req, res, next) {
    console.log('..... getCertificateById .......');

    let certificateId = req.query.id;
    console.log('certificateId is: ' + certificateId);

    let allCertificates = new Array();
    let businessNetworkConnection = new BusinessNetworkConnection();
    return businessNetworkConnection.connect(cardIDForNetworkAdmin)
        .then(() => {
            return businessNetworkConnection.query('selectCertificate',
                .then((certificateList) => {
                    for (let each in certificateList) {
                        (function (_idx, _arr) {
                            let _jsn = _arr[_idx];
                            let jsn = { "id": _jsn.certificateId,
                            "student": _jsn.issuedTo,
                            "issued_date": _jsn.issuedDate,
                            "program": _jsn.programName, };
                            allCertificates.push(jsn);
                        })(each, certificateList);
                    }
                    if (allCertificates.length > 0) {
                        res.send({ 'result': 'success', 'certificates': allCertificates });
                    } else {
                        res.send({ 'result': 'failed', 'certificates':
                    }
                })
                .catch((error) => { console.log('error with certificateById ', error); });
        })
        .catch((error) => { console.log('error with business network Connect', error); });
};
```

Fetch input arguments

Connect to the Business Network

Call Query with inputs as student ID

Process Results & build json object

Send the response json object

Actions:

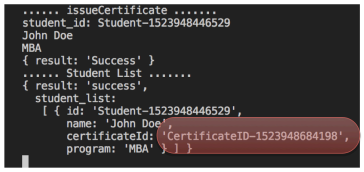
- Create Business Network object
- Connect to network using permissioned card name (Participant or Admin) depending on usecase
- Call the query using student ID as filter
- Build the response json object
- Handle failure conditions by logging error info

Step 6: To validate if getCertificateById () works, open terminal in visual studio code and cd into following directory

```
cd client_app/test_controller/
```

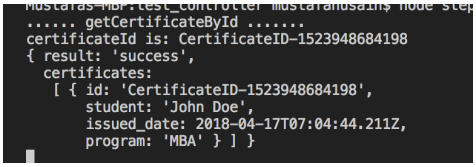
Step 7: Run the test script using nodejs as below

****Note**: Please use Certificate ID generated on your system




```
node step_viii.js CertificateID-1523948684198
```

The result success with details of certificate signifies that our transaction was successful

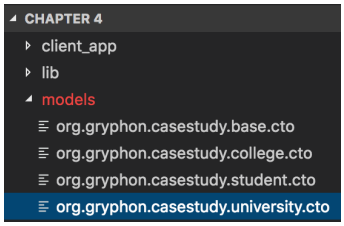


Task 2 is complete!

TASK#3: Blockchain Events & Subscription

Step 1: Open 'chapter04' code provided in Visual Studio Code.

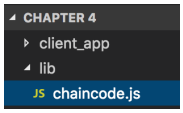
Step 2: Open file 'org.gryphon.casestudy.university.cto' under models folder



Step 3: Define Event

```
/**
 * Notify Issue of certificate
 * An event that can be substribed
 */
event certificateIssuedEvent {
  o String certificateId
}
```

Step 4: Browse to directory 'lib' and open file 'chaincode.js

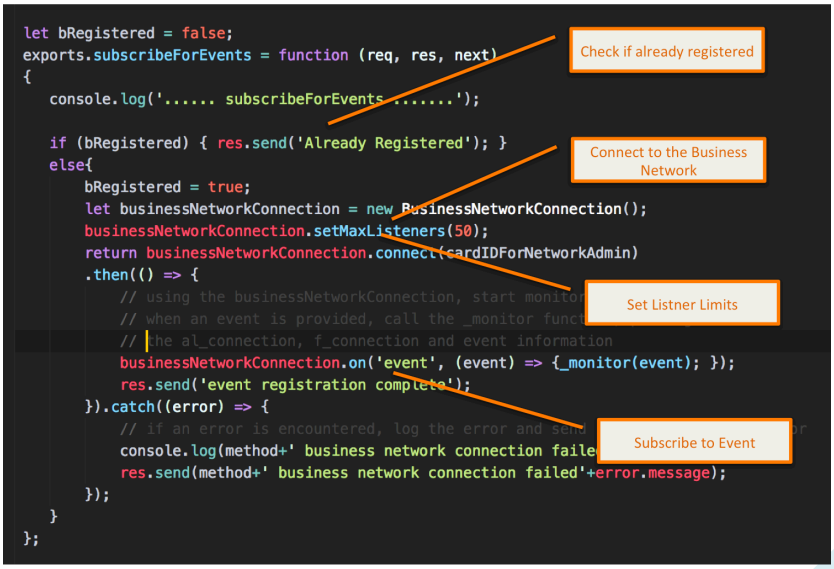


Step 5: Add Emit Event to the issueCertificate() chaincode

```
var event = factory.newEvent(NS_UNIVERSITY, 'certificateIssuedEvent');
event.certificateId = certificateId;
emit(event);
```

Step 6: Browse to directory 'client_app/controller/composer' and open file 'hlcClient.js

Step 7: Subscribing to Events. Coding subscribeForEvents() method



Actions:

- Create Business Network object
- Connect to network using permissioned card name (Participant or Admin) depending on usecase
- Set connection limits
- Subscribe to the events
- Handle failure conditions by logging error info

Step 8: Any blockchain event emitted will now call ***'_monitor()'***

```
function _monitor(_event)
{
    let method = '_monitor';
    console.log(method+ ' _event received: '+_event.$type);

    // Can use Socket.io to connect to UI
    // io.emit(_event.name, _event.value);

    if(_event.$type != 'certificateIssuedEvent') {
        // Probable Action: Fetch Certifcate & Student Details and Send Email
        // sendEmail(subject, message);
    }
}
```

- For all events same _monitor() method will be called
- We need to check if event called was 'certificateIssueEvent', we do it by peeking into _event.$type variable

Step 9: We will have to deploy the runtime again as the chaincode has changed. Browse into the 'script' folder

```
cd script
```

Step 10: Install the new chaincode by running the following script

```
./updateRuntime.sh
```

Step 11: To validate if Event Subscription works, open terminal in visual studio code and cd into following directory

```
cd client_app/test_controller/
```

Step 12: Run the test script using nodejs as below;

We will call issueCertificate() transaction so that the event is emitted

```
node step_x.js CertificateID-1523948684198
```

The result success with details of certificate and monitor method called signifies that our subscription was successful

```
Mustafas-MBP:test_controller mustafahusain$ node step_x.js
...... subscribeForEvents .......
event registration complete
...... issueCertificate .......
student_id: Student-1523948446529
John Doe
MBA
{ result: 'Success' }
_monitor _event received: certificateIssuedEvent
```

Task 3 is complete!

Test Your Knowledge

What class do you need to connect to Business Network?
- BusinessNetworkDefinition
- BusinessNetworkConnection
- FactoryClass
- BusinessNetworkCardStore

In which file do you define static queries?
- .ACL file
- .CTO file
- .QRY file
- .SH file

Client code cannot create an instance of the Admin & Business Network Connection in the same code
- True
- False

Execution of the Named query using the API require you to use the *businessNetworkConnection.buildQuery()* function:
- True
- False

A subscriber application using the composer API receives the event data in the form of
- JSON
- Array
- Message
- UTF8 String

CHAPTER 5: CREATING FRONT END INTERACTIVE INTERFACES

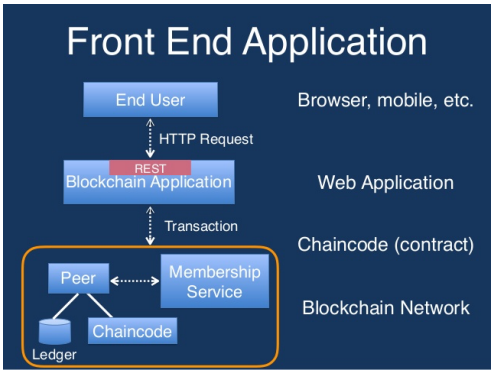
Theory

This chapter again will be completely hands-on. In this chapter we will build a complete end-to-end interactive usecase utilizing the University example that we have built so far. We will cover all the missing links and FAQs

- How do we invoke the chaincode from an HTML or a frontend application?
- How do we setup the node server and routes to run the client app?
- How does the bloackchain look like in a real work scenario?
- How can a end-user access the blockchain?

Front End Application Patterns

1. Composer Rest Server middleware Architecture:

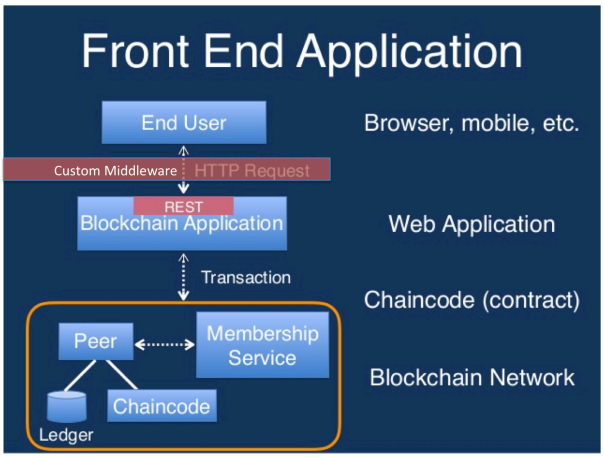


Considerations:

- Rest server must be secured – HTTPS
- Should use authentication – [Passport]
- Should use multi-user mode for rest api

2. Custom middleware pattern:

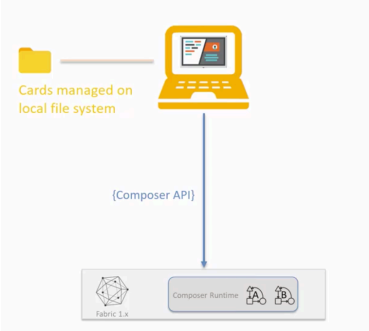
This is the pattern we will be using in our Demo usecase.



Advantages:

- More secure
- Better control and can plugin existing enterprise applications

3. Desktop Application Architecture



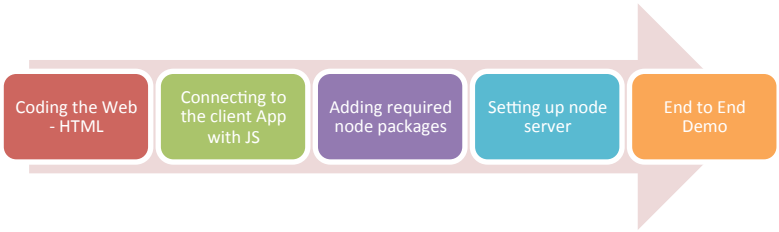
Pros:

- Most Secure

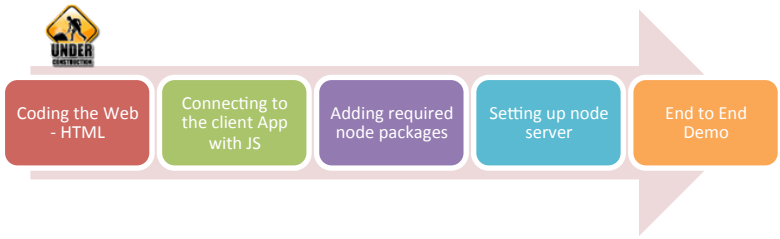
Cons:

- App distribution

Lab Exercise 5: Building Interactive Frontend



1. Coding the Web – HTML
    1. Directory Structure CSS and Script includes
    2. Coding HTML for different sections:
        i. University
        ii. College
        iii. Student
        iv. Verifier
        v. Historian
    3. Other Model Dialogs
2. Connecting to the client App with JS
    1. On load functions
    2. Javascript to connect HTML to client App
3. Adding required node packages
4. Setting up node server
5. End to End Demo



TASK#1: Coding the Web – HTML

Step 1: Open 'chapter05' code provided in Visual Studio Code.

*#**Info**: 'Chapter 5' is a Business Network Scaffolding as in Chapter#4.*

Additionally its has the following which we will build as a step by step process

1. *'html' folder which contains all the frontend code*
2. *Updated 'index.js' file*
3. *Updated 'pacakage.json' file*
4. *'router.js' added in 'client_app/controller' folder*

Step 2: Open terminal window of Visual Studio code and type run the following command

```
npm install
```

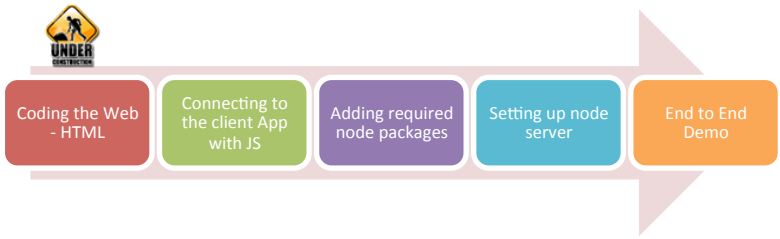
Step 3: Cd into the 'script' folder and run 'run.sh' script to start the node server so that we can launch the frontend for a preview

```
cd script
./run.sh
```

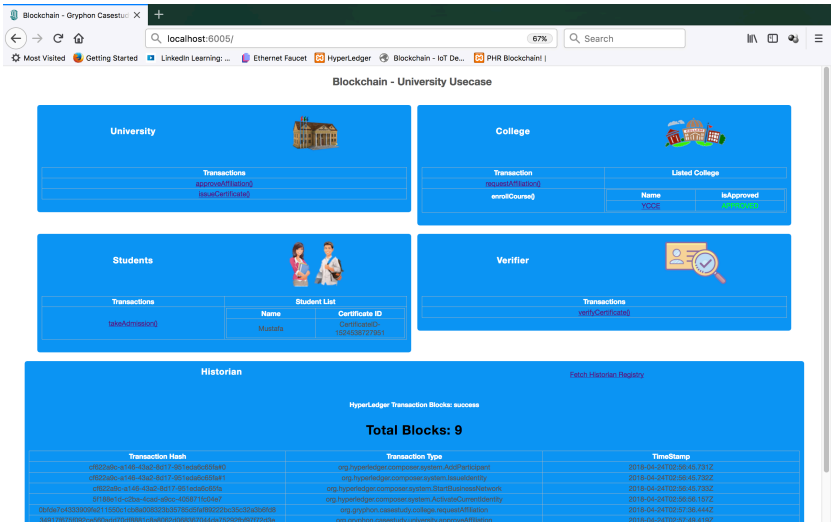
The node server will startup and show the port + address where you can view the frontend html pages in the browser.

```
[redacted]$ ./run.sh
Listening locally on port 6005
```

Step 4: Launch internet browser and type the URL as below with port number as displayed in previous step:



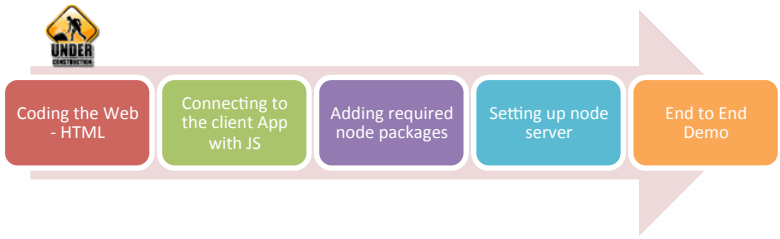
URL: localhost:6005



The UI has five sections, which we will build one by one:

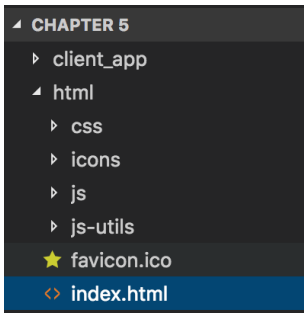
- University Section with Two clickable links
- College Section with clickable actions and table to display college list
- Student Section with clickable actions and table to display student list
- Verifier Section with a clickable link (Transaction) action
- Historian Section that displays all the Blocks/Transactions that have occurred so far

Let's build it step by step…



TASK#1.1: Directory Structure CSS and Script includes

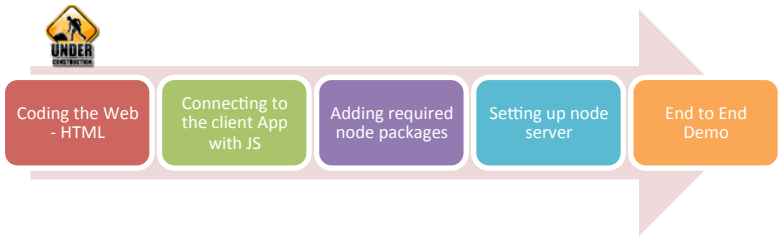
Step 1: Browse to directory 'html'



The 'html' folder has following contenst:

- Css folder:
  - Bootstrap CSS
  - Fonts CSS
  - Application CSS
- Icons:
  - Icons / Images used in our App Usecase
- Js:
  - Application JavaScript file: 'event.js'
- Js-utils:
  - JavaScript files for the 3rd party libraries like jquery and bootstrap
- Favicon.ico: Default icon for the browser (near url)
- Index.html: Default and primary html file for the frontend

Step 2: Under 'html' folder open the file 'index.html'



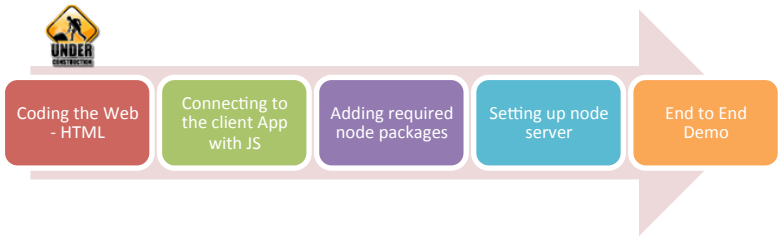
Step 2: Under '<head>' section of the file 'index.html', lets include the css files we need for our UI

```
<head>
    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Blockchain - Gryphon Casestudy University</title>
    <!-- build:css styles/main.css -->
    <link rel="stylesheet" href="css/main.css">
    <link rel="stylesheet" href="css/font-awesome.min.css"  type="text/css">
    <!-- endbuild -->
</head>
```

Step 3: Go to the End of the file 'index.html', and lets include the js files from both 'js' and 'js-util' folder

```
    <!-- build:js scripts/main.js -->
    <script src="js-utils/jquery-3.1.0.min.js"></script>
    <script src="js-utils/jquery-ui.js"></script>
    <script src="js-utils/bootstrap.min.js"></script>
    <script src="js/events.js"></script>
    <!-- endbuild -->

</body>
</html>
```

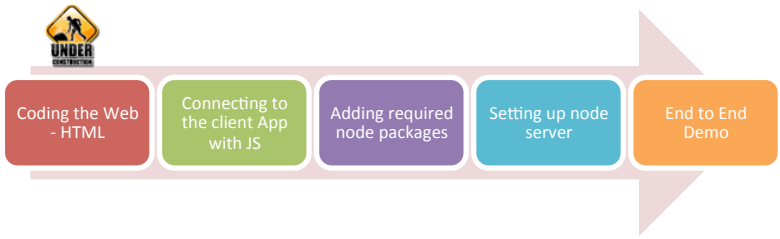
Task 1.1 is complete!



TASK#1.2: Coding HTML for different sections

Step 1: Browse to directory 'html' and open file 'index.html'. Look and code the university section


```
<!-- University Section -->
<div class="sidekick">
    <div>
        <table> <!-- Title & The University Logo -->
            <tr>
                <td> <h1>University</h1> </td>
                <td> <img src="./icons/uni.png"/> </td>
            </tr>
        </table>
    </div>
    <span/>
    <table id="tt"> <!-- Display Transactions that can be done by University -->
        <tr>
            <th>Transactions</th>
        </tr>
        <tr>
            <td> <a class="js-open-modal" href="#" data-modal-id="approveAffiliation">
                approveAffiliation()</a> </td>
        </tr>
        <tr>    <!-- Will Call Javascript Method "generateCertificate()" when link clicked -->
            <td> <a href="#" onClick="generateCertificate()">issueCertificate()</a> </td>
        </tr>
    </table>
</div>
```

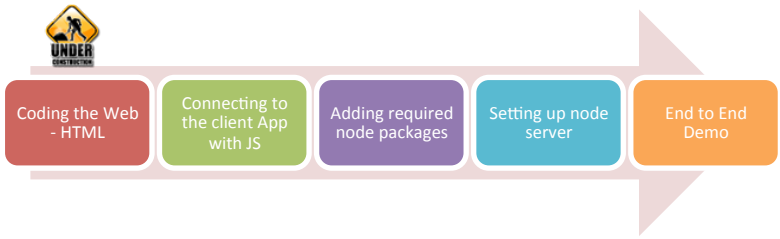


Step 2: Coding College Section

```
<!-- College Section -->
<div class="sidekick">
    <table>
        <tr> <!-- Title & The College Logo -->
            <td> <h1>College</h1> </td>
            <td> <img src="./icons/college.png"/> </td>
        </tr>
    </table>
    <span/>
    <!--div id='invoke_details'></div-->
    <div id="hash">
        <table id="tt"> <!-- Display Transactions that can be done by College -->
            <tr>
                <th>Transaction</th>
                <th>Listed College</th>
            </tr>
            <tr>
                <td> <a class="js-open-modal" href="#"
                    data-modal-id="requestAffiliation">requestAffiliation()</a> </td>
            </tr>
            <tr>
                <td> <h4>enrollCourse()</h4> </td>
                <td> <div id="college_list"></div> </td>
            </tr>
        </table>
    </div>
</div>
```

ID for Modal Dialog that will pop up onclick

College List will be dynamically generated & replaced in this division block

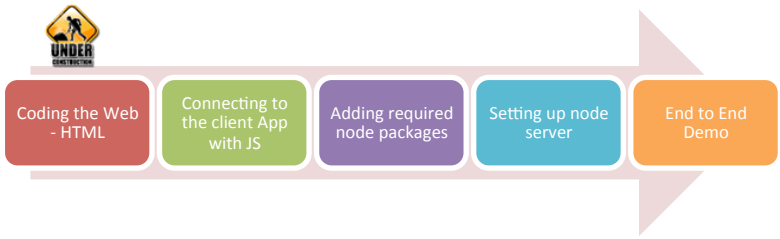


Step 3: Coding Student Section (Similar to College section with changes in Transaction function, images and corresponding texts)

```
<!-- Studen Section -->
<div class="sidekick">
    <table>  <!-- Title & The  Logo -->
        <tr>
            <td> <h1>Students</h1> </td>
            <td> <img src="./icons/stud.png"/> </td>
        </tr>
    </table>
    <span/>
    <table id="tt"> <!-- Display Transactions that can be done by Student -->
        <tr>
            <th>Transactions</th>
            <th>Student List</th>
        </tr>
        <tr>
            <td> <a href="#" onClick="takeAdmission()">takeAdmission()</a> </td>
            <td> <div id="student_list"></div> </td>
        </tr>
    </table>
</div>
```

Step 4: Similarly code the verifier Section

```
<!-- Verifier Section -->
<div class="sidekick">
    <table>
        <tr>
            <th> <h1>Verifier</h1> </th>
            <th> <img src="./icons/Verifier.png"/> </th>
        </tr>
    </table>
    <span/>
    <div id="hash">
        <table id="tt">
            <tr> <td>Transactions</td> </tr>
            <tr>
                <td> <a class="js-open-modal" href="#"
                    data-modal-id="verifyCertificate">verifyCertificate()</a><br/> </td>
            </tr>
        </table>
    </div>
</div>
<!-- Verifier Section End -->
```



Step 5: Coding the Last section i.e Historian that has a link that invokes the javascript and returns the complete history table and that is displayed in the div section with name **id="historian"**

```
<div class="sidekick" style="width:95%">
    <div>
        <table>
            <tr>
                <td> <h1>Historian</h1> </td>
                <td> <a href="#" onclick="getHistorian()">Fetch Historian Registry</a> </td>
            </tr>
        </table>
    </div>
    <span/>
    <div id="historian"></div>
</div>
```

Step 6: Coding Modal Dialogs

```
<!-- Modal -->
<div id="requestAffiliation" class="modal-box">
    <header>
        <a href="#" class="js-modal-close close">x</a>
        <h3>Request Affiliation</h3>
    </header>
    <div class="modal-body">
        <input type="text" id="college_name" placeholder="College Name"></input>
        <div id="college_progress"></div>
    </div>
    <footer>
        <a href="#" class="js-modal-btn" onclick="requestAffiliation()">Submit</a>
        <a href="#" class="js-modal-close">Close</a>
    </footer>
</div>
```

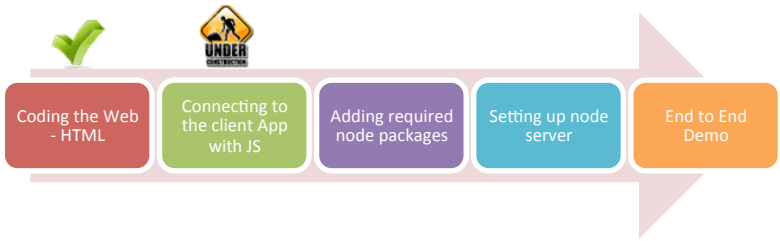
ID for Modal Dialog that will pop up onclick

Will be replaced by spinner image to show progress

Submit the Transaction

Step 7: Similarly coding for other Modal Dialogs

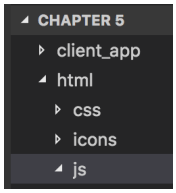
Task 1 is complete!



TASK#2: Connecting the Frontend to call Client App

Step 1: Open 'chapter05' code provided in Visual Studio Code. Browse to the directory 'html/js' and open file 'event.js'

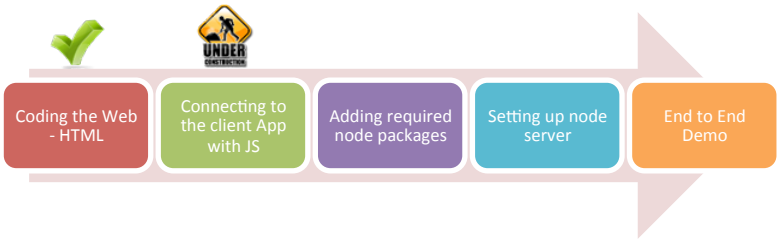
Step 2: Let's code the methods that we have declared with 'OnClick' in HTML file in Task#1.



Step 2: Coding requestAffiliation() method that invokes the Client App rest api as below;

```
function requestAffiliation() {
    // Show Progress untill the task is complete
    document.getElementById('college_progress').innerHTML = '<img src="./icons/progress.gif"/>';

    // Get the attributes from the UI i.e College Name
    let college_name = document.getElementById('college_name').value;

    // Set the Payload for the Post restapi call
    let options = { 'college_name': college_name };
    {
        // Use Async ajax call to post a request to the Client App
        $.when($.post('/composer/client/requestAffiliation', options)).done(function (_res) {

            // End progress display
            document.getElementById('college_progress').innerHTML = '';

            // Update the UI with college list
            displayCollegeList();

            // Close the Modal Dialog
            closeModal();
        });
    }
}
```

Will will setup router to call client app api in next Task

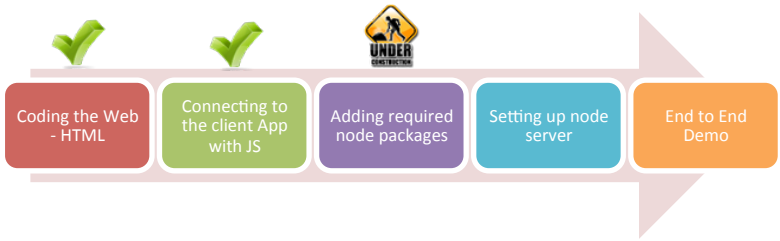


Step 3: Retrieving list of colleges from the client App and processing it to be displayed in HTML

```
/**
 * Display Colleges
 */
function displayCollegeList() {

    // Use Async ajax call to get a list of colleges from the Client App
    $.when($.get('composer/client/getCollegeList')).done(function (_res) {

        // Successful response received, lets check it on browser console
        console.log(_res.college_list);

        let _str = '';
        let _nstr = '';

        // We will now build the Table to be displayed in UI
        _str += '<table><tr><th>Name</th><th>isApproved</th></tr>';
        _res.college_list.forEach(function (_row) {
            // Check is college is approved or NOT aaproved by university
            let td = (_row.is_approved == 0) ? '<td class="red"> NOT APPROVED</td>'
                    : '<td class="green"> APPROVED</td>';

            _str += '<tr><td>' + '<a href="#" onClick=enrollProgram("' + _row.id + '")>'
                    + _row.name + '</a></td>' + td + '</tr>';
            if (_row.is_approved == 0) {
                _nstr += '<input type="checkbox" name="collegeIds" value="'
                    + _row.id + '">' + _row.name + '</input></br>';
            }
        })
        _str += '</table>';

        // Display College List
        document.getElementById('college_list').innerHTML = _str;
        document.getElementById('approve_list').innerHTML = _nstr;
    });
```

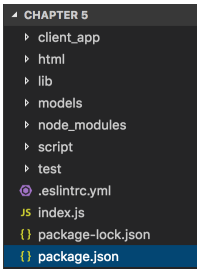
Step 4: Similarly we can code other methods to call the client App methods and process them to be displayed in HTML. Please visit 'events.js' for other methods.

Task 2 is complete!

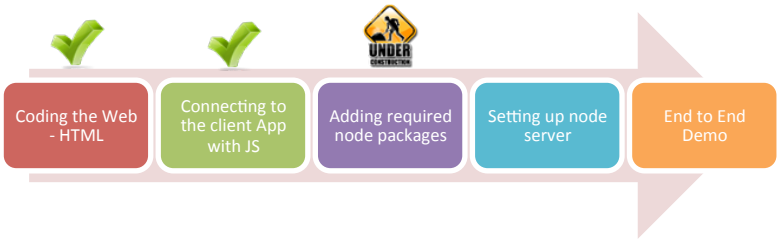


TASK#3: Adding dependency packages

Step 1: Open 'chapter05' code provided in Visual Studio Code. Browse to the open file 'package.json'



Step 2: Let's add dependency packages that we will need for our nodejs server and routing the restapi calls



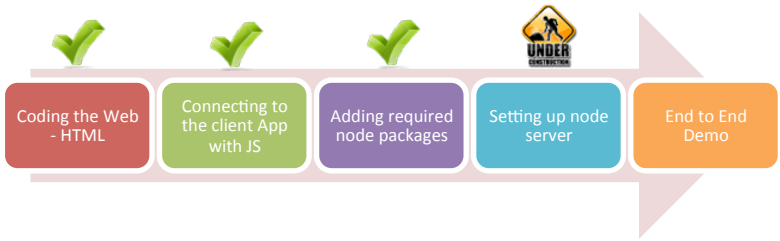
Step 3: Open terminal window of Visual Studio code and type run the following command

```
npm install
```

There might be some warnings but that is normal. Packages are now added to the system

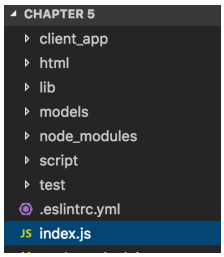
```
xcode-select: error: tool 'xcodebuild' requires Xcode, but active developer directory '/Library/Developer/CommandLineTools' i
  a command line tools instance

gyp WARN download NVM_NODEJS_ORG_MIRROR is deprecated and will be removed in node-gyp v4, please use NODEJS_ORG_MIRROR
  CXX(target) Release/obj.target/node_sleep/sleep.o
  SOLINK_MODULE(target) Release/node_sleep.node
npm WARN university_example@0.0.1 No repository field.
npm WARN university_example@0.0.1 license should be a valid SPDX license expression

added 213 packages in 22.266s
```

Task 3 is complete!



TASK#4: Setting up the node server

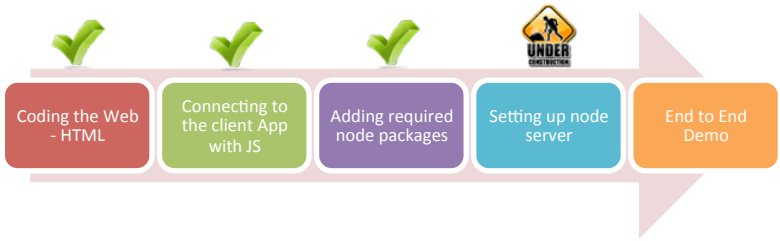
Step 1: Open 'chapter05' code provided in Visual Studio Code. Browse to the open file 'index.js' in the main folder



Step 2: Open 'chapter05' code provided in Visual Studio Code. Browse to the open file 'index.js' in the main folder

```
/*
 * University Usecase - Node Server */
var express = require('express');
var http = require('http');
var https = require('https');
var path = require('path');
var bodyParser = require('body-parser');
var cfenv = require('cfenv');

var appEnv = cfenv.getAppEnv();
var app = express();
```

- Define required packages
- Setup the environment variable for selecting the right port



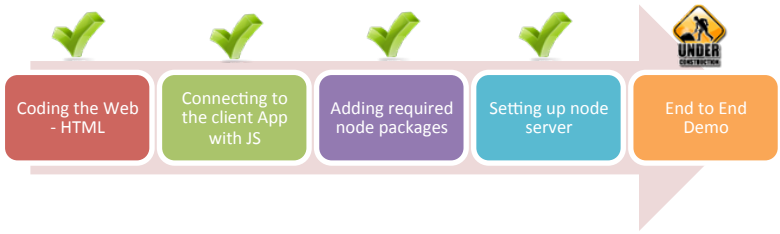
Step 3: Define where are the views, json parsing and application name for port registration

```
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.set('appName', 'org.gryphon.casestudy');
app.set('port', appEnv.port);

app.set('views', path.join(__dirname + '/html'));
app.engine('html', require('ejs').renderFile);
app.set('view engine', 'ejs');
app.use(express.static(__dirname + '/html'));
app.use(bodyParser.json());
```

Step 4: Define restapi router and start listening to the port for rest api calls

```
// Define your own router file in controller folder, export the router, add it into the index.js.
// app.use('/', require("./controller/yourOwnRouter"));
app.use('/', require("./client_app/controller/router"));

if (cfenv.getAppEnv().isLocal == true)
  {
    var server = app.listen(app.get('port'), function() {
    console.log('Listening locally on port %d', server.address().port);}); }
  else {
    var server = app.listen(app.get('port'), function() {
      console.log('Listening remotely on port %d', server.address().port);});
  }
```

Task 4 is complete!



TASK#5: Running the use-case End to End

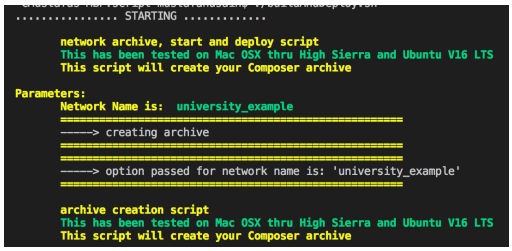
Step 1: Open 'chapter05' code provided in Visual Studio Code. Open the terminal window and cd into the scripts folder

```
cd script
```

Step 2: Let's completely build and deploy the blockchain application. Use the following script

```
./buildAndDeploy.sh
```

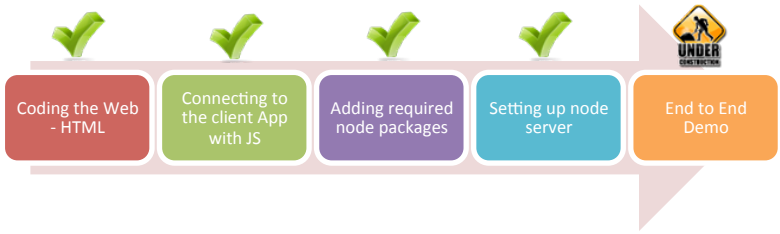
This will start the Hyperledger fabric, install the Business Network Application and created the Peer / Admin cards

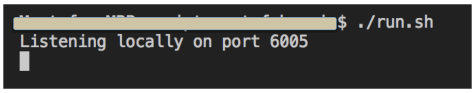


Step 3: Now let's run the rest server using the following script

```
./run.sh
```

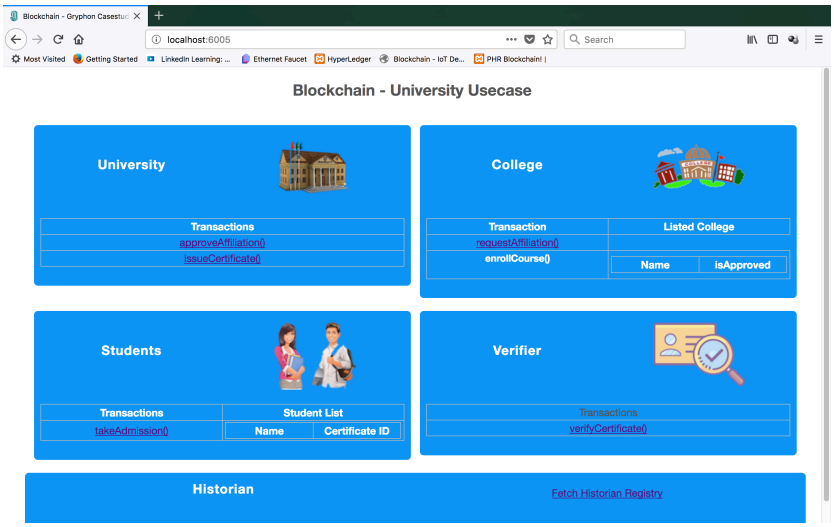
The node server will startup and show the port + address where you can view the frontend html pages in the browser.





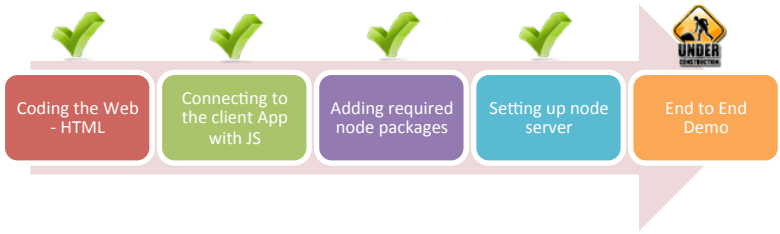
Step 4: Launch Internet browser and type the URL as below with port number as displayed in previous step:

URL: localhost:6005 *(Use Port displayed on your screen)*



Step 5: Our use-case UI is ready for action, the sequence of order of transactions will be:

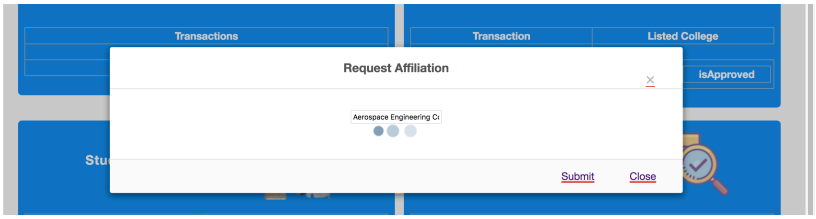
1. requestAffiliation() [By College]
2. approveAffliation() [By University]
3. enrollProgram() [By College]
4. takeAdmission() [By Student]



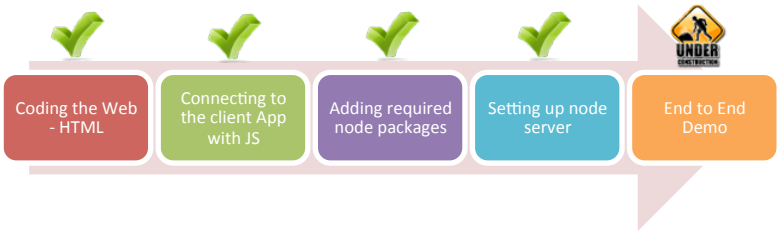
5. issueCertificate() [By University]
6. verifyCertificate() [By Verifier]
7. Finally we will check the transaction history using the Historian

Let's execute the first transaction by clicking requestAffiliation() in College section.

Step 6: In the popup dialog enter any college name and click submit:

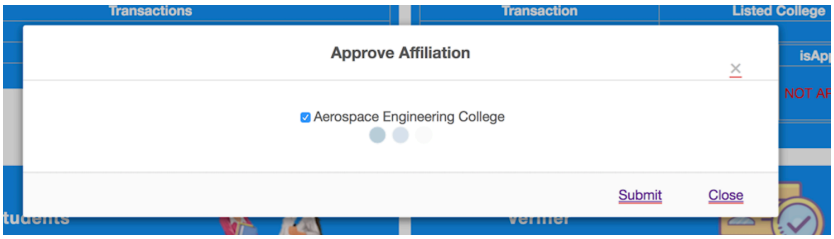


Progress image will show up and when transaction is complete, college will be listed in the College Section:

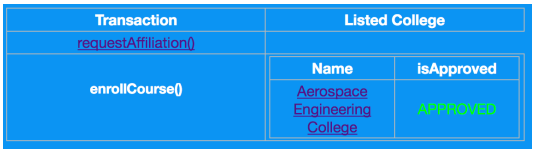


Step 7: Now from University Panel click on the approveAffiliation() link

Step 8: On the new popup dialog, select the college checkbox and click submit

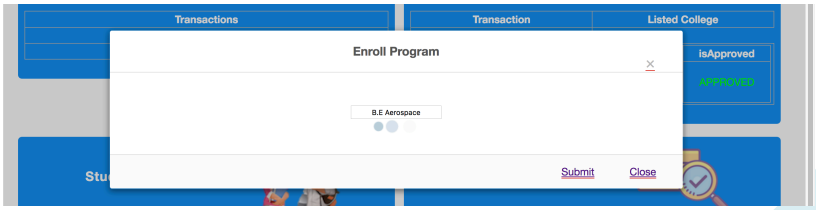


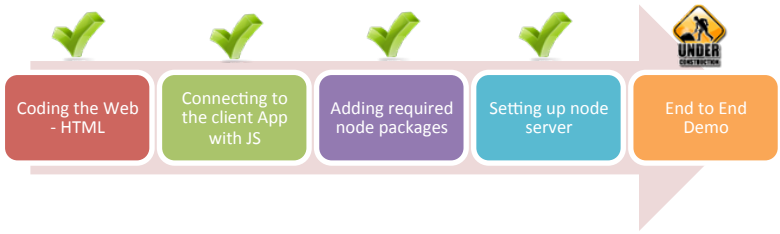
The college status should show up in green and as 'Approved'



Step 9: Now college can start enrolling new programs. Click the college name to enroll program.

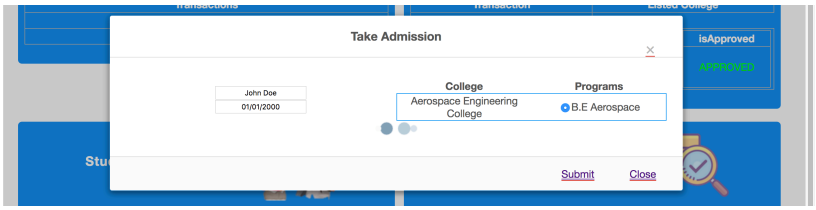
Step 10: A popup dialog asking for Program name should open up. Enter any name of the program and click submit



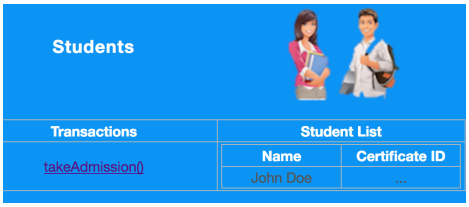


Step 11: Students can now take admission to the college for the respective programs offered by the college. Under Student section, click on takeAdmission() transaction link

Step 12: On the new opened dialog, enter the Student Name, date of birth and select appropriate college and program. Then click submit.

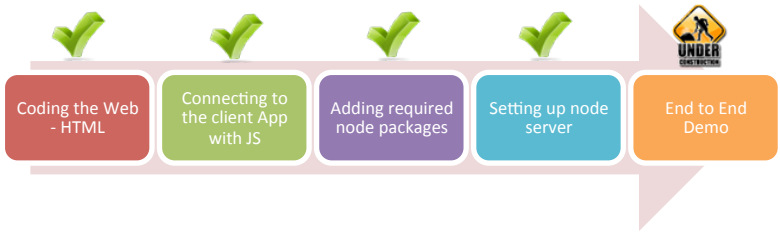


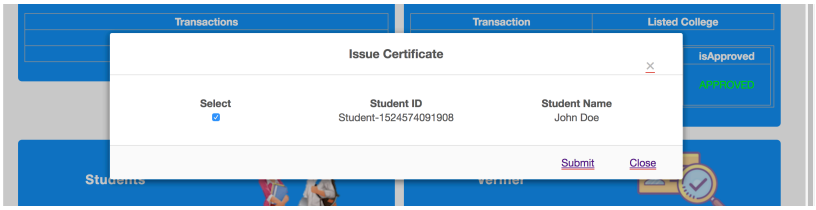
Student should be enrolled and student's name starts showing up on the Student section



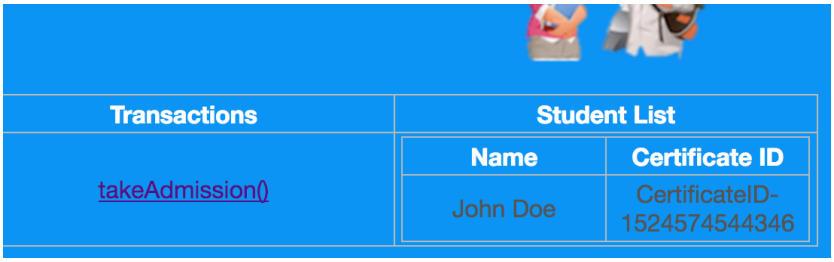
Step 13: Student has now completed the program and University will issue the certificate. Click on issueCertificate() under University section to generate the certificate for the student

Step 14: Select the student who has to be awarded the certificate and click submit



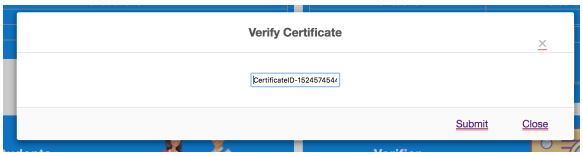


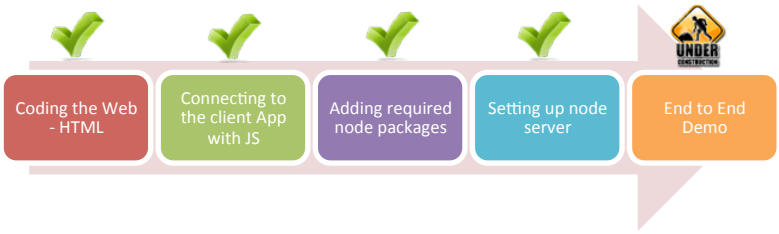
Step 14: Select the student who has to be awarded the certificate and click submit



A certificate is generated and the Certificate unique ID is displayed against the student.

Step 15: Select the CertifcateID and copy it in clipboard. Now for verification click on verifyCertificate() under the verifier section.

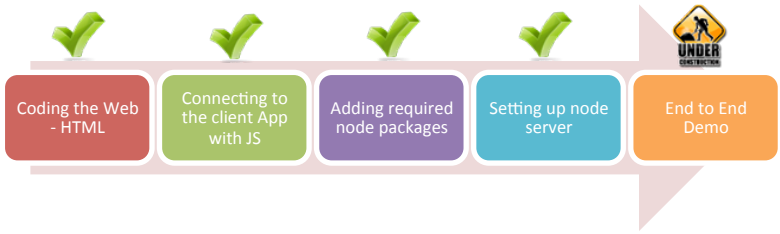




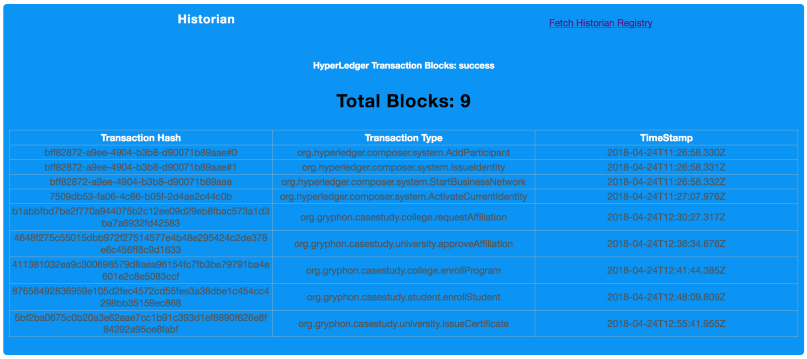
Step 16: Enter the generated certificate ID on the new window and click submit.

Certificate of completion of the program will be displayed. The Asset is now in blockchain, secured and uniquely identifies the student's authentic academics.

Step 17: Blockchain represent an un-mutable set of transactions. To check what transactions have occurred on the blockchain and validate the authenticity of the transaction that created the certificate we can view the Historian – i.e the transaction log.



Step 18: To view the Historian, click on the 'Fetch Historian Registry' link in the bottom section.



Complete list of transaction with their unique ID and time are displayed.

Task 5 is complete!

SUMMARY

End to End use-case for Hyperledger is now ready for demo! Students are encouraged to play around with the UI and the functionality defined here and better it.

Please be aware that this is just for learning and many corner test-cases have not been considered.

Further to this there are few advanced topics that we shall cover in our next chapters like:

- Securing the Rest Server
- Adding Organizations & Creating the channels

In this chapter we have learnt the following:

- Developing Interactive frontend for Hyperledger
- Bridging the UI with client App using javascript calls
- Setting up node server and routing rest-api calls to call the client app
- Invoking Client App API

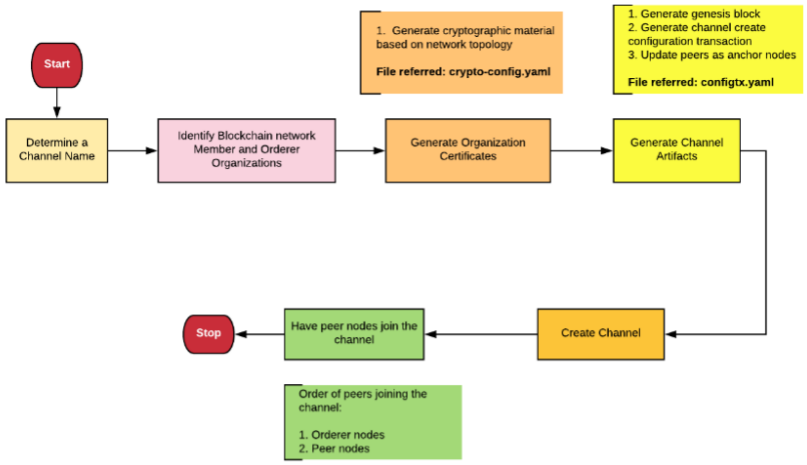
CHAPTER 6: Exploring Blockchain

Theory

In this chapter we will use Hyperledger Fabric provided sample application and a "Hyperledger Explorer" application to dig deeper and view the working & internals of Hyperledger in action;

- Multiple Organization & Peers
- Transactions & Actual Blocks of the blockchain
- Channel that is created for communication

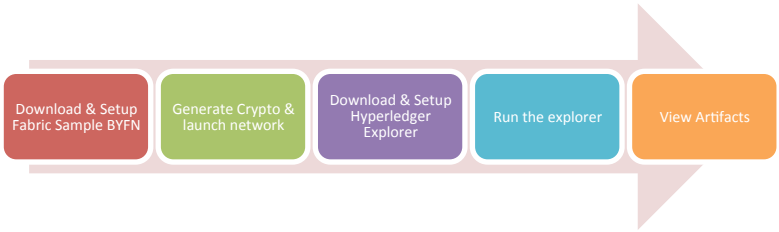
Hyperledger Fabric Blockchain Creation Workflow



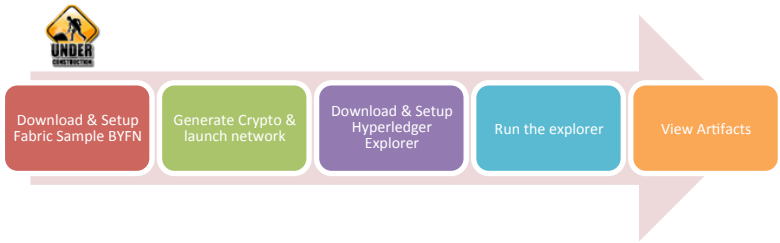
Using the Hyperledger Fabric example, we will

- Create a channel
- Identify the members, orderer
- Generate cryptographic material
- Join the network
- Explore the network

Lab Exercise 6: Exploring Hyperledger



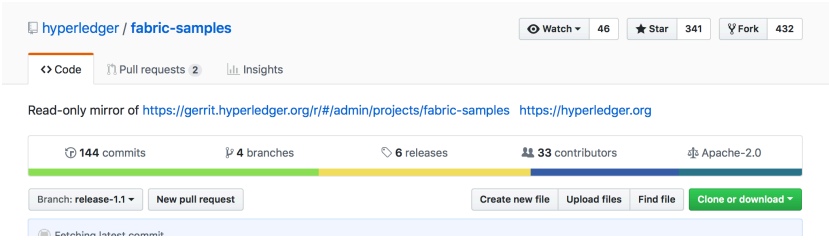
1. Download & Setup Fabric Sample BYFN
    a. Download fabric Samples
    b. Download Docker Images & crypto gen path setup
2. Generate Crypto & launch network
    a. Creating the Orgs, Solo Orderer, Peers and the channel (mychannel)
    b. Adding Peers to the Channel
    c. Identifying the Anchor peers for each org
    d. Installation and Instantiation of Chaincode on the peers
    e. Querying chaincode
    f. Invoking transactions
3. Download & Setup Hyperledger Explorer
    a. Download Explorer
    b. Install Posgre SQL
    c. Configure Postgre SQL
    d. Run Tests & Build React App
4. Run the explorer
5. View Artifacts



TASK#1: Download & Setup Fabric Sample - BYFN

Step 1: Launch internet browser and type the URL as below:

https://github.com/hyperledger/fabric-samples



Step 2: Using the 'clone or download' button in green, download the fabric-samples code.

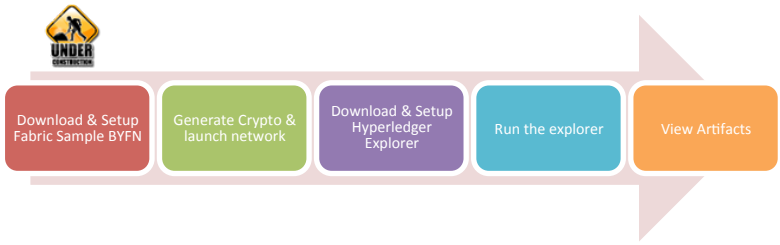
Step 3: Unzip the file just downloaded.

Step 4: Open the terminal window and CD into the unzip folder directory

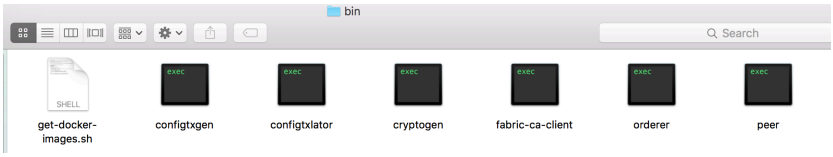
```
cd Download/fabric-samples-release-1.1/ first-network
```

Step 5: Execute the following command from within the directory into which you will extract the platform-specific binaries:

```
curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0
```



The command above downloads and executes a bash script that will download and extract all of the platform-specific binaries you will need to set up your network and place then in the **bin subdirectory**

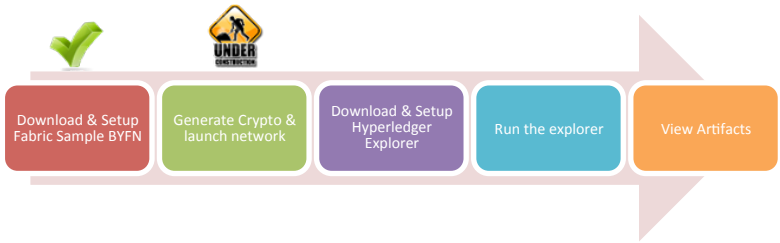


Step 6: Add that to your PATH environment variable so that these can be picked up without fully qualifying the path to each binary

```
export PATH=<path to download location>/bin:$PATH
```

At this stage we will have all the docker images required for our sample network downloaded, crypto binaries will be available in 'bin' folder and appropriate path is setup for our next step.

Task 1 is complete!



TASK#2: Generate Crypto Material & launch network

Step 1: Using the same above terminal where we have setup path for our binaries we will use it to generate crypto materials

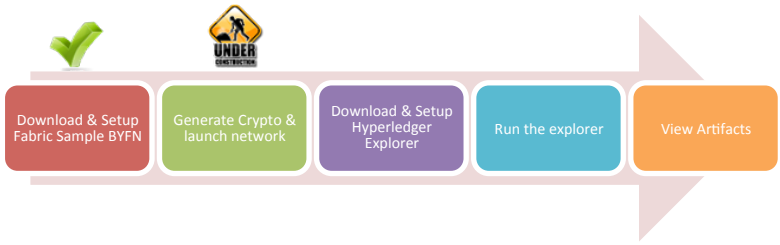
Now, move to the directory "first-network":

```
cd fabric-samples/first-network
```

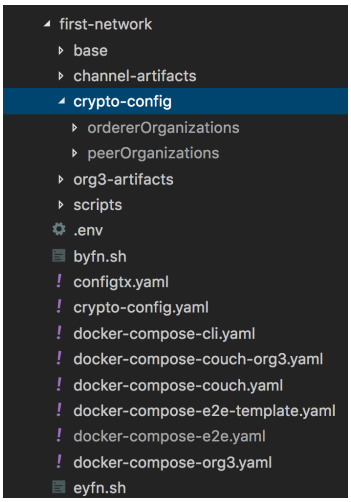
Step 2: Generate crypto materials using the following command:

```
./byfn -m generate
```

```
root@ip-172-31-84-59:~/fabric-samples/first-network# ./byfn.sh -m generate
Generating certs and genesis block for with channel 'mychannel' and CLI timeout of '10' seconds and CLI delay of '3' seconds
Continue (y/n)? y
proceeding ...
/home/ubuntu/bin/cryptogen

##########################################################
##### Generate certificates using cryptogen tool #########
##########################################################
org1.example.com
org2.example.com

/home/ubuntu/bin/configtxgen
##########################################################
#########  Generating Orderer Genesis block ##############
##########################################################
2018-01-08 00:26:05.253 UTC [common/tools/configtxgen] main -> INFO 001 Loading configuration
2018-01-08 00:26:05.261 UTC [common/tools/configtxgen] doOutputBlock -> INFO 002 Generating genesis block
2018-01-08 00:26:05.262 UTC [common/tools/configtxgen] doOutputBlock -> INFO 003 Writing genesis block

#################################################################
### Generating channel configuration transaction 'channel.tx' ###
#################################################################
2018-01-08 00:26:05.269 UTC [common/tools/configtxgen] main -> INFO 001 Loading configuration
2018-01-08 00:26:05.277 UTC [common/tools/configtxgen] doOutputChannelCreateTx -> INFO 002 Generating new channel configtx
2018-01-08 00:26:05.277 UTC [common/tools/configtxgen] doOutputChannelCreateTx -> INFO 003 Writing new channel tx

#################################################################
#######    Generating anchor peer update for Org1MSP   ##########
#################################################################
2018-01-08 00:26:05.285 UTC [common/tools/configtxgen] main -> INFO 001 Loading configuration
2018-01-08 00:26:05.292 UTC [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 002 Generating anchor peer update
2018-01-08 00:26:05.293 UTC [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Writing anchor peer update

#################################################################
#######    Generating anchor peer update for Org2MSP   ##########
#################################################################
2018-01-08 00:26:05.300 UTC [common/tools/configtxgen] main -> INFO 001 Loading configuration
2018-01-08 00:26:05.308 UTC [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 002 Generating anchor peer update
2018-01-08 00:26:05.308 UTC [common/tools/configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Writing anchor peer update
```

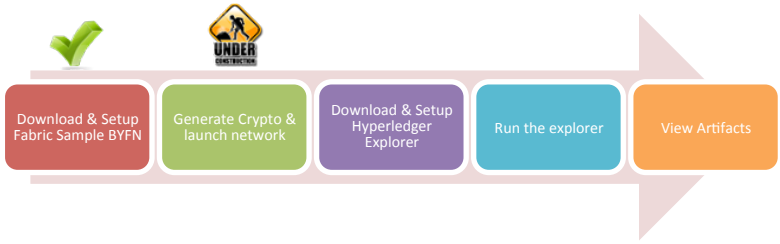


This command will add the 'crypto-config' directory to the "first-network" folder



Step 3: Now, bring up the "first-network" and run the following commands:

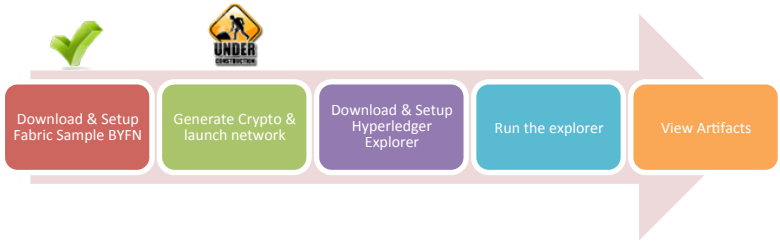
```
./byfn -m up
```



Understanding what went under the hood:

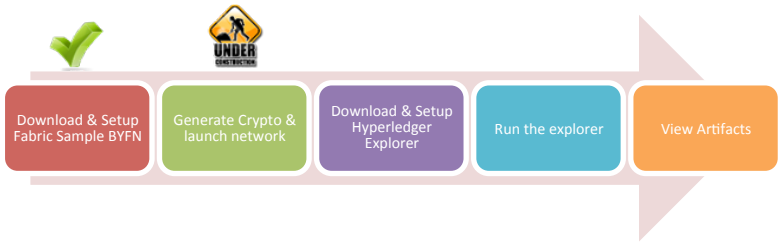
Creating the Orgs, Solo Orderer, Peers and the channel (mychannel)

```
[root@ip-172-31-84-59:~/fabric-samples/first-network# ./byfn.sh -m up
Starting with channel 'mychannel' and CLI timeout of '10' seconds and CLI delay of '3' seconds
Continue (y/n)? y
proceeding ...
Creating network "net_byfn" with the default driver
Creating peer1.org1.example.com
Creating orderer.example.com
Creating peer0.org1.example.com
Creating peer1.org2.example.com
Creating peer0.org2.example.com
Creating cli

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

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

Channel name : mychannel
Creating channel...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org1MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
2018-01-08 00:30:32.556 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:32.556 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:32.561 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2018-01-08 00:30:32.561 UTC [msp] GetLocalMSP -> DEBU 004 Returning existing local MSP
2018-01-08 00:30:32.561 UTC [msp] GetDefaultSigningIdentity -> DEBU 005 Obtaining default signing identity
2018-01-08 00:30:32.561 UTC [msp] GetLocalMSP -> DEBU 006 Returning existing local MSP
2018-01-08 00:30:32.561 UTC [msp] GetDefaultSigningIdentity -> DEBU 007 Obtaining default signing identity
2018-01-08 00:30:32.561 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0AA2060A074F7267314D53501296062D...53616D706C65436F6E736F727469756D
2018-01-08 00:30:32.561 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: 2E2FAED23C7E131CA1065433730B458DFA76224CE2636009BFAFABA5BD4C2DF5
2018-01-08 00:30:32.561 UTC [msp] GetLocalMSP -> DEBU 00a Returning existing local MSP
2018-01-08 00:30:32.561 UTC [msp] GetDefaultSigningIdentity -> DEBU 00b Obtaining default signing identity
2018-01-08 00:30:32.561 UTC [msp] GetLocalMSP -> DEBU 00c Returning existing local MSP
2018-01-08 00:30:32.561 UTC [msp] GetDefaultSigningIdentity -> DEBU 00d Obtaining default signing identity
2018-01-08 00:30:32.561 UTC [msp/identity] Sign -> DEBU 00e Sign: plaintext: 0AD9060A1508021A0608ABF7CAD20522...37D4597246D9124DA2383A7CB77468A8
2018-01-08 00:30:32.561 UTC [msp/identity] Sign -> DEBU 00f Sign: digest: EF75E253339E69E76624353 0E09F13E7DAAAA4B45B366BDFD023075D511A23D6
2018-01-08 00:30:32.620 UTC [msp] GetLocalMSP -> DEBU 010 Returning existing local MSP
2018-01-08 00:30:32.620 UTC [msp] GetDefaultSigningIdentity -> DEBU 011 Obtaining default signing identity
2018-01-08 00:30:32.620 UTC [msp] GetLocalMSP -> DEBU 012 Returning existing local MSP
2018-01-08 00:30:32.620 UTC [msp] GetDefaultSigningIdentity -> DEBU 013 Obtaining default signing identity
2018-01-08 00:30:32.620 UTC [msp/identity] Sign -> DEBU 014 Sign: plaintext: 0AD9060A1508021A0608ABF7CAD20522...3B3E9B4C490B12080A021A0012021A00
2018-01-08 00:30:32.620 UTC [msp/identity] Sign -> DEBU 015 Sign: digest: A3C9190B1DDF16768DD708457AFCD46E5BB873DAC04D37BB978D585F6F6EFD48
2018-01-08 00:30:32.656 UTC [channelCmd] readBlock -> DEBU 016 Received block: 0
2018-01-08 00:30:32.656 UTC [main] main -> INFO 017 Exiting.....
===================== Channel "mychannel" is created successfully =====================

Having all peers join the channel...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org1MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
2018-01-08 00:30:32.698 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:32.698 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:32.701 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2018-01-08 00:30:32.701 UTC [msp/identity] Sign -> DEBU 004 Sign: plaintext: 0AA0070A5C080114A0C08ABF7CAD20510...D445B178EC611A088A000A000A000A00
2018-01-08 00:30:32.701 UTC [msp/identity] Sign -> DEBU 005 Sign: digest: 4C4A4123625B5AACC312E24D11FDBE568C5EF1482832B494EAE9D47BF4375FF8
2018-01-08 00:30:32.757 UTC [channelCmd] executeJoin -> INFO 006 Peer joined the channel!
2018-01-08 00:30:32.757 UTC [main] main -> INFO 007 Exiting.....
```



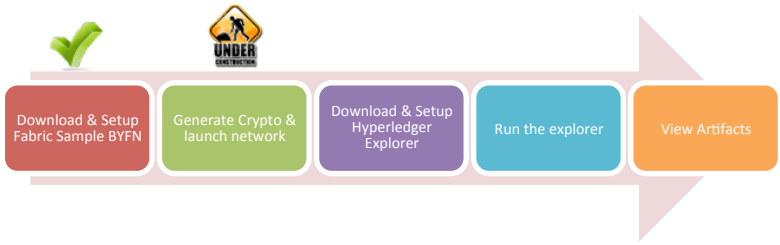
Adding Peers to the Channel

```
==================== PEER0 joined on the channel "mychannel" ====================

CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org1MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer1.org1.example.com:7051
2018-01-08 00:30:35.798 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:35.798 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:35.802 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2018-01-08 00:30:35.802 UTC [msp/identity] Sign -> DEBU 004 Sign: plaintext: 0AA0070A5C0B011A0C08ABF7CAD20510...D445B170EC611A080A000A000A000A00
2018-01-08 00:30:35.802 UTC [msp/identity] Sign -> DEBU 005 Sign: digest: F2084262797F79B670D4FCA3DC49813D0BFFCA3E7C888E71CE135EDBE1528A28
2018-01-08 00:30:35.854 UTC [channelCmd] executeJoin -> INFO 006 Peer joined the channel!
2018-01-08 00:30:35.856 UTC [main] main -> INFO 007 Exiting.....
==================== PEER1 joined on the channel "mychannel" ====================

CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org2MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
2018-01-08 00:30:38.897 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:38.897 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:38.899 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2018-01-08 00:30:38.900 UTC [msp/identity] Sign -> DEBU 004 Sign: plaintext: 0A9C070A5C0B011A0C08AEF7CAD20510...D445B170EC611A080A000A000A000A00
2018-01-08 00:30:38.900 UTC [msp/identity] Sign -> DEBU 005 Sign: digest: A78C58B8AE1118581D4E179CFB3478E037DFAEC6942CCBBC2AD7B895B0E92A8
2018-01-08 00:30:38.956 UTC [channelCmd] executeJoin -> INFO 006 Peer joined the channel!
2018-01-08 00:30:38.956 UTC [main] main -> INFO 007 Exiting.....
==================== PEER2 joined on the channel "mychannel" ====================

CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org2MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer1.org2.example.com:7051
2018-01-08 00:30:41.996 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:41.996 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:41.999 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2018-01-08 00:30:41.999 UTC [msp/identity] Sign -> DEBU 004 Sign: plaintext: 0A9C070A5C0B011A0C08B1F7CAD20510...D445B170EC611A080A000A000A000A00
2018-01-08 00:30:41.999 UTC [msp/identity] Sign -> DEBU 005 Sign: digest: DC3D175B6BE654BA26EBCA1227757A4F7B3650639 3A4219A9778E59AA0779DD3
2018-01-08 00:30:42.064 UTC [channelCmd] executeJoin -> INFO 006 Peer joined the channel!
2018-01-08 00:30:42.064 UTC [main] main -> INFO 007 Exiting.....
==================== PEER3 joined on the channel "mychannel" ====================

Updating anchor peers for org1...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org1MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
2018-01-08 00:30:45.105 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:45.105 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:45.108 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2018-01-08 00:30:45.108 UTC [msp] GetLocalMSP -> DEBU 004 Returning existing local MSP
2018-01-08 00:30:45.109 UTC [msp] GetDefaultSigningIdentity -> DEBU 005 Obtaining default signing identity
2018-01-08 00:30:45.109 UTC [msp] GetLocalMSP -> DEBU 006 Returning existing local MSP
2018-01-08 00:30:45.109 UTC [msp] GetDefaultSigningIdentity -> DEBU 007 Obtaining default signing identity
2018-01-08 00:30:45.109 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0AA2060A074F7267314D53501296062D...2A0641646D696E732A0641646D696E73
2018-01-08 00:30:45.109 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: 0A4B5CE461E4115076 7A420479C24B3A6CBC339ABADA50034B0CC3D5B3D4BA9E
2018-01-08 00:30:45.109 UTC [msp] GetLocalMSP -> DEBU 00a Returning existing local MSP
2018-01-08 00:30:45.109 UTC [msp] GetDefaultSigningIdentity -> DEBU 00b Obtaining default signing identity
2018-01-08 00:30:45.109 UTC [msp] GetLocalMSP -> DEBU 00c Returning existing local MSP
2018-01-08 00:30:45.109 UTC [msp] GetDefaultSigningIdentity -> DEBU 00d Obtaining default signing identity
2018-01-08 00:30:45.109 UTC [msp/identity] Sign -> DEBU 00e Sign: plaintext: 0AD9060A1508021A0608B5F7CAD20522...7F345955079DFEF3E3A361C6495DB28E
2018-01-08 00:30:45.109 UTC [msp/identity] Sign -> DEBU 00f Sign: digest: C3AEE1C7A230706F7EE24105ED7A63FAB1EF1D28E91F8B36AAD7F97D096D81F
2018-01-08 00:30:45.137 UTC [main] main -> INFO 010 Exiting.....
```



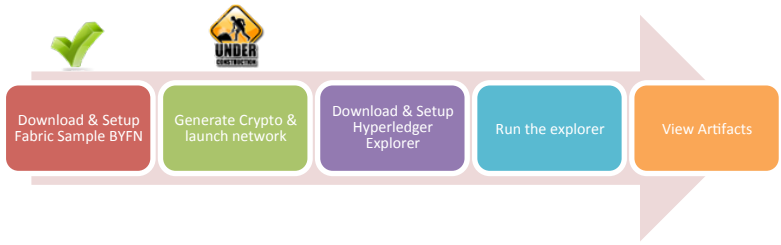
Identifying the Anchor peers for each org

```
===================== Anchor peers for org "Org1MSP" on "mychannel" is updated successfully =====================

Updating anchor peers for org2...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org2MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
2018-01-08 00:30:48.179 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:48.179 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:48.182 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2018-01-08 00:30:48.182 UTC [msp] GetLocalMSP -> DEBU 004 Returning existing local MSP
2018-01-08 00:30:48.182 UTC [msp] GetDefaultSigningIdentity -> DEBU 005 Obtaining default signing identity
2018-01-08 00:30:48.183 UTC [msp] GetLocalMSP -> DEBU 006 Returning existing local MSP
2018-01-08 00:30:48.183 UTC [msp] GetDefaultSigningIdentity -> DEBU 007 Obtaining default signing identity
2018-01-08 00:30:48.183 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0A9E060A074F7267324D535012920620...2A0641646D696E732A0641646D696E73
2018-01-08 00:30:48.183 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: 03937D2DF998F3D0B531F3388DDB807FC2CFCC1C5190516ECF96D08D2B51D89
2018-01-08 00:30:48.183 UTC [msp] GetLocalMSP -> DEBU 00a Returning existing local MSP
2018-01-08 00:30:48.183 UTC [msp] GetDefaultSigningIdentity -> DEBU 00b Obtaining default signing identity
2018-01-08 00:30:48.183 UTC [msp] GetLocalMSP -> DEBU 00c Returning existing local MSP
2018-01-08 00:30:48.183 UTC [msp] GetDefaultSigningIdentity -> DEBU 00d Obtaining default signing identity
2018-01-08 00:30:48.183 UTC [msp/identity] Sign -> DEBU 00e Sign: plaintext: 0AD5060A1508021A0608BBF7CAD20522...390D4EA5262282BCB3CEB5884805C45F
2018-01-08 00:30:48.183 UTC [msp/identity] Sign -> DEBU 00f Sign: digest: 9B4811F02F7BC22DDB5CD2895BFD4493505C568CB13D83F0490ED0BF1C7CA9B5
2018-01-08 00:30:48.209 UTC [main] main -> INFO 010 Exiting.....
===================== Anchor peers for org "Org2MSP" on "mychannel" is updated successfully =====================

Installing chaincode on org1/peer0...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org1MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
2018-01-08 00:30:51.253 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:51.253 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:51.253 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2018-01-08 00:30:51.253 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2018-01-08 00:30:51.253 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode disabled
2018-01-08 00:30:51.435 UTC [golang-platform] getCodeFromFS -> DEBU 006 getCodeFromFS github.com/chaincode/chaincode_example02/go/
2018-01-08 00:30:51.921 UTC [golang-platform] func1 -> DEBU 007 Discarding GOROOT package fmt
2018-01-08 00:30:51.921 UTC [golang-platform] func1 -> DEBU 008 Discarding provided package github.com/hyperledger/fabric/core/chaincode/shim
2018-01-08 00:30:51.921 UTC [golang-platform] func1 -> DEBU 009 Discarding provided package github.com/hyperledger/fabric/protos/peer
2018-01-08 00:30:51.921 UTC [golang-platform] func1 -> DEBU 00a Discarding GOROOT package strconv
2018-01-08 00:30:51.921 UTC [golang-platform] GetDeploymentPayload -> DEBU 00b done
2018-01-08 00:30:51.923 UTC [msp/identity] Sign -> DEBU 00c Sign: plaintext: 0AA0070A5C08031A0C08BBF7CAD20510...F619FF0E0000FFFFACD4020D001C0000
2018-01-08 00:30:51.923 UTC [msp/identity] Sign -> DEBU 00d Sign: digest: C04D1D4D331E7D69E2F7FDAB6440ED6ABADA9A947343CF91785C498DBF297C89
2018-01-08 00:30:51.927 UTC [chaincodeCmd] install -> DEBU 00e Installed remotely response:<status:200 payload:"OK" >
2018-01-08 00:30:51.927 UTC [main] main -> INFO 00f Exiting.....
```



Installation and Instantiation of Chaincode on the peers

```
==================== Chaincode is installed on remote peer PEER0 ====================

Install chaincode on org2/peer2...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org2MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
2018-01-08 00:30:51.970 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:51.970 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:51.970 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2018-01-08 00:30:51.970 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2018-01-08 00:30:51.970 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode disabled
2018-01-08 00:30:51.997 UTC [golang-platform] getCodeFromFS -> DEBU 006 getCodeFromFS github.com/chaincode/chaincode_example02/go/
2018-01-08 00:30:52.095 UTC [golang-platform] func1 -> DEBU 007 Discarding GOROOT package fmt
2018-01-08 00:30:52.095 UTC [golang-platform] func1 -> DEBU 008 Discarding provided package github.com/hyperledger/fabric/core/chaincode/shim
2018-01-08 00:30:52.095 UTC [golang-platform] func1 -> DEBU 009 Discarding provided package github.com/hyperledger/fabric/protos/peer
2018-01-08 00:30:52.095 UTC [golang-platform] func1 -> DEBU 00a Discarding GOROOT package strconv
2018-01-08 00:30:52.095 UTC [golang-platform] GetDeploymentPayload -> DEBU 00b done
2018-01-08 00:30:52.096 UTC [msp/identity] Sign -> DEBU 00c Sign: plaintext: 0A9B070A5B08031A0808BCF7CAD20510...F619FF0E0808FFFFACD4020D001C0808
2018-01-08 00:30:52.096 UTC [msp/identity] Sign -> DEBU 00d Sign: digest: 644203CA8B445B9961FF80DA212EE9A8B802F0F18F8A143306E7B73587248438
2018-01-08 00:30:52.101 UTC [chaincodeCmd] install -> DEBU 00e Installed remotely response:<status:200 payload:"OK" >
2018-01-08 00:30:52.101 UTC [main] main -> INFO 00f Exiting.....
==================== Chaincode is installed on remote peer PEER2 ====================

Instantiating chaincode on org2/peer2...
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
CORE_PEER_LOCALMSPID=Org2MSP
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
2018-01-08 00:30:52.144 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-01-08 00:30:52.144 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-01-08 00:30:52.147 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2018-01-08 00:30:52.147 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2018-01-08 00:30:52.147 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode disabled
2018-01-08 00:30:52.148 UTC [msp/identity] Sign -> DEBU 006 Sign: plaintext: 0AA6070A660B031A0B08BCF7CAD20510...324D53500A046573636300A0476736363
2018-01-08 00:30:52.148 UTC [msp/identity] Sign -> DEBU 007 Sign: digest: D77BF120C43442823CDA893CDD98E964FC19ED749922160388B5193C6AE1B72EA
2018-01-08 00:31:14.893 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0AA6070A660B031A0B08BCF7CAD20510...0DE7558F8D92736A6C63725B5C57E55C
2018-01-08 00:31:14.893 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: 2935DF77BE7EA93448DD4F60B70F9F7B16479FC4A245DAB1258CFB93BD55F698
2018-01-08 00:31:14.897 UTC [main] main -> INFO 00a Exiting.....
==================== Chaincode Instantiation on PEER2 on channel 'mychannel' is successful ====================
```



Querying chaincode

Invoking transactions

Task 2 is complete!

TASK#3: Setup Hyperledger Explorer

Step 1: Open new terminal window and enter the following commands:

```
sudo apt-get update
```

```
sudo apt-get install postgresql postgresql-contrib
```

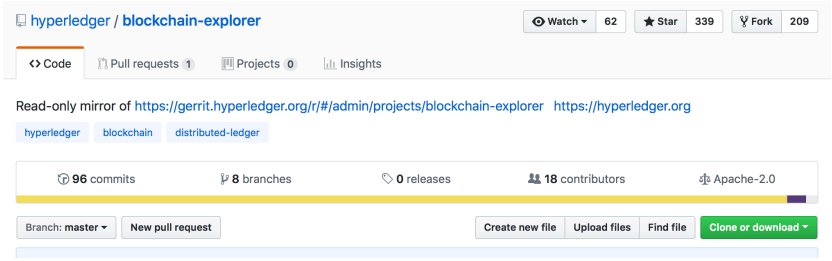
Above commands will install postgresql to the system

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
postgresql-contrib is already the newest version (9.5+173ubuntu0.1).
Suggested packages:
  postgresql-doc
The following NEW packages will be installed:
  postgresql
0 upgraded, 1 newly installed, 0 to remove and 154 not upgraded.
Need to get 0 B/5,450 B of archives.
After this operation, 59.4 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

** Please Note: On Mac – download the Postgre Installable Binary and install it.

Step 2: Launch internet browser and type the URL as below:

https://github.com/hyperledger/blockchain-explorer



Step 3: Using the 'clone or download' button in green, download the blockchain-explorer.

Step 4: Unzip the file just downloaded.

Step 5: Open the terminal window and CD into the unzip folder directory

```
cd blockchain-explorer
```

Step 6: Execute the following command from within the blockchain-explorer: to connect to the postgresql database server

```
sudo -u postgres psql
```

```
[sudo] password for mustafa:
psql (9.5.12)
Type "help" for help.

postgres=#
```

You will be logged in to postgres db with postgres command prompt

Step 7: Execute the following command to run create database script

```
\i app/db/explorerpg.sql
```

and

```
\i app/db/updatepg.sql
```

This will create the entire required database for the explorer application

Step 8: To validate the db is created successfully Run db status commands as below;

```
\l
```

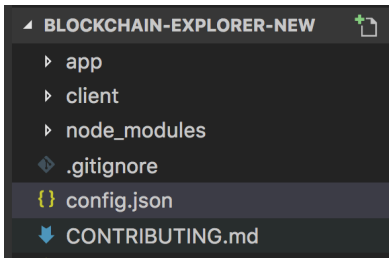
```
fabricexplorer=# \l
                                  List of databases
      Name      |  Owner   | Encoding | Collate | Ctype |   Access privileges
----------------+----------+----------+---------+-------+-----------------------
 fabricexplorer | hppoc    | UTF8     | en_IN   | en_IN |
 postgres       | postgres | UTF8     | en_IN   | en_IN |
 template0      | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
                |          |          |         |       | postgres=CTc/postgres
 template1      | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
                |          |          |         |       | postgres=CTc/postgres
(4 rows)

fabricexplorer=#
```

Also run the following command to view created tables

```
\d
```


```
fabricexplorer=# \d
                List of relations
 Schema |           Name            |   Type   | Owner
--------+---------------------------+----------+-------
 public | blocks                    | table    | hppoc
 public | blocks_id_seq             | sequence | hppoc
 public | chaincodes                | table    | hppoc
 public | chaincodes_id_seq         | sequence | hppoc
 public | channel                   | table    | hppoc
 public | channel_id_seq            | sequence | hppoc
 public | peer                      | table    | hppoc
 public | peer_id_seq               | sequence | hppoc
 public | peer_ref_channel          | table    | hppoc
 public | peer_ref_channel_id_seq   | sequence | hppoc
 public | transaction               | table    | hppoc
 public | transaction_id_seq        | sequence | hppoc
 public | write_lock                | table    | hppoc
 public | write_lock_write_lock_seq | sequence | hppoc
(14 rows)

fabricexplorer=#
```

Step 8: Open Visual Studio code application and open the file "***config.json***" under "blockchain-explorer" directory

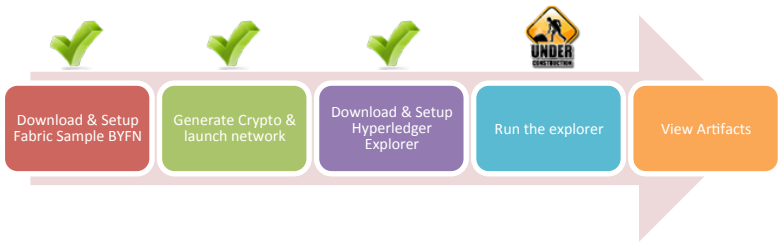


Step 9: Edit the path to "crypto-config" directory

Modify config.json to update network-config.

- Change "fabric-path" to your fabric network path, example: "/home/user1/Download/fabric-samples" for the following keys: "tls_cacerts", "key", "cert".
- Final path for key "tls_cacerts" will be: "/home/user1/ Download /fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt".
- Similiarly replace all "fabric-path" to reflect the actual path on your system

Task 3 is complete!



TASK#4: Build & Run the Explorer

Step 1: Open new terminal window and cd into the "blockchain-explorer" directory where the explorer code is

```
cd blockchain-explorer
```

Step 2: Run npm install in the main explorer folder

```
npm install
```

Step 3: Cd into the app/test folder

```
cd app/test
```

Step 4: Again run npm install now in the app/test folder

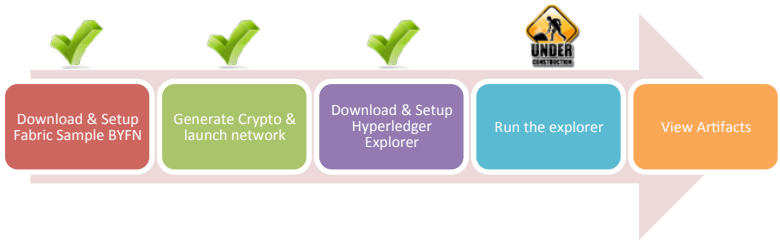
```
npm install
```

Step 5: Now cd into the client folder

```
cd ../../client/
```

Step 6: Again run npm install now in the client folder

```
npm install
```



Step 7: Finally run the test and coverage to verify things are OK

```
npm install npm test -- -u --coverage
```

Step 8: Everything is now set to run build and create production react explorer app

```
npm run build
```

Step 9: Once the build is done; cd back to main directory

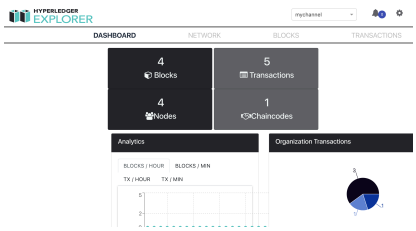
```
cd ..
```

Step 10: Time to run the explorer App

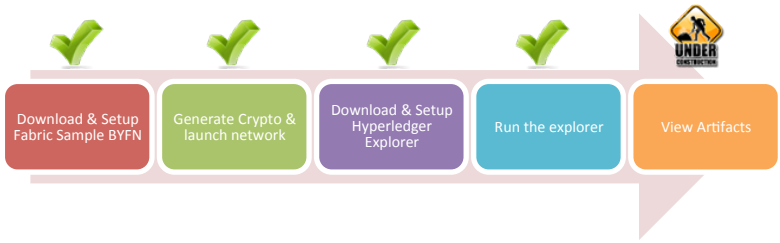
```
./start.sh
```

This will start the Hyperledger Blockchain explorer – nodejs server

Step 10: Launch Internet explorer and type the following URL: *http://localhost:8080/*

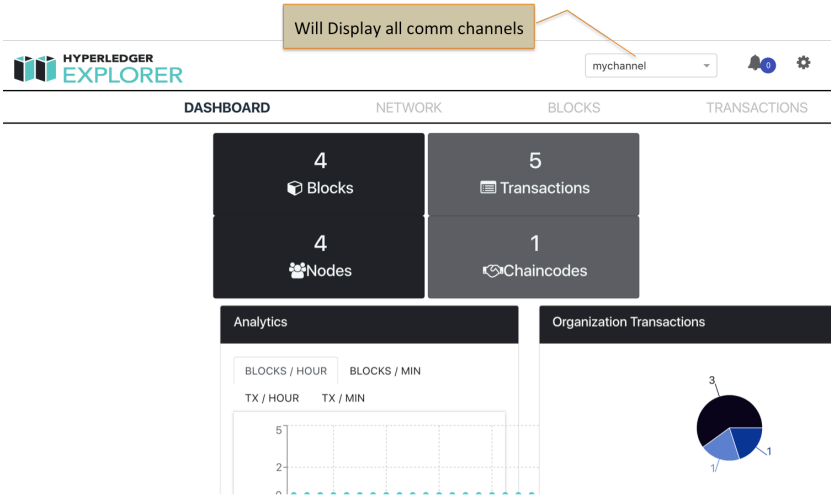


Task 4 is complete!



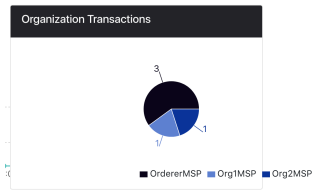
TASK#5: Exploring the explorer…

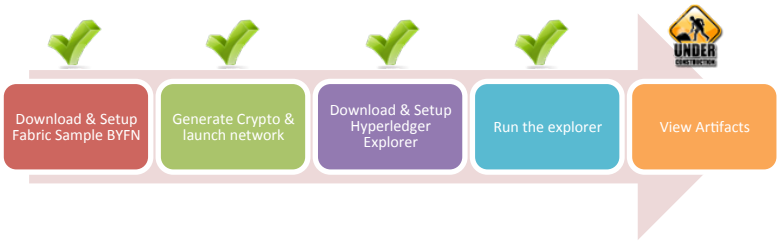
Step 1: Hyperledger Explorer Dashboard



You can also create new channels using ./byfn.sh script and check the transaction and new channel updated here.

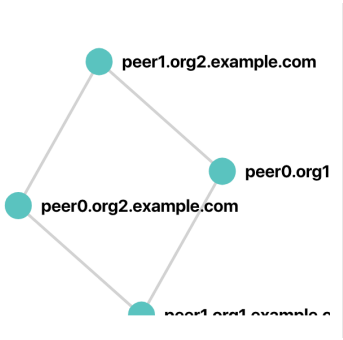
Step 2: Observe the Organization Graph section:





The graph shows that our network is having 2 organizations and three orderer service.

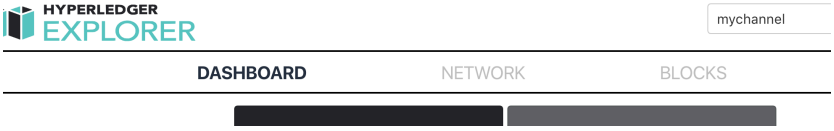
Step 3: Scroll Down to the PeerGraph section:



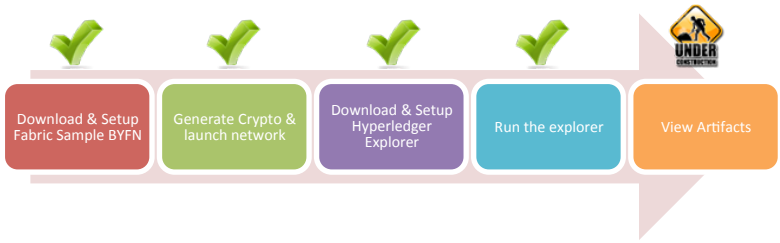
Its shows that we have two organizations and each org has 2 peers (peer0 & peer1)

This can be validated from the Task #2 logs

Step 3: Click on the Network Tab of the main menu



This Validates the step#2 about 4 peers



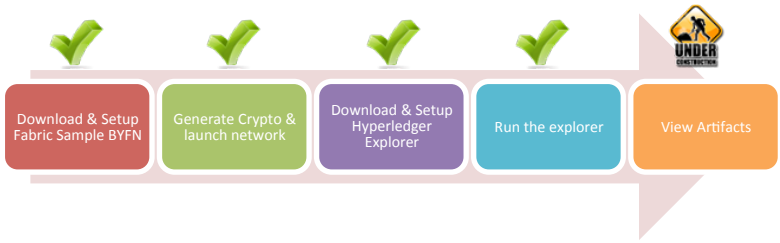
peer0.org1.example.com	grpcs://127.0.0.1:7051
peer1.org1.example.com	grpcs://127.0.0.1:8051
peer0.org2.example.com	grpcs://127.0.0.1:9051
peer1.org2.example.com	grpcs://127.0.0.1:10051

Step 4: Click on the BLOCKS Tab of the main menu

This Tab shows all the blocks generated till now for all the transactions that have been done so far – with current and prev hash

Block Number	Number of Tx	Data	Previous Hash	Transactions
4	1	b66bf1359a5bb6d831553bd17bdf7aa943aae...	40332448a71e1...	2aa0837a212ed635a39d87491c448a9e29a9...
3	1	6de6f4b7779514ec6ea86a4b931714061adda...	b2be48184258f...	f9c26204170d922cd68db404cb547733f29f2...
2	1	659a3931d0152ddd8e0f9383cc98e483a558...	33777c6e88209...	aeeb7b12985964e4be7b39bd3a178e8319ef4... 8534e34c84f96d6bc22d6f2d983aa36f9bd2...
1	1	23492f0433ad14b4691ec937b4763a546b9c...	fac43a6a26f515...	b1b87235894ef71447127726e1739ef54fb116...
0	1	38de89555fe28a001d9999544a237c1da416f...		

Step 4: Finally click on the TRANSACTIONS Tab of the main menu



Creator	Tx Id	Type	Chaincode	Timestamp
Org1MSP	2aa0837a212ed635a39d87...	ENDORSER_TRANSACTION	mycc	5-24-2018 12:12 PM IST
Org2MSP	f9c26204170d922cd68db4...	ENDORSER_TRANSACTION	mycc	5-24-2018 12:11 PM IST
OrdererMSP		CONFIG		5-24-2018 12:11 PM IST
OrdererMSP		CONFIG		5-24-2018 12:11 PM IST
OrdererMSP		CONFIG		5-24-2018 12:10 PM IST

It displays all the transactions that has been endorsed and by which orgs.

Step 5: Click on any Transaction ID (shown in blue as url)

Transaction Detail

Tx:2aa0837a212ed635a39d87491c448a9e29a92d0fa4d210018e3b6759faed27a6
Creator MSP: Org1MSP
Endorser: {"Org1MSP"}
Chaincode Name: mycc
Type: ENDORSER_TRANSACTION
Time: 5-24-2018 12:12 PM IST
Reads:
lscc
key:mycc ,version:(block:3,tx:0)

mycc
key:a ,version:(block:3,tx:0)
key:b ,version:(block:3,tx:0)

Writes:
lscc

mycc
key:a ,is_delete:false,value:90
key:b ,is_delete:false,value:210

Details of Transaction, endorser, chaincode, time & block are displayed.

Task 5 is complete!

SUMMARY

In this chapter we

- Created a Business network with 2 organization and 2 peers each (total 4 peers).
- Generated the crypto material for the organizations and peers
- Created Solo Orderer and the channel (mychannel)
- Added Peers to the Channel
- Identified the Anchor peers for each org
- Installed Chaincode on the peers
- Query chaincode & Invoked transactions

Finally we used a Blockchain explorer to view all the above hyperledger components.

Students are further encouraged to explore deeper and play with the Hyperledger explorer for better learning.

CHAPTER 7: Adding a new Peer

Theory

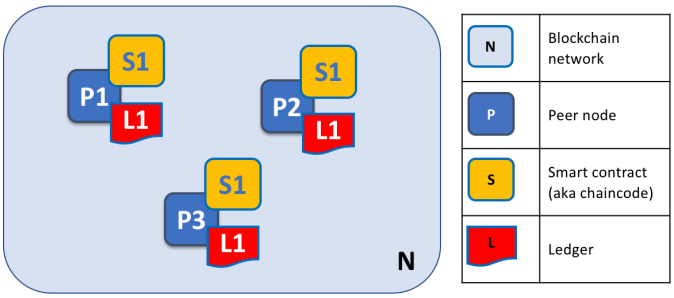
In this chapter we will add a new peer to our business network and deploy our University Usecase to it.

This topic focusses on peers, and their relationship to those other elements in a Hyperledger Fabric blockchain network.

Peers

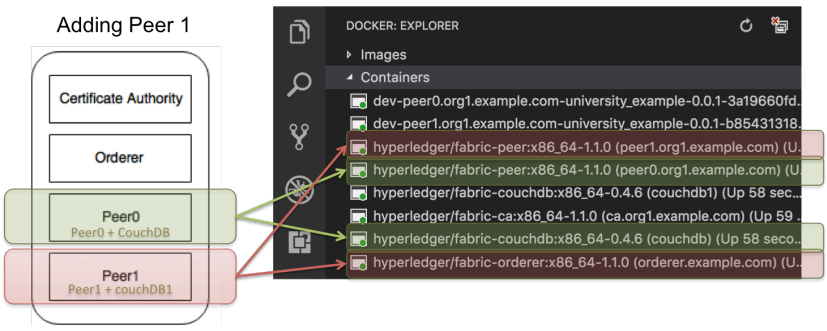
A blockchain network is primarily comprised of a set of peer nodes. Peers are a fundamental element of the network because they host ledgers and smart contracts.

Ledger immutably records all the transactions generated by smart contracts. Smart contracts and ledgers are used to encapsulate the shared processes and shared information in a network, respectively. These aspects of a peer make them a good starting point to understand Hyperledger Fabric network.



A blockchain network is formed from peer nodes, each of which can hold copies of ledgers and copies of smart contracts. In this example, the network N is formed by peers P1, P2 and P3. P1, P2 and P3 each maintain their own instance of the ledger L1. P1, P2 and P3 use chaincode S1 to access their copy of the ledger L1.

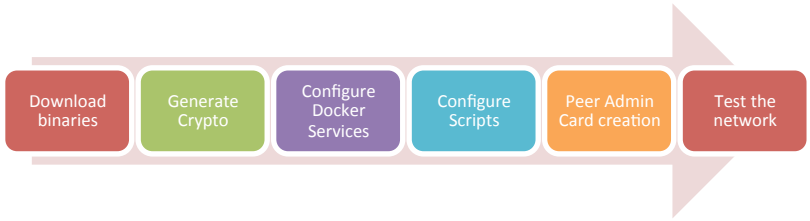
University Usecase – A new Peer



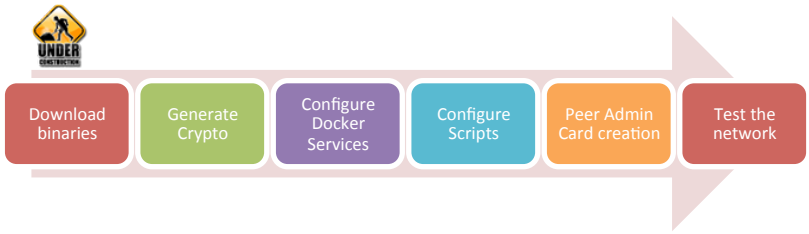
Using the University Usecase Hyperledger Fabric example, we will

- Create a channel - **composerchannel**
- Identify the members,
    - Orderer
    - CA
    - Peer0 (+ couchdb)
    - Peer1 (+ couchdb1)
- Generate cryptographic material
- Setup Docker for multiple peers
- Join peer to the channel
- Create Peer Admin card
- Deploy the business network
- Test the peers

Lab Exercise 7: Adding a peer to Hyperledger Fabric



1. Download binaries to generate crypto material
2. Generate Crypto material
    a. Update cypto-config.yaml
    b. Generate Crypto
3. Configure Docker Services
    a. Add Peers + CouchDb
    b. Update crypto keys
4. Configure Scripts
    a. Add script to join Peer1 to the channel
5. Peer Admin Card creation
    a. Update connection string
6. Test the network
    a. Visualize Docker setup
    b. Deploy university usecase
    c. Stop 1 Peer and validate



TASK#1: Download Binaries & setup path

Step 1: Open the terminal window and CD into the Fabric's root folder directory

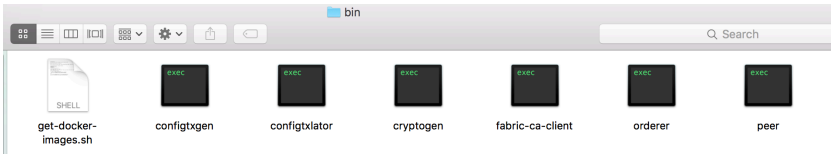
** **Please Note:** In this book for compatibility reasons we have renamed Fabric's root directory to **'fabric-tools'**. So If you have **NOT** installed Hyperledger using this Book & Scripts root folder name will be **'fabric-dev-servers'** instead

```
cd $HOME/fabric-tools
```

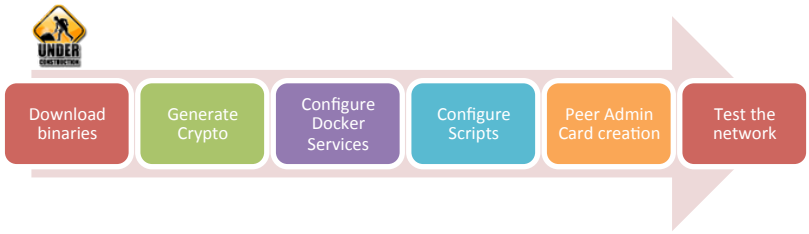
Step 2: Execute the following command from within the directory into which you will extract the platform-specific binaries:

```
curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0 -d -s
```

The command above downloads and executes a bash script that will download and extract all of the platform-specific binaries you will need to set up your network and place then in the **bin subdirectory**



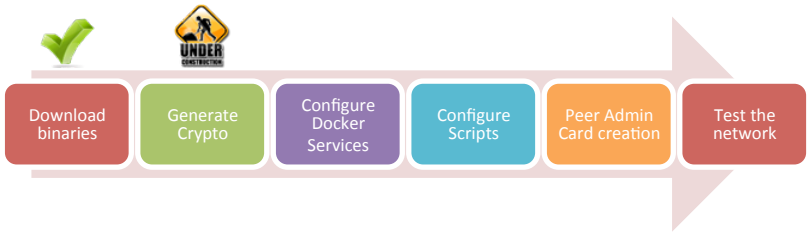
Step 3: Add that to your PATH environment variable so that these can be picked up without fully qualifying the path to each binary




```
export PATH=$HOME/fabric-tools/bin:$PATH
```

** Don't close the Terminal Window, we will be using the same terminal window for generating crypto-materials

Task 1 is complete!



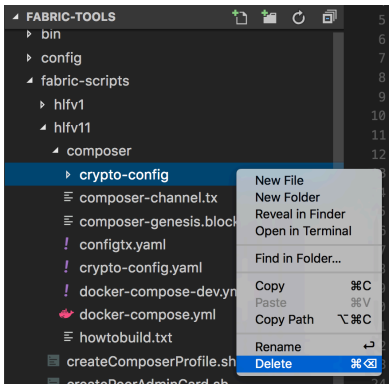
TASK#2: Generate Crypto Material

Step 1: Using the same above terminal where we have setup path for our binaries we will use it to generate crypto materials

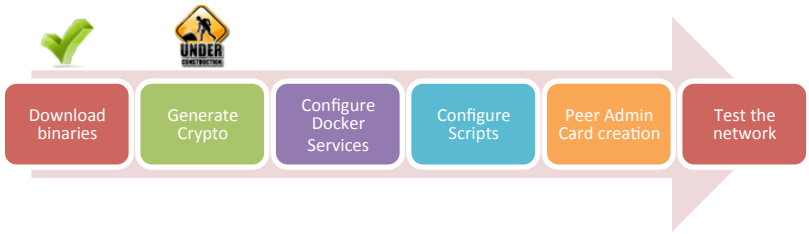
Now, move to the directory Hyperledger 1.1 – composer directory:

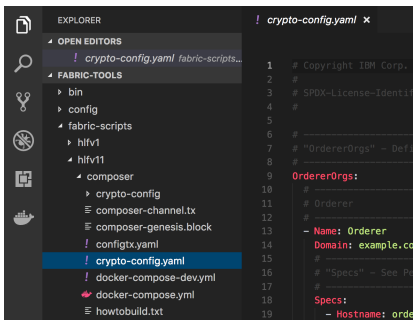
```
cd $HOME/fabric-tools/fabric-scripts/hlfv11/composer
```

Step 2: Delete the existing 'crypto-config' directory

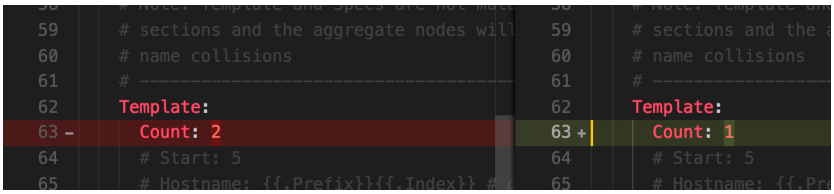


Step 3: Using Visual Studio Code Open the "fabric-tools" folder and open the "crypto-config.yaml" for editing





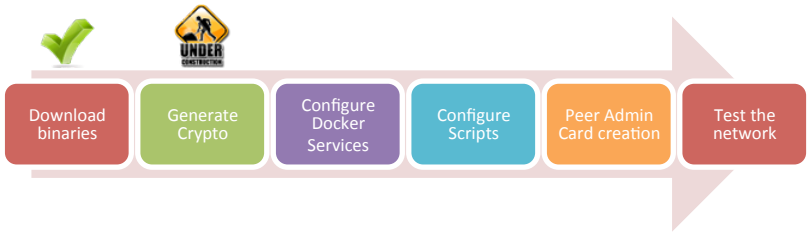
Step 2: We want 2 peers instead of existing 1 – hence we update the crypto material required in "crypto-config.yaml" to **'2' as shown below**



Step 3: Generate crypto materials using the following commands:

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

```
export FABRIC_CFG_PATH=$PWD
```



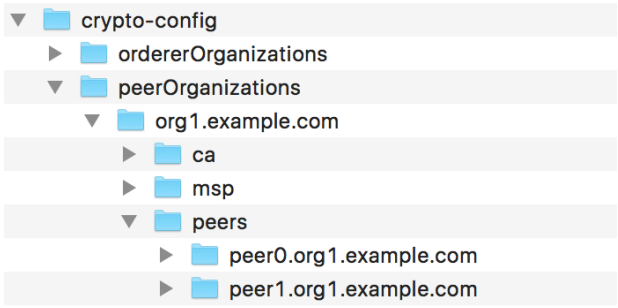
- ✓ Now write genesis block

```
configtxgen -profile ComposerOrdererGenesis -outputBlock
./composer-genesis.block
```

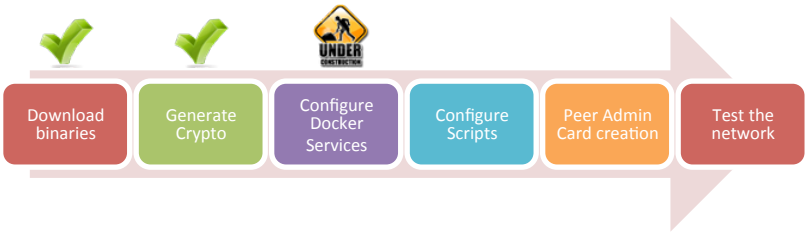
- ✓ Time to define new channel

```
configtxgen -profile ComposerChannel -
outputCreateChannelTx ./composer-channel.tx -channelID
composerchannel
```

On running these commands new crypto material is generated in the "cryto-config" directory as shown below;



Task 2 is complete!



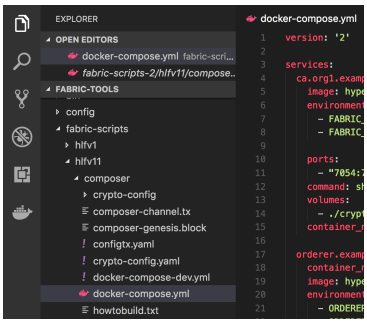
** Please Note: As a reference a sample **Fabric Root** folder **'fabric-tools-2peers.zip'** is provided. Its encouraged that Students should do a Diff of files to counter-check and not land up in errors.

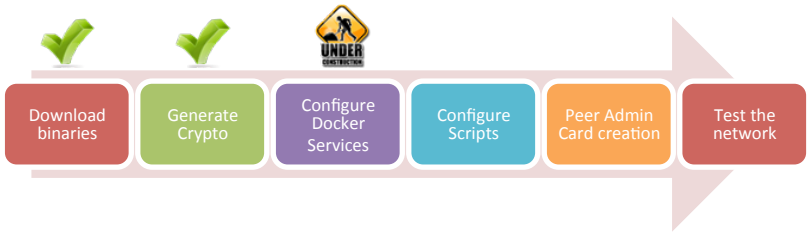
TASK#3: Configure Docker Services

In the docker-compose.yml we already have services 1 to 4 in the listing below. We will add the additional peer1 & couchdb1 services.

1. ca.org1.example.com [Update crypto-material]
2. orderer.example.com
3. peer0.org1.example.com
4. couchdb
5. peer1.org1.example.com [Add New]
6. couchdb1 [Add New]

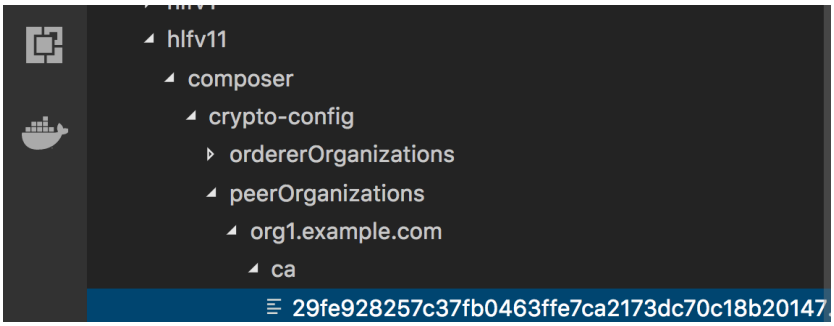
Step 1: Using Visual Studio Code Open the "fabric-tools" folder and open the "docker-compose.yml" of hlfv11 for editing





Step 2: Configure the Certificate Authority, using the certificates that have been generated newly. In the command section of ca.org1.example.com make sure to use the proper private key file.

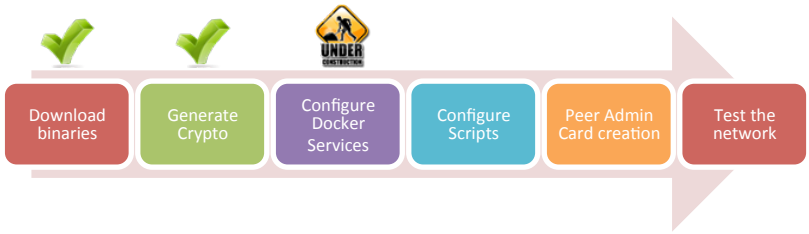
Private Key File located under composer/crypto-config/peerOrganizations/



- ✓ Use this **File Name** – ends with **_sk** in the **"docker-compose.yml"**

Step 3: Add the new "peer1.org1.example.com:" section in the **"docker-compose.yml"**

Please Note: Take special care about the changes marked in image below;




```
peer1.org1.example.com:
  container_name: peer1.org1.example.com
  image: hyperledger/fabric-peer:$ARCH-1.1.0
  environment:
    - CORE_LOGGING_LEVEL=debug
    - CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_PEER_ID=peer1.org1.example.com
    - CORE_PEER_ADDRESS=peer1.org1.example.com:7051
    - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
    - CORE_PEER_LOCALMSPID=Org1MSP
    - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
    - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
    - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric
  command: peer node start
  ports:
    - 8051:7051
    - 8053:7053
  volumes:
      - /var/run/:/host/var/run/
      - ./:/etc/hyperledger/configtx
      - ./crypto-config/peerOrganizations/org1.example.com/peers/peer1.org1.exampl
      - ./crypto-config/peerOrganizations/org1.example.com/users:/etc/hyperledger/
  depends_on:
    - orderer.example.com
    - couchdb1
```

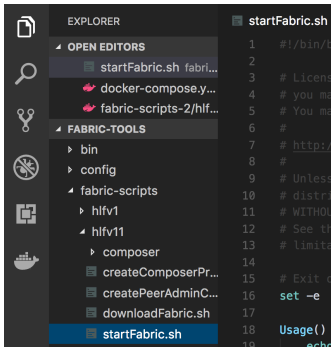
Step 4: Add the new “couchdb” section in the **“docker-compose.yml”**

```
couchdb1:
  container_name: couchdb1
  image: hyperledger/fabric-couchdb:$ARCH-0.4.6
  ports:
    - 6984:5984
  environment:
    DB_URL: http://localhost:5984/member_db
```

Task 3 is complete!

TASK#4: Configure Scripts – Join Peer1 to the channel

Step 1: Using Visual Studio Code Open the "fabric-tools" folder and open the "startFabric.sh" of hlfv11 for editing

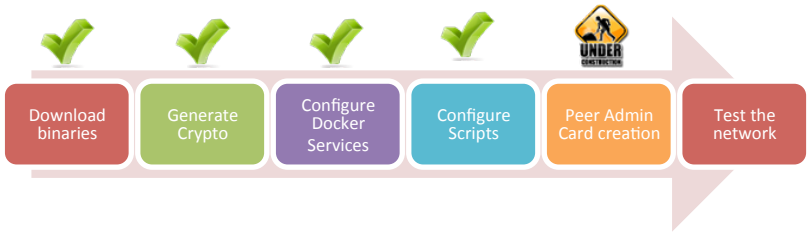


Step 2: Channel is already created and peer0 is added to the channel as shown below:

Add following lines below it to add peer1 to the channel

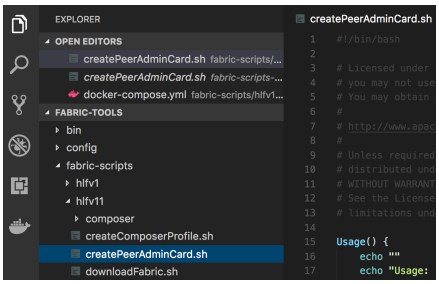
```
# # Fetch channel block from orderer
docker exec -e
"CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin
@org1.example.com/msp" peer1.org1.example.com peer
channel fetch config -o orderer.example.com:7050 -c
composerchannel

# # Join peer1.org1.example.com to the channel.
docker exec -e
"CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin
@org1.example.com/msp" peer1.org1.example.com peer
channel join -b composerchannel_config.block
```

Task 4 is complete!



TASK#5: Peer Admin Card creation

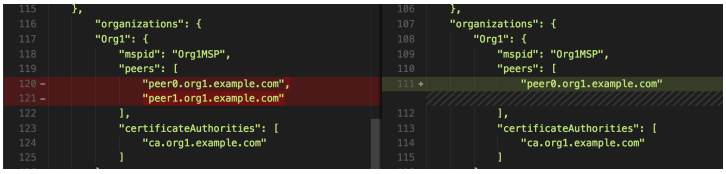
Step 1: Using Visual Studio Code Open the "fabric-tools" folder and open the "createPeerAdminCard.sh" of hlfv11 for editing

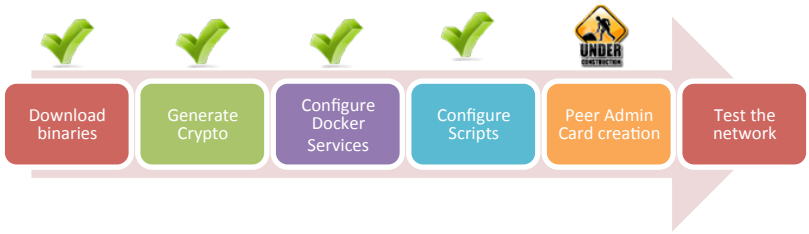


Step 2: Update the connection string to add **peer1** as shown in the images below:

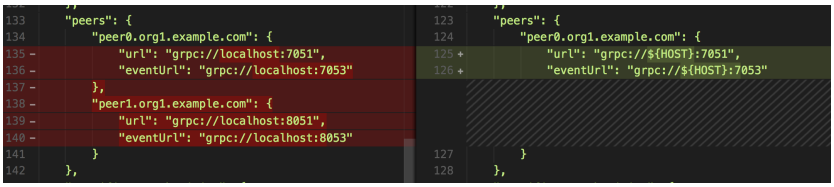
#1:

#2:





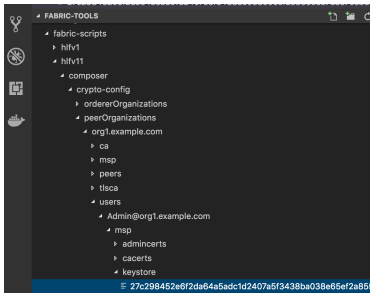
#3:



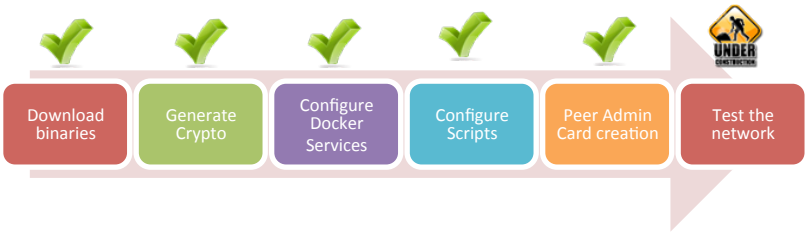
Step 3: Update the PRIVATE KEY for the MSP as we did in TASK#3 Step-2.

LOCATION for Key:

crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore/



Task 5 is complete!



TASK#6: Test the network

TASK#6.1: Visualize Docker Setup

Step 1: Open the terminal window and CD into the Fabric's root folder directory

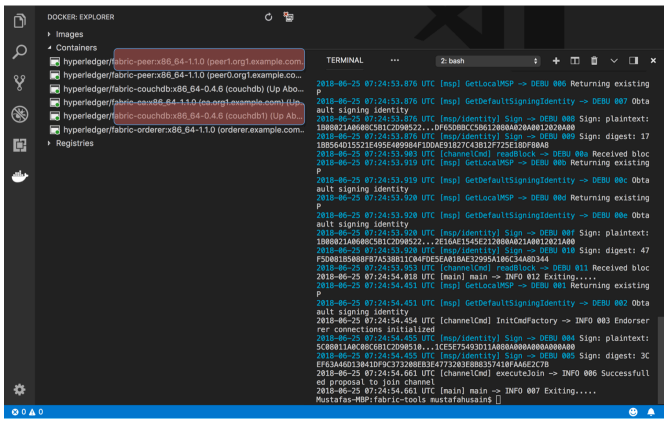
```
cd $HOME/fabric-tools
```

Step 2: Start Fabric using the following command

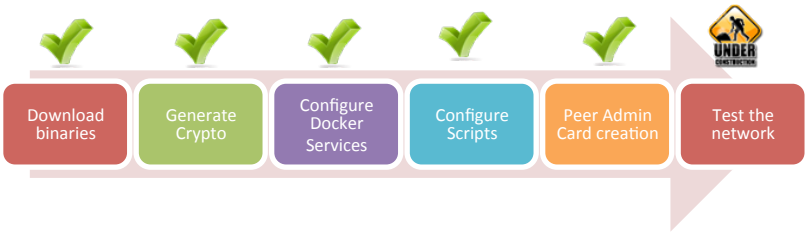
**Please ensure docker service is running

```
./startFabric.sh
```

Fabric should now start with Peer0 and Peer1 along with new couchdb1



Task 6.1 is complete!



TASK#6.2: Launch University-Usecase example

Step 1: Unzip “chapter05.zip” code and CD into the “chapter05” directory

Step 2: Run NPM install to install all required dependencies

```
npm install
```

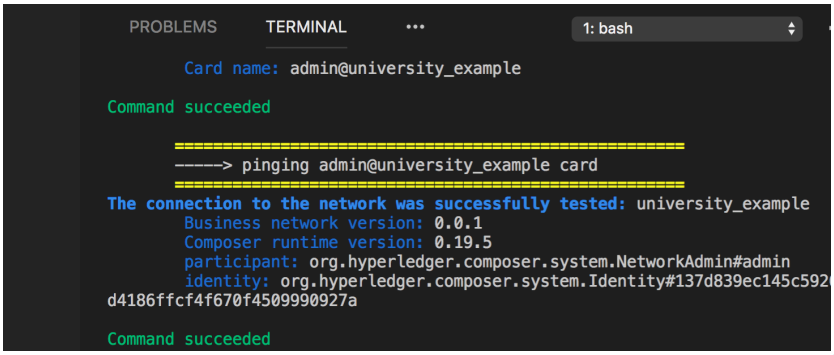
Step 3: CD in the scripts directory

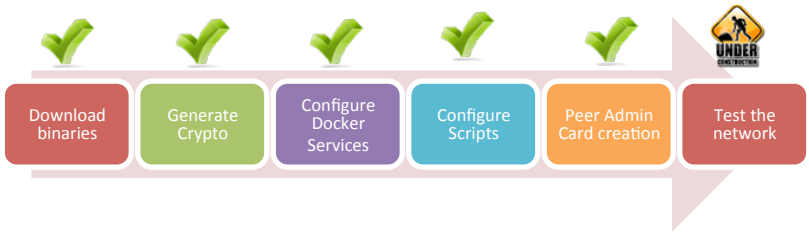
```
cd script
```

Step 4: Run buildAndDeploy Script

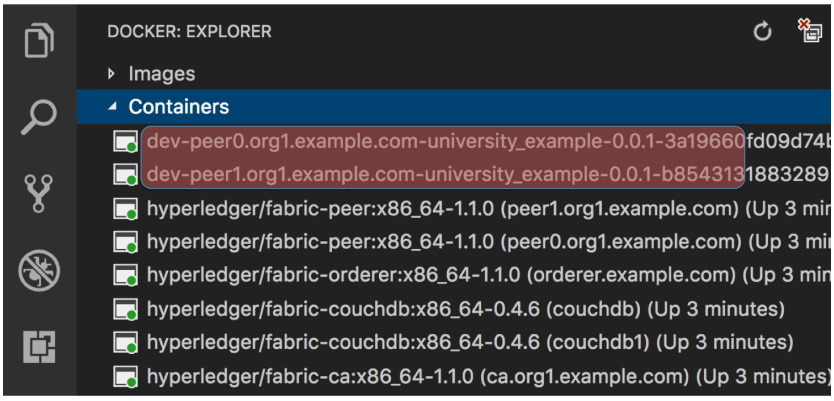
```
./buildAndDeploy.sh
```

On Successful completion of command

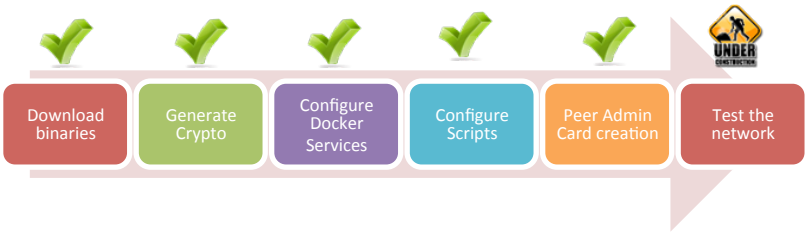




Composer uses the “connection” profile we added to the PeerAdminCard and deploys university example chaincode to both peers



Task 6.2 is complete!



TASK#6.3: Stop 1 Peer and Validate Blocks

Student are encouraged to complete this validation on their own. Few minimal pointers shall be provided which are essential.

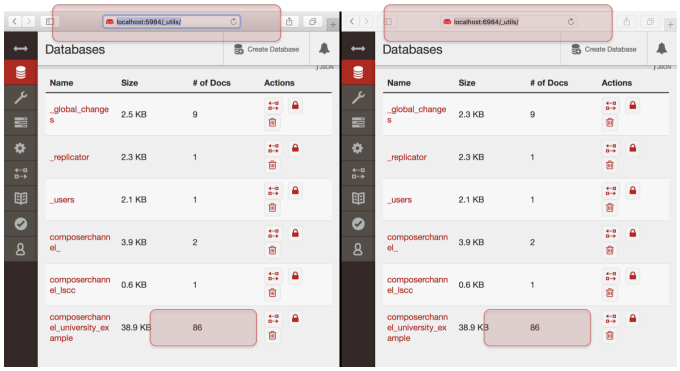
Step 1: Post deploying the "chapter05" University usecase run the UI using the following command from the 'script' folder

```
./run.sh
```

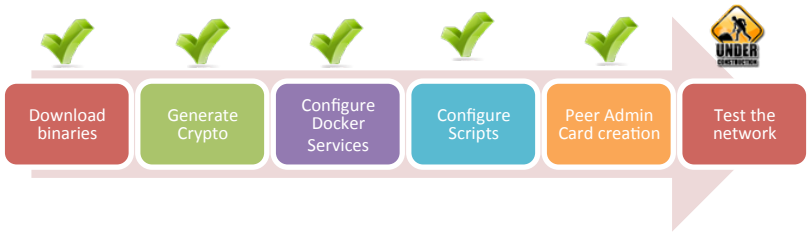
Step 2: Open couchDB database admin/util page using separate browsers:

For Peer0 : couchDB
URL: localhost:5984/_utils/

For Peer1 : couchDB1
URL: localhost:6984/_utils/



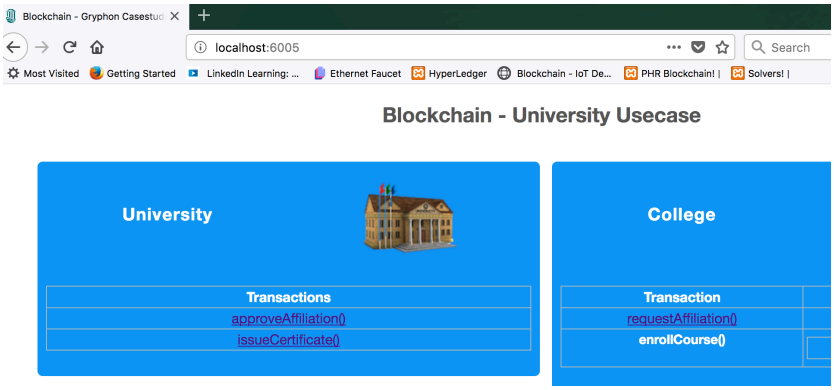
Please Note: Note that both the blocks are in sync as shown highlighted



Step 3: Open UI using the URL as shown up after running ./run.sh script in step-1

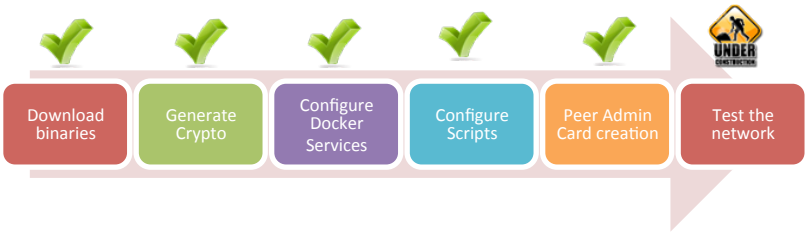
```
Command succeeded

Mustafas-MBP:script mustafahusain$ ./run.sh
Listening locally on port 6005
```

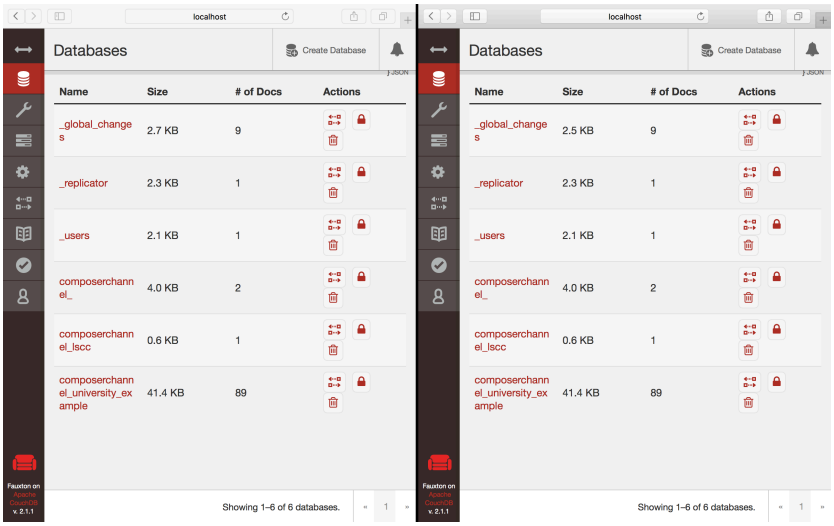
URL: localhost:6005



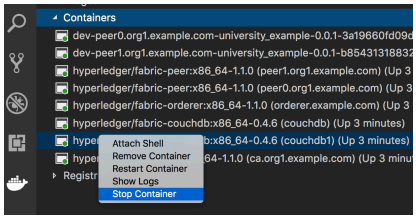
Step 4: Run requestAffiliation() Transaction

Step 5: Refresh both DB browsers and check that the blocks are updated

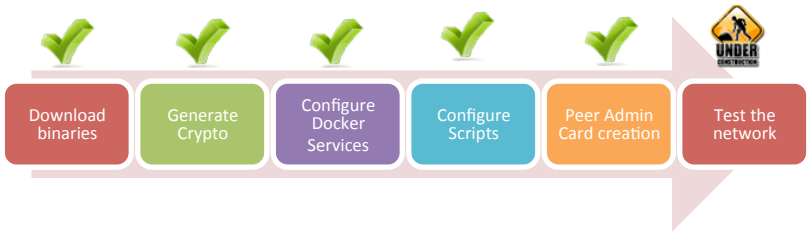




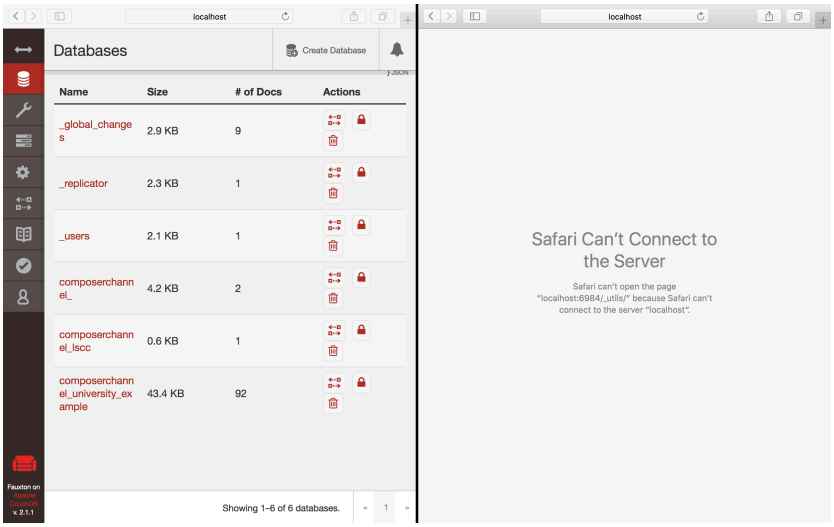
Step 6: Bring down couchdb1 service



Step 7: Run requestAffiliation() Transaction Again

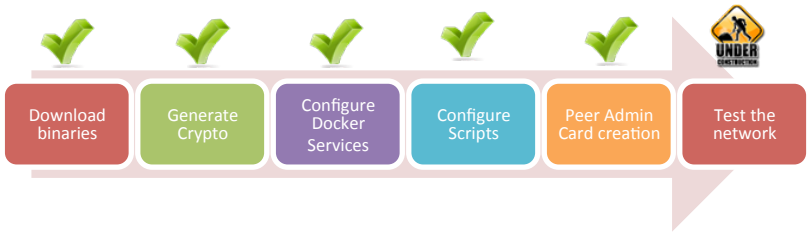


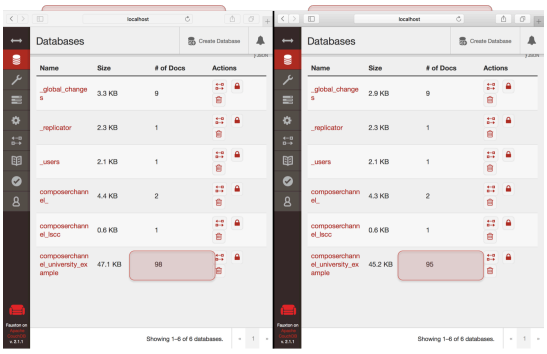
Step 8: Refresh both db browser. One DB should reflect the new block creation. Other should be down.



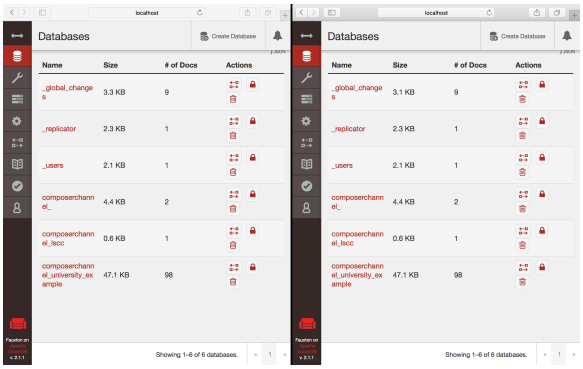
Step 9: Bring Up the couchdb service again – Using restart command

Step 10: Verify the blocks again: You will see a difference in Number of blocks





Step 11: Now bring up the peer1.example.com node as well using restart command



Orderer Sync the blocks and now it starts reflecting on the restarted peer couchdb.

Task 6.3 is complete!

SUMMARY

In this chapter we

- Created a Business network with 1 organization and 2 peers
- Generated the crypto material for the organizations and peers
- Created CA, Orderer and peer services
- Defined the channel (composerchannel)
- Added Peers to the Channel
- Installed university Usecase Chaincode on the peers
- Tested the Peers

Students are further encouraged to explore deeper and play with the Hyperledger business network & the explorer for better learning.

CHAPTER 8: Adding a new organization to the existing Network.

Theory

Congratulations!!! Our TRUSTED Academic Certificate on the Blockchain concept was a big hit!!! Now many other universities want their certificates also on the blockchain…

Problem Statement:

- All other Universities don't want to create their own blockchain network. Else there would be hundreds of such network and if someone as a verifier who wants to validate the certificate authenticity would first have to find the respective university website for validation or it would never work.
- Each University would want to have their own set of Admin users and participant to be able to manage the registration and certification process.
- It has to be a distributed Application and NOT centrally owned
- If we create a new network from scratch – existing data will be washed out and no more be a part of blockchain decentralized network

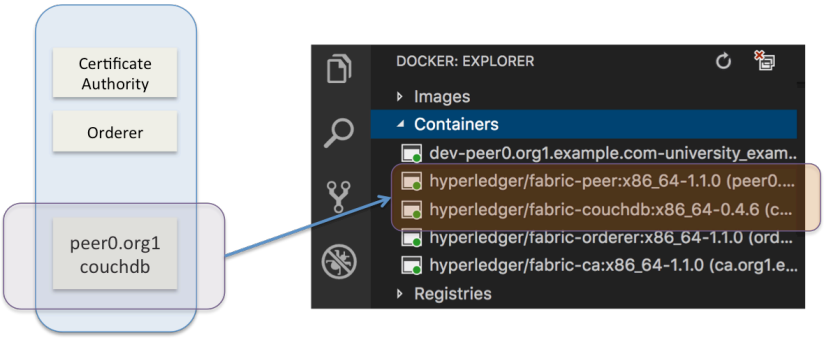
Solution:

- Add a new Organizations to the existing system

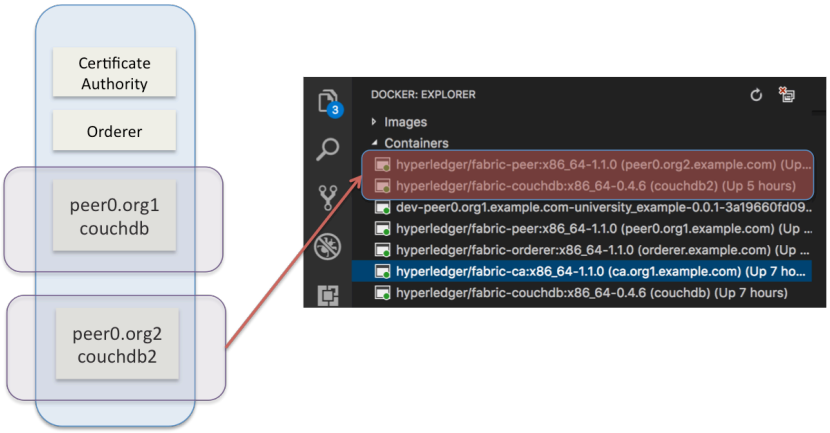
Business Network

Current Setup

Our current composer business network compromises of the following:



New Setup



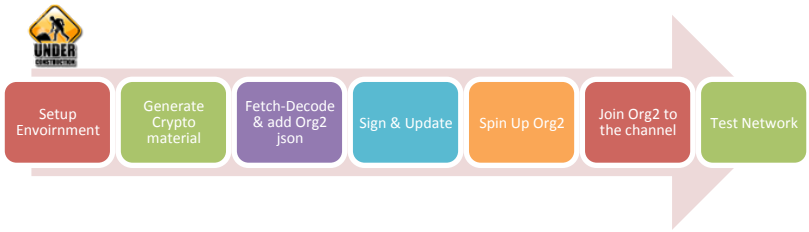
To be able to add new Organization to the existing system we will need following steps:

- Create crypto material and artifacts for the new Organization
- Decode the existing configuration block and update with new organization details
- Encode; sign and update orderer with new configuration transaction
- Spin up new Organization and peers/couchdb
- Join new Org to the existing channel

In this chapter we will add a new organization to our existing business network.

Lab Exercise 8: Adding a new Org to existing Hyperledger Fabric Composer Network

1. Setup Envoirnment
    a) Copy org2-artifact files necessary for adding envoirnment & download binaries + setup path
    b) [Optional] Make changes to existing docker-compose yaml – (Just to validate intermediate steps. And we are going as expected)
    c) Spin Up the university example if NOT already running
2. Generate Crypto material
    a. Generate Crypto for new Org
    b. Generate configuration json for new Org
    c. Copy crypto material to crypto-config folder
3. Fetch-Decode existing configurations & add Org2 json
4. Sign & Update new configurations with Org2
5. Spin Up new Org2 Containers
6. Join Org2 to the channel
7. Test the network
    a. Visualize Docker setup
    b. Using CouchDB
    c. Blockchain Explorer



TASK#1: Setup Environment

** **Please NOTE:** This chapter considers that Our Network has 1 Org and 1 Peer. If you have worked with Lab-7 and have 2 Peer in Org1, the steps still remains the same. NO Change is NEEDED.

TASK#1.a: Copy org2 artifacts & setup fabric binaries

Step 1: Take 'chapter08' and cd into it

```
cd chapter08
```

Step 2: 'org2-artifacts' Folder contain the artifacts for Org2. Lets copy that to the **'fabric-tools'** home directory.

** **Please Note:** In this chapter for compatibility reasons we have renamed Fabric's root directory to **'fabric-tools'**. So If you have **NOT** installed Hyperledger using Chapter#0 of this booklet, root folder name will be **'fabric-dev-servers'** instead

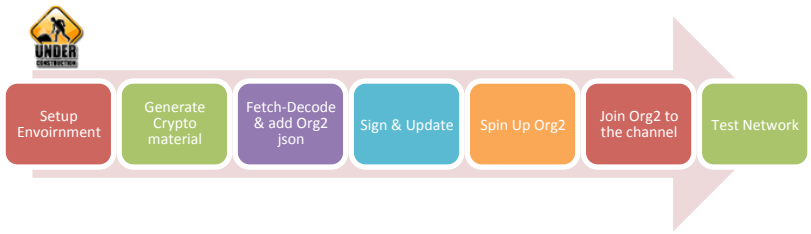
```
mv org2-artifacts $HOME/fabric-tools
```

Step 3: CD into the **'fabric-tools'** home directory

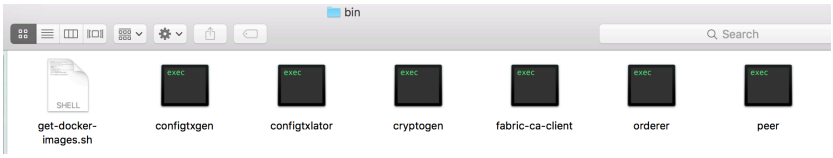
```
cd $HOME/fabric-tools/org2-artifacts
```

Step 4: Execute the following command from within the directory into which you will extract the platform-specific binaries:

```
curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0 -d -s
```



The command above downloads and executes a bash script that will download and extract all of the platform-specific binaries you will need to set up your network and place then in the **bin subdirectory**



Step 5: Add that to your PATH environment variable so that these can be picked up without fully qualifying the path to each binary

```
export PATH=$HOME/fabric-tools/org2-artifacts/bin:$PATH
```

** Don't close the Terminal Window, we will be using the same terminal window for generating crypto-materials

Task 1.a is now Complete!

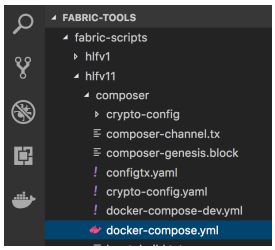
TASK#1.b: Map peer0.org1 volume to host

***** Task#1.b is NOT mandatory for adding the new Org, however for our learning purpose it is HIGHLY recommended***

Step 1: Launch Visual Studio editor and open the 'fabric-tools' folder.

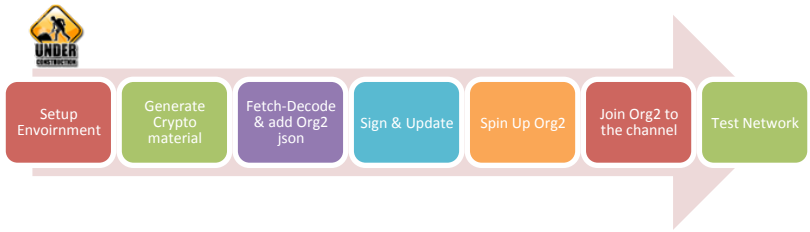
Step 2: Open "docker-compose.yml" file for editing

fabric-tools -> fabric-scripts -> hlfv11 -> composer -> docker-compose.yml



Step 8: Map the 'org2-artifacts' directory in the yaml file.

This is recommended so that we are able to get all the files that will be generated in the ***peer0.org1*** container in the ***org2-artifacts*** folder without having to copy it every time for inspection.

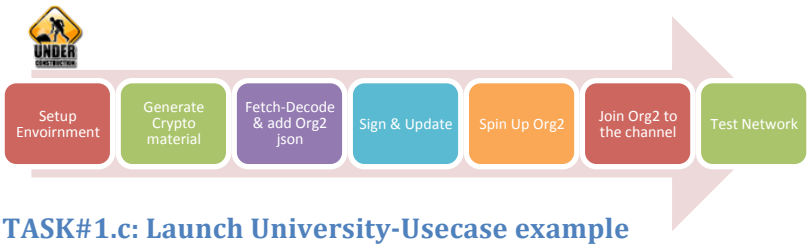



```
peer0.org1.example.com:
  container_name: peer0.org1.example.com
  image: hyperledger/fabric-peer:$ARCH-1.1.0
  environment:
    - CORE_LOGGING_LEVEL=debug
    - CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_PEER_ID=peer0.org1.example.com
    - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
    - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
    - CORE_PEER_LOCALMSPID=Org1MSP
    - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
    - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
    - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb:5984
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric
  command: peer node start
  ports:
    - 7051:7051
    - 7053:7053
  volumes:
      - /var/run/:/host/var/run/
      - ./:/etc/hyperledger/configtx
      - ./crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp:/etc/hyper
      - ./crypto-config/peerOrganizations/org1.example.com/users:/etc/hyperledger/msp/users
      - ./../../../org2-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/org2-artifacts
  depends_on:
    - orderer.example.com
```

Now we are all set to start our university_example [if NOT already running]

****Note**: Because we have made changes to ***docker-compose.yml*** file we will have to re-launch the network so that the volume mapping exists in the peer container.

Task 1.b is now Complete!



TASK#1.c: Launch University-Usecase example

****Note**: If you have University-Usecase already running and have NOT used step #1.b, you may choose NOT to re-run the example to preserve the existing state of the blockchain. For new students it **recommended** to re-run following step to avoid any un-desired behaviors

Step 1: Unzip "chapter05.zip" code and CD into the "chapter05" directory

Step 2: Run NPM install to install all required dependencies

```
npm install
```

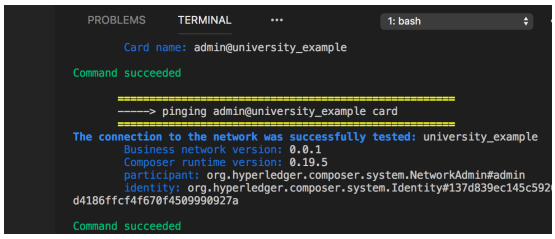
Step 3: CD in the scripts directory

```
cd script
```

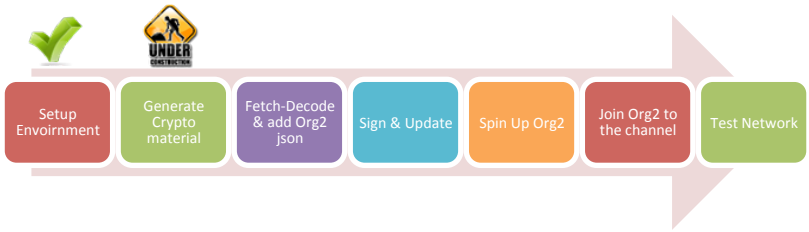
Step 4: Run buildAndDeploy Script

```
./buildAndDeploy.sh
```

On Successful completion of command



Task #1.c is now Complete!



TASK#2: Generate Crypto Material for Org2

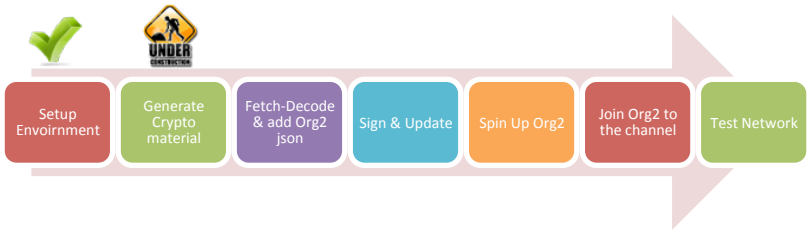
In task#1 we copied '***org2-artifacts***' folder to the '**fabric-tools**' folder it has following files:

- **org2-crypto.yaml :** It Defines
  - The name of new Org,
  - Domain name for the org and
  - Number of peers for which crypto material has to be generated
  - ** We are using USERS – count as '0' hence we will have only Admin user generated

```
PeerOrgs:
  # ---------------------------------------------------------------------------
  # Org3
  # ---------------------------------------------------------------------------
  - Name: Org2
    Domain: org2.example.com
    Template:
      Count: 1
    Users:
      Count: 0
```

- **configtx.yaml :** Which Defines
  - The MSP id for the new ORG
  - Directory path where MSP crypto material need to be placed
  - Anchor peers and the ports

```
Organizations:
    - &Org2
        # DefaultOrg defines the organization which is used in the samp
        # of the fabric.git development environment
        Name: Org2MSP

        # ID to load the MSP definition as
        ID: Org2MSP

        MSPDir: crypto-config/peerOrganizations/org2.example.com/msp

        AnchorPeers:
            # AnchorPeers defines the location of peers which can be us
            # for cross org gossip communication.  Note, this value is
            # encoded in the genesis block in the Application section c
            - Host: peer0.org2.example.com
              Port: 7051
```



Step 1: Using the same above terminal where we have setup path for our binaries we will use it to generate crypto materials

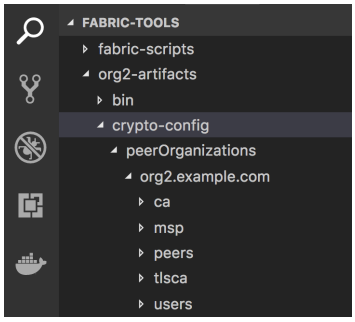
Now, move to the directory for org2 artifacts:

```
cd $HOME/fabric-tools/org2-artifacts/
```

Step 2: Run the following command to generate the crypto material

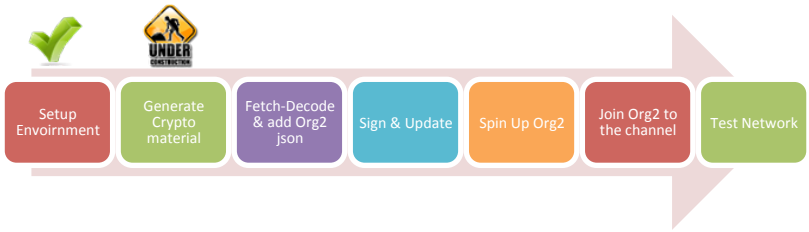
```
cryptogen generate --config=./org2-crypto.yaml
```

A folder within the '**org2-artifacts**' directory with name '**crypto-config'** is created which has org2.example.com as shown below;

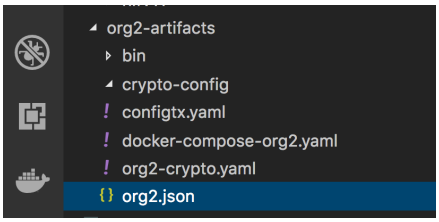


Step 2: Also generate the config json for Org2 with these crypto material recently generated

```
configtxgen -printOrg Org2MSP > org2.json
```



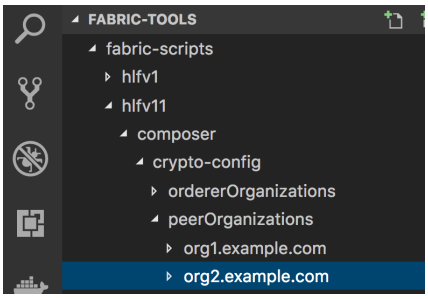
Config file with name org.json is created



Step 3: Let's move the '**org2.example.com**' directory to the peerOrganization folder where org1 crypto material already exists

```
mv ./crypto-config/peerOrganizations/org2.example.com
./../fabric-scripts/hlfv11/composer/crypto-
config/peerOrganizations
```

Post above command we get the following directory structure;

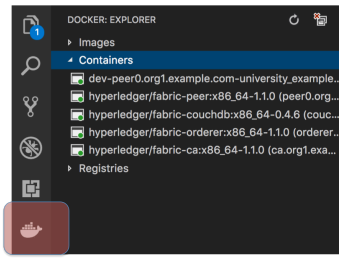


Task #2 is now Complete!

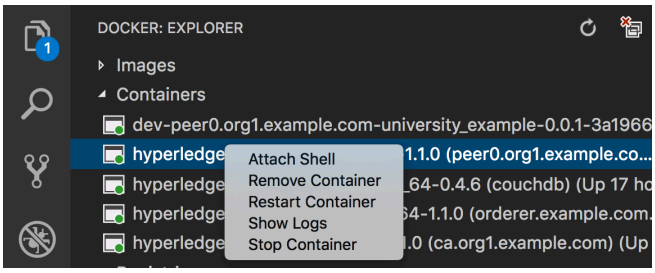
TASK#3: Fetch current config block and append the new Org2 material

Step 1: Under visual code editor we have already installed docker plugin. Launch **Docker Explorer** as shown below;

** Click the tab in RED below for docker explorer



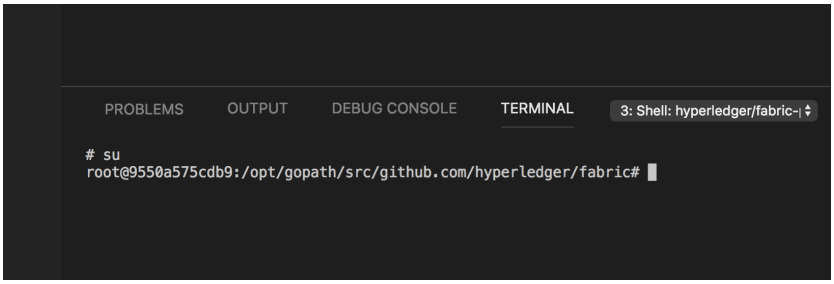
Step 2: Right click on the '**peer0.org1.example.com**' container and click 'Attach Shell' option to move into peer0 container of org1 as shown below;



Step 3: A shell terminal for peer0 is started in Visual Code. Type following command to switch to super user


```
su
```

After this command we should be able to see the following in the VS code terminal window;



Step 4: Use the following command to install '**curl**' and '**jq**' utilities on the peer0 container. JQ utility will be use for json manipulations

*** Wait for sometime as it might take a long time and look like the command didn't work;

```
apt-get -y -qq update && \
apt-get install -y -qq curl && \
apt-get clean
```

```
curl -o /usr/local/bin/jq
http://stedolan.github.io/jq/download/linux64/jq && \
chmod +x /usr/local/bin/jq
```


```
root@9550a575cdb9:/opt/gopath/src/github.com/hyperledger/fabric# apt-get -y -qq update && \
> apt-get install -y -qq curl && \
> apt-get clean

root@9550a575cdb9:/opt/gopath/src/github.com/hyperledger/fabric# curl -o /usr/local/bin/jq http://stedolan.gi
thub.io/jq/download/linux64/jq && \
> chmod +x /usr/local/bin/jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  486k  100  486k    0     0   224k      0  0:00:02  0:00:02 --:--:--  224k
root@9550a575cdb9:/opt/gopath/src/github.com/hyperledger/fabric#
```

Step 5: Download Fabric binaries on the peer0 container using following commands;

```
curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0 -s -d
```

Step 6: Update PATH variable so that the system can locate fabric binaries;

```
export PATH=/opt/gopath/src/github.com/hyperledger/fabric/bin:$PATH
```

** We will now be generating many intermediate files; lets move into the artifact directory we mapped in docker-composer.yaml in Task#1

Step 7: CD into the mapped directory for creating artifacts;

```
cd org2-artifacts
```

Step 8: Start the configtxlator tool in the background so that we can fetch the configurations and translate them to readable json files

```
configtxlator start &
```


```
nfigtxlator] startServer -> INFO 001 Serving HTTP requests on 0.0.0.0:7059

root@c2c0d2194c53:/opt/gopath/src/github.com/hyperledger/fabric#
```

Step 8: Now do some house-keeping to save efforts using variables

Let's make our life easy using some variables: use exports for channel name and the transalator url…

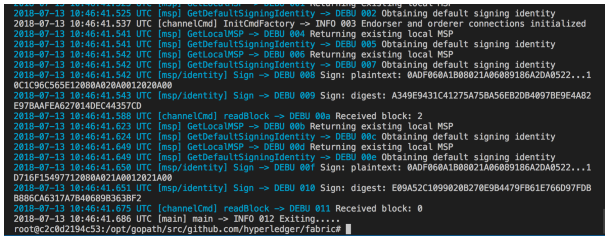
```
export CHANNEL_NAME=composerchannel
```

```
export CONFIGTXLATOR_URL=http://127.0.0.1:7059
```

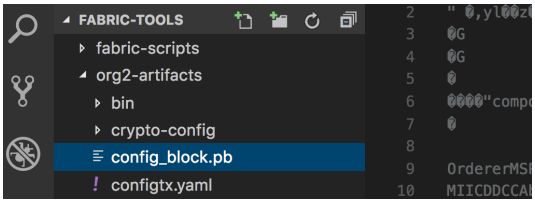
Step 9: Finally… now let's retrieve the current configuration

```
peer channel fetch config config_block.pb -o
orderer.example.com:7050 -c $CHANNEL_NAME
```

We should receive the block 2 – after genesis block for configuration as shown below;



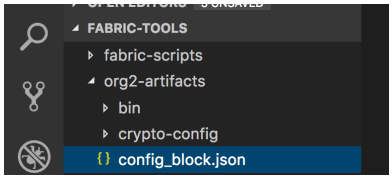
We should be able to see the retrieved '**config_block.pb**' file in our mapped '**org2-artifact**' folder of the host system



Step 10: Using configtxlator we can convert this config into a readable JSON file

```
curl -X POST --data-binary @config_block.pb
$CONFIGTXLATOR_URL/protolator/decode/common.Block > config_block.json
```

On running the above command a new json file is created '**config_block.json**'



Step 11: Using jq tool extract the config section

```
jq .data.data[0].payload.data.config config_block.json >
config.json
```

Step 12: In Task#2 we had created a config file for Org2, lets use this org2.json config file to get the additional configuration settings

```
jq -s '.[0] * {"channel_group": {"groups":
{"Application": {"groups": { "Org2MSP":.[1]}}}}}'
config.json org2.json >& updated_config.json
```

Step 13: Encode both updated and original configurations using configtxlator.

```
curl -X POST --data-binary @config.json
$CONFIGTXLATOR_URL/protolator/encode/common.Config > config.pb
```

```
curl -X POST --data-binary @updated_config.json
$CONFIGTXLATOR_URL/protolator/encode/common.Config >
updated_config.pb
```

Step 15: Compute the config update delta, which represents the changes to the config.

```
curl -X POST -F original=@config.pb -F updated=@updated_config.pb
$CONFIGTXLATOR_URL/configtxlator/compute/update-from-configs -F
channel=$CHANNEL_NAME > config_update.pb
```

Step 16: Now we decode the delta config block to human readable json so that we can wrap it again with header contents that we removed earlier;

```
curl -X POST --data-binary @config_update.pb
$CONFIGTXLATOR_URL/protolator/decode/common.ConfigUpdate >
config_update.json
```

Step 17: Create the final configuration json by wraping the header informations

```
echo
'{"payload":{"header":{"channel_header":{"channel_id":"'$
CHANNEL_NAME'","type":2}},"data":{"config_update":'$(cat
config_update.json)'}}}' > config_update_as_envelope.json
```

Step 18: Convert back readable json to a binary file that is understood by the Fabric envoirnment

```
curl -X POST --data-binary
@config_update_as_envelope.json
$CONFIGTXLATOR_URL/protolator/encode/common.Envelope >
config_update_as_envelope.pb
```

We now have the updated new config file with Org2 additions. In next task we will sign and submit this configuration.

** Don't close the Terminal Window, we will be using the same terminal window for updating configuration to the fabric environment in next TASK

Task #3 is now complete!

TASK#4: Sign & Update the new configuration

Step 1: Using the same terminal window as in Task#3; use the following command to sign the new updated configuration block with peer0 crypto material

```
peer channel signconfigtx -f config_update_as_envelope.pb
```

The terminal output shows the success of signing to ensure everything went right as below;

```
2018-07-13 11:44:22.639 UTC [msp] GetLocalMSP -> DEBU 006 Returning existing local MSP
2018-07-13 11:44:22.639 UTC [msp] GetDefaultSigningIdentity -> DEBU 007 Obtaining default signing iden
tity
2018-07-13 11:44:22.640 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0AA2060A074F7267314D53501
296062D...7269746572731200 2A0641646D696E73
2018-07-13 11:44:22.641 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: ADED87E196360BA3A3A80F68C940
074E9D30B4C73F5221ADE53868AEFE146D44
2018-07-13 11:44:22.642 UTC [msp] GetLocalMSP -> DEBU 00a Returning existing local MSP
2018-07-13 11:44:22.642 UTC [msp] GetDefaultSigningIdentity -> DEBU 00b Obtaining default signing iden
tity
2018-07-13 11:44:22.642 UTC [msp] GetLocalMSP -> DEBU 00c Returning existing local MSP
2018-07-13 11:44:22.642 UTC [msp] GetDefaultSigningIdentity -> DEBU 00d Obtaining default signing iden
tity
2018-07-13 11:44:22.643 UTC [msp/identity] Sign -> DEBU 00e Sign: plaintext: 0ADF060A1B08021A060896A1A
2DA0522...CD06240F8095BECA64BA55F15D3C9870
2018-07-13 11:44:22.643 UTC [msp/identity] Sign -> DEBU 00f Sign: digest: 7274BF53718CB046E799B2DDEF7A
6DB66993B730A920618F49C9F827BFD0E6FA
2018-07-13 11:44:22.646 UTC [main] main -> INFO 010 Exiting.....
root@c2c0d2194c53:/opt/gopath/src/github.com/hyperledger/fabric/org2-artifacts#
```

Step 2: Finally submit to Fabric for update channel configuration transaction using the commands below;

```
peer channel update -f config_update_as_envelope.pb -o
orderer.example.com:7050 -c $CHANNEL_NAME
```

Terminal window logs will show if the update was successful; as shown below

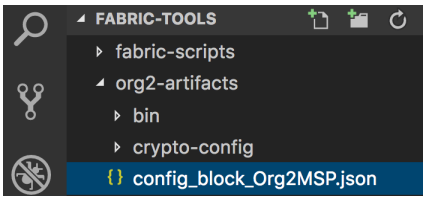
```
2018-07-13 11:47:06.696 UTC [msp] GetLocalMSP -> DEBU 006 Returning existing local MSP
2018-07-13 11:47:06.696 UTC [msp] GetDefaultSigningIdentity -> DEBU 007 Obtaining default signing iden
tity
2018-07-13 11:47:06.696 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0AA2060A074F7267314D53501
296062D...72697465727312002A0641646D696E73
2018-07-13 11:47:06.697 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: 8897A3E201683E808F403E05A55A
BDBCD1CF46274CA0D4C37F8E03E8123D1FCF
2018-07-13 11:47:06.698 UTC [msp] GetLocalMSP -> DEBU 00a Returning existing local MSP
2018-07-13 11:47:06.698 UTC [msp] GetDefaultSigningIdentity -> DEBU 00b Obtaining default signing iden
tity
2018-07-13 11:47:06.700 UTC [msp] GetLocalMSP -> DEBU 00c Returning existing local MSP
2018-07-13 11:47:06.700 UTC [msp] GetDefaultSigningIdentity -> DEBU 00d Obtaining default signing iden
tity
2018-07-13 11:47:06.700 UTC [msp/identity] Sign -> DEBU 00e Sign: plaintext: 0ADF060A1B08021A0608BAA2A
2DA0522...7C700CC0AEBF3F8B17C1E814883E77DB
2018-07-13 11:47:06.701 UTC [msp/identity] Sign -> DEBU 00f Sign: digest: EDC8EBC274BDF9BC40576518A47E
26516309E7F4EC1C78908C1322A332B71BF2
2018-07-13 11:47:06.781 UTC [channelCmd] update -> INFO 010 Successfully submitted channel update
2018-07-13 11:47:06.781 UTC [main] main -> INFO 011 Exiting.....
```

Step 3: Lets also validate by retrieving the updated configuration block from the – Orderer;

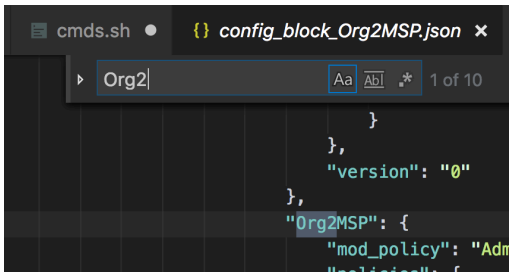
```
peer channel fetch config config_block_Org2MSP.pb -o
orderer.example.com:7050 -c  $CHANNEL_NAME
```

```
curl -X POST --data-binary @config_block_Org2MSP.pb
$CONFIGTXLATOR_URL/protolator/decode/common.Block >
config_block_Org2MSP.json
```

With above commands we fetch and also convert the new configuration block from orderer to a readable json

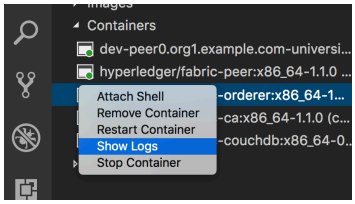


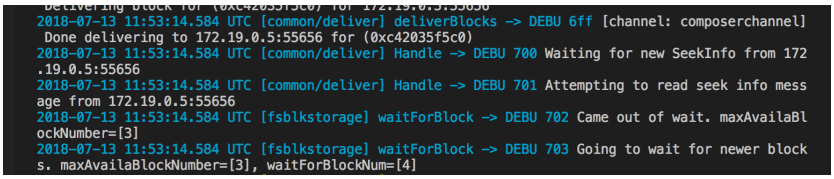
Step 4: Search for 'Org2' in the json file retrieved '**config_block_Org2MSP.json'**



In the configuration retrieved from Orderer we find instances of Org2 – hence we can conclude that – update transaction was a success.

Step 5: Also verify orderer logs as below'





This shows that a new block – '3' was created with the update transaction success.

Task #4 is now complete!

TASK#5: Spin Up Org2

** Now we have updated the channel configuration to accommodate Org2. We can now launch a new peer node with Org2 crypto materials

Under the 'org2-artifacts' folder we copied a '***docker-compose-org2.yaml***' file from the chapter08 file. Lets check what we have in that file;

```
version: '2'

services:

  peer0.org2.example.com:
    container_name: peer0.org2.example.com
    image: hyperledger/fabric-peer:x86_64-1.1.0
    dns_search: .
    environment:
      - CORE_LOGGING_LEVEL=debug
      - CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - CORE_PEER_ID=peer0.org2.example.com
      - CORE_PEER_ADDRESS=peer0.org2.example.com:7051
      - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
      - CORE_PEER_LOCALMSPID=Org2MSP
      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb2:5984
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: peer node start
    ports:
      - 8051:7051
      - 8053:7053
    volumes:
        - /var/run/:/host/var/run/
        - ./:/etc/hyperledger/configtx
        - ./crypto-config/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/msp:/etc/hyperledger/peer/msp
        - ./crypto-config/peerOrganizations/org2.example.com/users:/etc/hyperledger/msp/users
    depends_on:
      - couchdb2

  couchdb2:
    container_name: couchdb2
    image: hyperledger/fabric-couchdb:x86_64-0.4.6
    ports:
      - 6984:5984
    environment:
      DB_URL: http://localhost:5984/member_db
```

It is similar to the original '**docker-composer.yaml'** file but has only 1 peer service with name peer0.org2.example.com which was specified in the channel configurations

Step 1: Open a terminal window and CD into the '**org2-artifacts**' folder

```
cd $HOME/fabric-tools/org2-artifacts
```

Step 2: We have all the crypto material in the '**composer**/crypto-config' folder of hlf11 where we copied the org2 crypto as well in Task#1.

Lets move the '**docker-composer-org2.yaml'** file to the composer folder

```
mv ./docker-compose-org2.yaml $HOME/fabric-tools/fabric-scripts/hlfv11/composer
```

Step 3: CD into the composer folder to launch docker container

```
cd $HOME/fabric-tools/fabric-scripts/hlfv11/composer
```

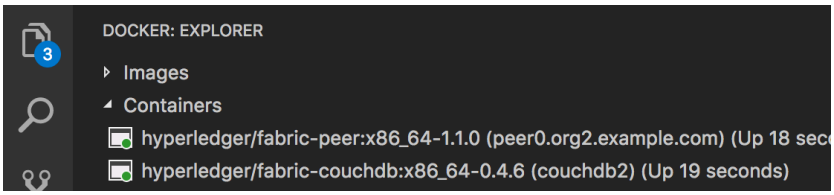
Step 4: Retrieve machine architecture into a variable so that appropriate docker container can be launched

```
ARCH=`uname -m`
```

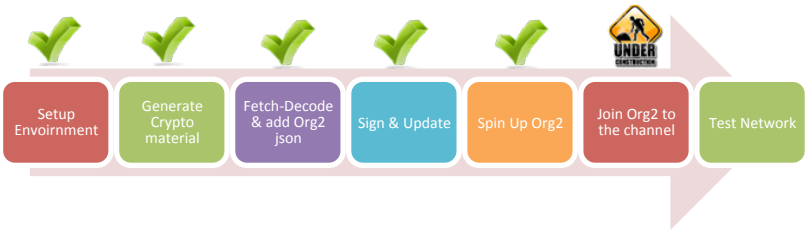
Step 5: Launch the new peer0.org2 node docker container

```
ARCH=$ARCH docker-compose -f docker-compose-org2.yaml up
```

Step 6: Use Docker explorer to validate that both 'peer0.org2' and 'couchdb2' nodes have successfully started;



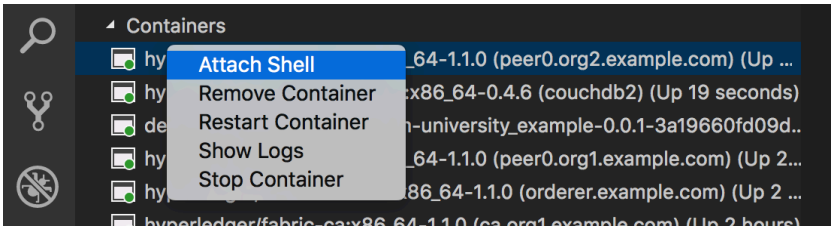
Task#5 is now complete!



TASK#6: Org2 Joins the channel

Step 1: Launch Visual Studio code and open the Docker explorer as in task#5

Step 2: This time we want to launch bash terminal for '**peer0.org2**' so that Org2 can join the channel. Use image below;



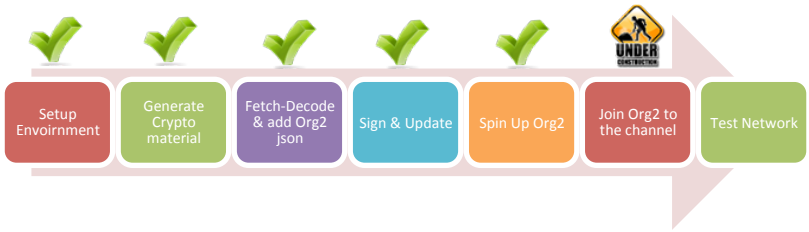
Step 3: In the terminal window for **peer0.org2** use following command to switch to super user or use bash command

```
su
```

```
388731de37d587843d6e /bin/sh
# su
root@f029b36b0b3b:/opt/gopath/src/github.com/hyperledger/fabric#
```

Step 4: Update the Environment variable so that the peer can use the correct crypto materials;

** As a part of crypto material generation for Org2 we had created an Admin user only which has the authority to transact with Orderer when dealing with Org2




```
CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@
org2.example.com/msp
```

Step 5: Set channel name to be used in further commands;

```
export CHANNEL_NAME=composerchannel
```

Step 6: We now need to fetch the genesis block in order for the new org to join. Use the following command;

```
peer channel fetch 0 composerchannel.pb -o
orderer.example.com:7050 -c composerchannel
```

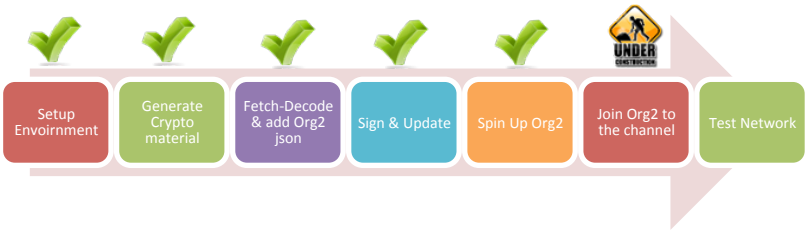
We are fetching the genesis block here!!! Hence will receive block 0 as shown below;

```
2018-07-13 13:15:17.106 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-07-13 13:15:17.106 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-07-13 13:15:17.112 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2018-07-13 13:15:17.116 UTC [msp] GetLocalMSP -> DEBU 004 Returning existing local MSP
2018-07-13 13:15:17.116 UTC [msp] GetDefaultSigningIdentity -> DEBU 005 Obtaining default signing identity
2018-07-13 13:15:17.118 UTC [msp] GetLocalMSP -> DEBU 006 Returning existing local MSP
2018-07-13 13:15:17.118 UTC [msp] GetDefaultSigningIdentity -> DEBU 007 Obtaining default signing identity
2018-07-13 13:15:17.118 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0ADF060A1B08021A0608E5CBA2DA0522...61E3C
1A0012021A00
2018-07-13 13:15:17.118 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: 35FA60C7F8467E4F37A844EE501025FC092F2374FAF
4118B9
2018-07-13 13:15:17.154 UTC [channelCmd] readBlock -> DEBU 00a Received block: 0
2018-07-13 13:15:17.159 UTC [main] main -> INFO 00b Exiting.....
```

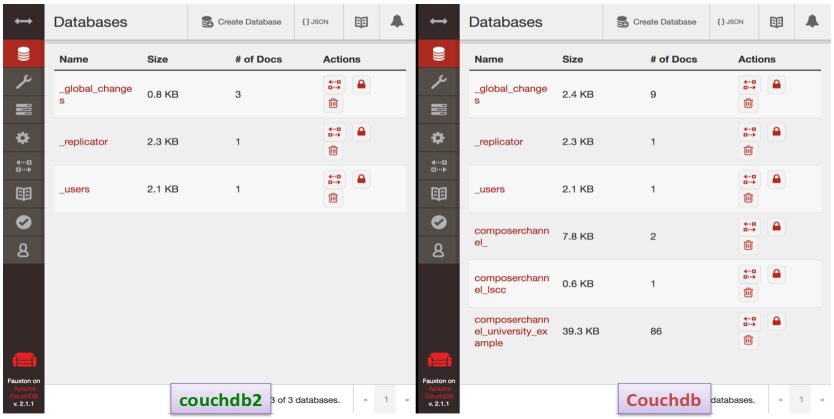
Step 7: We have couchdb and couchdb2 up and running; let's check out what documents we have in both so far;

Launch internet explorer and go to the following URL for **couchdb**:
http://localhost:5984/_utils/

Step 8: Launch second internet explorer and go to the following URLfor **couchdb2**:
http://localhost:6984/_utils/



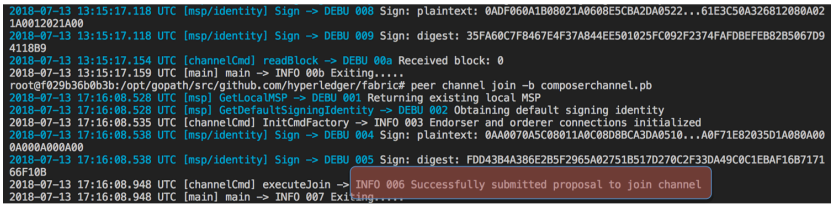
Step 9: Compare that couchdb2 still does not have the channel: composerchannel transaction details NOR our university_example BNA chaincode & transactions



Step 10: Join Org2 to the existing network now using the composerchannel genesis block file & the following command;

```
peer channel join -b composerchannel.pb
```

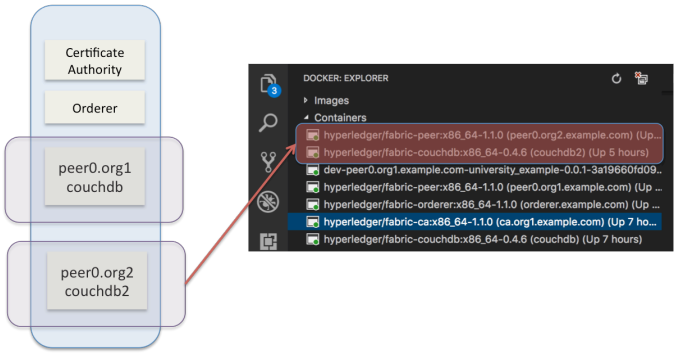
Terminal log shows that Org2 has been successfully inducted to the channel



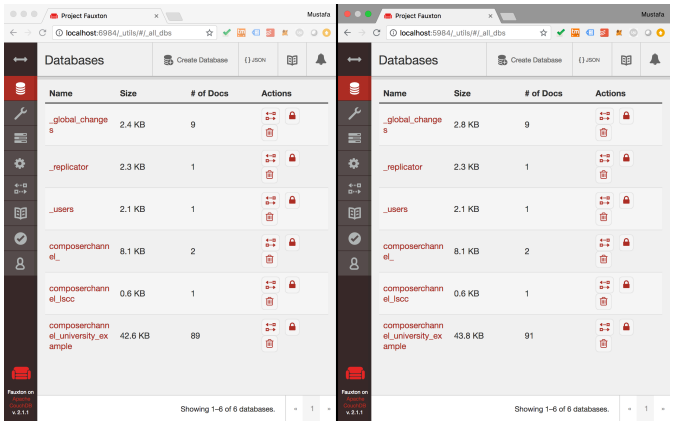
Task#6 is now complete!

TASK#7 Test Network

Step 1: Visualize that the docker containers for couchdb2 and peer0.org2.example.com both exists;



Step 2: Check both couchdb again now as done in Task#6



Please Note: In above couchdb & couchdb2 utils page side by side

- Composer channel table and corresponding entries now exists for Org2
- It also has university_example BNA transaction documents

Step 3: Validate using Hyperledger Blockchain explorer;

**Students are encouraged to Utilize-Chapter 6 to setup blockchain explorer and view complete Org1 & Org2 artifacts

Task #8 is complete!

SUMMARY

In this chapter we

- Decoded the existing network/channel configuration
- Generated the crypto material for the second organization and peer
- Updated the channel configuration setup with new Org2
- Launched containers for the new Org2
- Added new Org2 to the channel

Students are further encouraged to explore deeper and play with the Hyperledger business network & the explorer for better learning.

CHAPTER 9: Scaling Hyperledger on Multiple Hosts

Theory

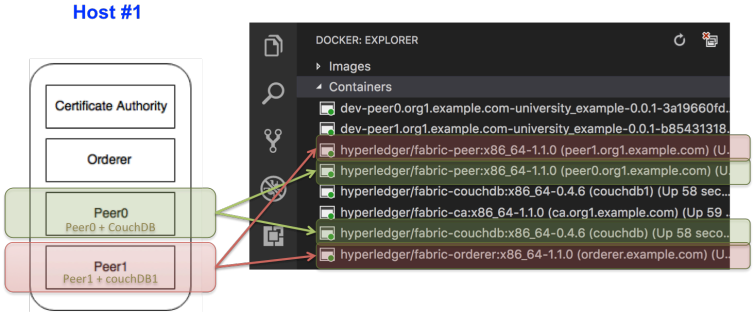
In this chapter we will add a new peer to our business network and deploy it on a separate host machine.

University Usecase – New Setup

Host#1:

- CA – ca.org1.example.com
- Orderer – orderer.example.com
- Peer0 – peer0.org1.example.com
- Couchdb – couchdb
- Peer1 – peer1.org1.example.com
- Couchdb1 – couchdb1

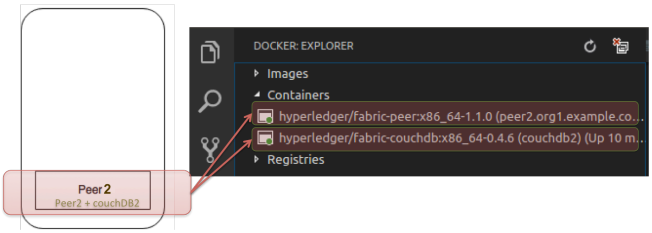
Host #1

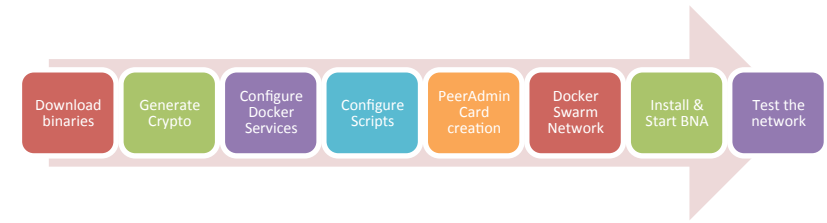


Host#2:

- Peer2 – peer2.org1.example.com
- Couchdb2 – couchdb2

Host #2

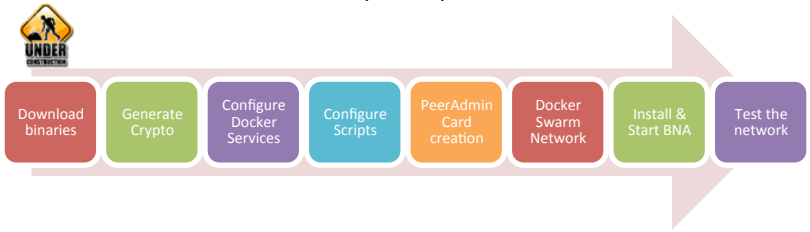




Lab Exercise 9: Scaling Hyperledger on Multiple Hosts

1. Download binaries to generate crypto material
2. Generate Crypto material & copy on second host
    a. Update cypto-config.yaml
    b. Generate Crypto
    c. Copy the crpto material on Host#2
3. Configure Docker Services
    a. Add new Peer + CouchDb to Host#1
    b. Update crypto keys
    c. Create new Docker Service for Host#2
4. Configure Scripts
    a. Update startFabric.sh script to join Peer1 to the channel on Host#1
    b. Update startFabric.sh script to join Peer2 to the channel on Host#2
5. Peer Admin Card creation
    a. Update connection string for Host#1
    b. Make connection string for Host#2
6. Create Docker Swarm network
    a. Create docker swarm on Host#1 & Host#2 join as Manager

7. Install & Start Business Network
8. Test the network
    a. Visualize Docker setup
    b. Validate Couch DB on both Hosts
    c. Install University example & validate blocks on both hosts



**Note: Multi-Host does NOT currently work on Mac system. Hence you will need TWO Ubuntu 16.04 systems

Prerequisites:

- Hyperledger Fabric Composer is installed on both hosts

Color Coding:

All tasks in this chapter shall start with an indication where this task has to be executed. It will be either of the following

- ✓ **Host#1:** To be executed on Host#1 only
- ✓ **Host#2:** To be executed on Host#2 only
- ✓ **Host#1 & Host#2:** To be executed on both hosts

TASK#1: Download Binaries & setup path On Host#1

Step 1: Open the terminal window and CD into the Fabric's root folder directory

** **Please Note:** In this chapter for compatibility reasons we have renamed Fabric's root directory to **'fabric-tools'**. So If you have **NOT** installed Hyperledger using this Book & Scripts, root folder name will be **'fabric-dev-servers'** instead

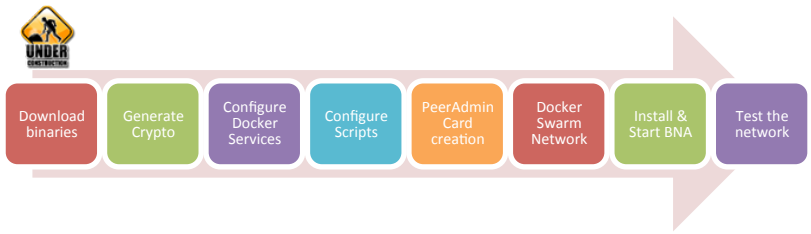
Host#1

```
cd $HOME/fabric-tools
```

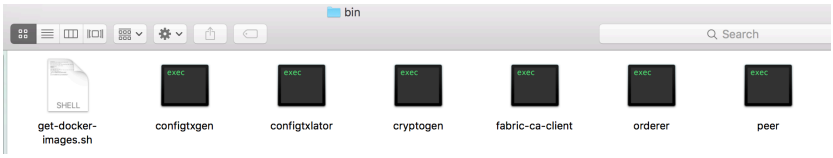
Step 2: Execute the following command from within the directory into which you will extract the platform-specific binaries:

Host#1

```
curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0 -d -s
```



The command above downloads and executes a bash script that will download and extract all of the platform-specific binaries you will need to set up your network and place then in the **bin subdirectory**



Host#1

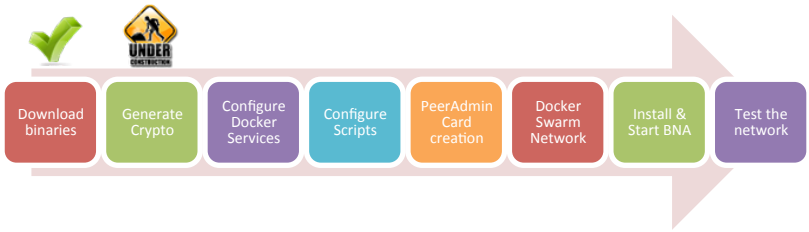
Step 3: Add that to your PATH environment variable so that these can be picked up without fully qualifying the path to each binary

Host#1

```
export PATH=$HOME/fabric-tools/bin:$PATH
```

** Don't close the Terminal Window, we will be using the same terminal window for generating crypto-materials

Task 1 is complete!



TASK#2: Generate Crypto Material

Step 1: Using the same above terminal where we have setup path for our binaries we will use it to generate crypto materials

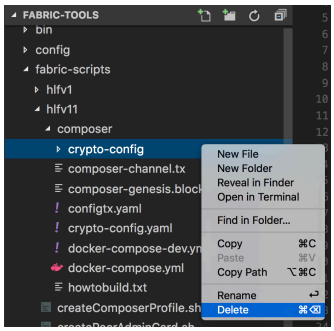
Now, move to the directory Hyperledger 1.1 – composer directory:

Host#1

```
cd $HOME/fabric-tools/fabric-scripts/hlfv11/composer
```

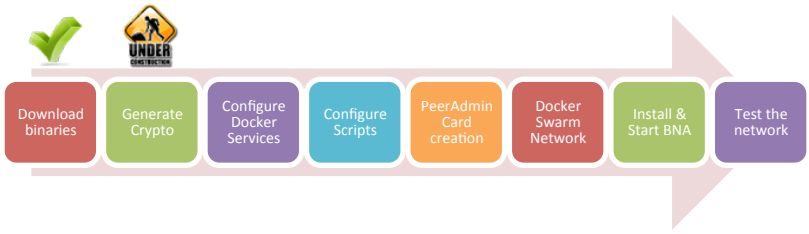
Host#1 & Host#2

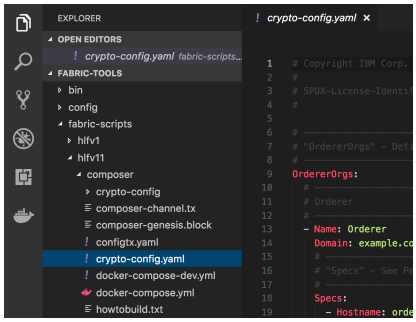
Step 2: Delete the existing 'crypto-config' directory



Host#1

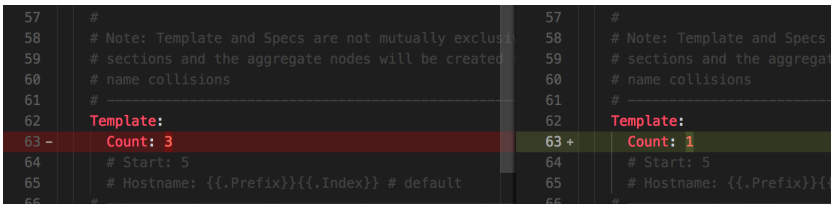
Step 3: Using Visual Studio Code Open the "fabric-tools" folder and open the "crypto-config.yaml" for editing





Host#1

Step 2: We want 3 peers instead of existing 1 – hence we update the crypto material required in "crypto-config.yaml" to **'3' as shown below**

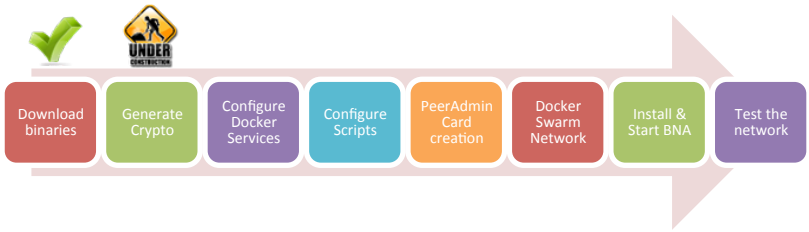


Host#1

Step 3: Generate crypto materials using the following commands:

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

```
export FABRIC_CFG_PATH=$PWD
```



- ✓ Now write genesis block

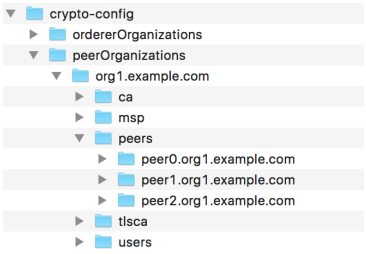
```
configtxgen -profile ComposerOrdererGenesis -outputBlock
./composer-genesis.block
```

- Time to define new channel

```
configtxgen -profile ComposerChannel -
outputCreateChannelTx ./composer-channel.tx -channelID
composerchannel
```

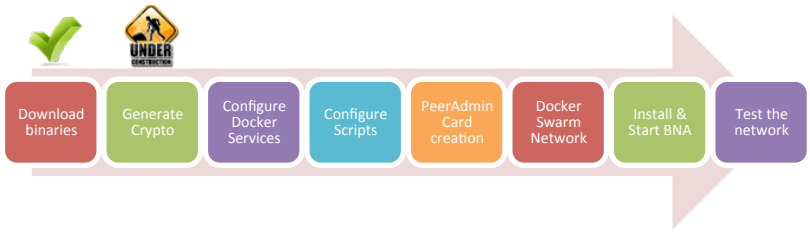
Host#1

On running these commands new crypto material is generated in the "cryto-config" directory as shown below;



Host#1

Step 4: Crypto Materials have been generated now we need to move the crypto materials for Peer2 to Host#2



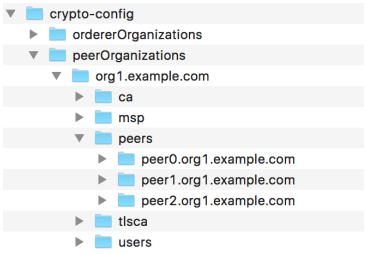
**IMP:

- Ensure that '**crypto-config'** folder on Host#2 in hlfv11/composer directory is deleted
- Replace $USERNAME with the username of Host#2
- Replace $HOST2_IP with the IP of Host#2

```
scp $HOME/fabric-tools/fabric-scripts/hlfv11/composer
$USERNAME@$HOST2_IP:/fabric-tools/fabric-
scripts/hlfv11/composer
```

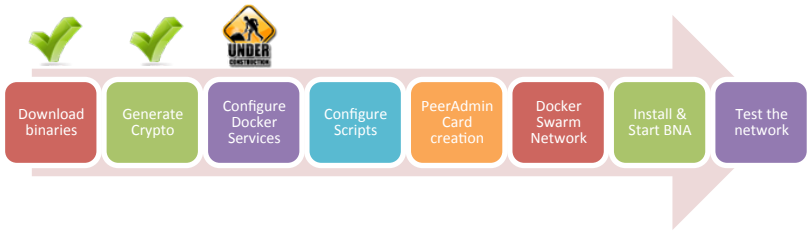
Host#2

Step 5: Ensure that all the crypto material has been transferred to Host#2



***Please Note**: In actual scenario we should only transfer the crypto material relevant to Peer2 to Host#2 but for simplicity of learning here we are transferring all the crypto material.

Task 2 is complete!



** **Please Note:** As a reference a sample **Fabric Root** folder for Host#1 & Host#2 are provided. Its encouraged that Students should do a Diff of files to counter-check and not land up in errors.

TASK#3: Configure Docker Services

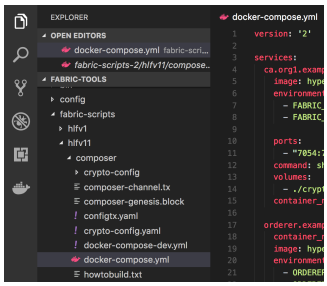
Host#1

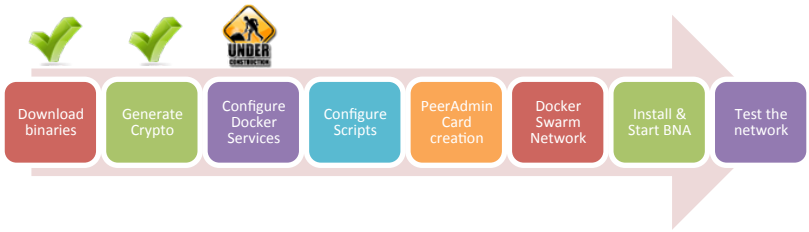
In the docker-compose.yml we already have services 1 to 4 in the listing below. We will add the additional peer1 & couchdb1 services.

1. ca.org1.example.com [Update crypto-material]
2. orderer.example.com
3. peer0.org1.example.com
4. couchdb
5. peer1.org1.example.com [Add New]
6. couchdb1 [Add New]
7. Use Docker Swarm Network [Add New]

Host#1

Step 1: Using Visual Studio Code Open the "fabric-tools" folder and open the "docker-compose.yml" of hlfv11 for editing

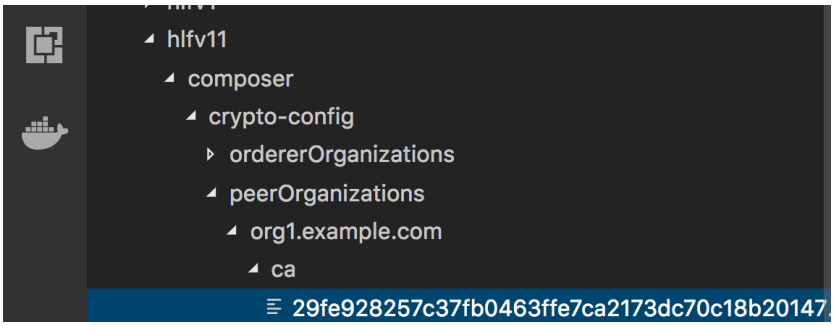




Host#1

Step 2: Configure the Certificate Authority, using the certificates that have been generated newly. In the command section of ca.org1.example.com make sure to use the proper private key file.

Private Key File located under composer/crypto-config/peerOrganizations/

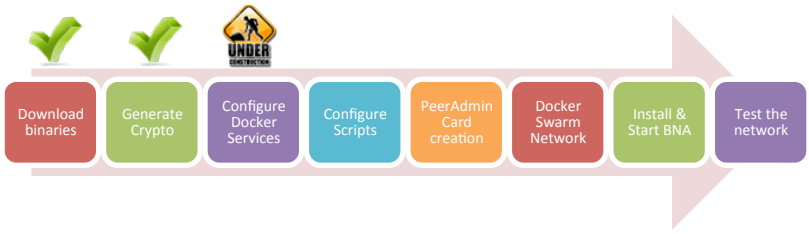


- Use this **File Name** – ends with **_sk** in the **"docker-compose.yml"**

Host#1

Step 3: Add the new "peer1.org1.example.com:" section in the **"docker-compose.yml"**

Please Note: Take special care about the changes marked in image below;

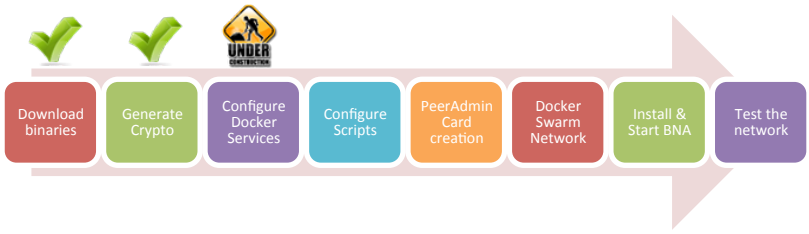



```
peer1.org1.example.com:
  container_name: peer1.org1.example.com
  image: hyperledger/fabric-peer:$ARCH-1.1.0
  environment:
    - CORE_LOGGING_LEVEL=debug
    - CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_PEER_ID=peer1.org1.example.com
    - CORE_PEER_ADDRESS=peer1.org1.example.com:7051
    - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
    - CORE_PEER_LOCALMSPID=Org1MSP
    - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
    - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
    - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric
  command: peer node start
  ports:
    - 8051:7051
    - 8053:7053
  volumes:
      - /var/run/:/host/var/run/
      - ./:/etc/hyperledger/configtx
      - ./crypto-config/peerOrganizations/org1.example.com/peers/peer1.org1.exampl
      - ./crypto-config/peerOrganizations/org1.example.com/users:/etc/hyperledger/
  depends_on:
    - orderer.example.com
    - couchdb1
```

Host#1

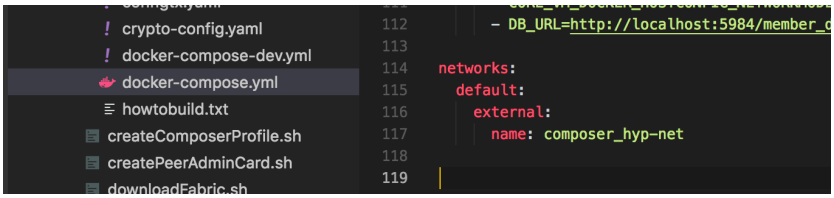
Step 4: Add the new “couchdb” section in the **“docker-compose.yml”**

```
couchdb1:
  container_name: couchdb1
  image: hyperledger/fabric-couchdb:$ARCH-0.4.6
  ports:
    - 6984:5984
  environment:
    DB_URL: http://localhost:5984/member_db
```



Host#1

Step 5: Request docker to use externally created docker swarm network for all the containers; Add the following code to the **"docker-compose.yml"**



Host#1

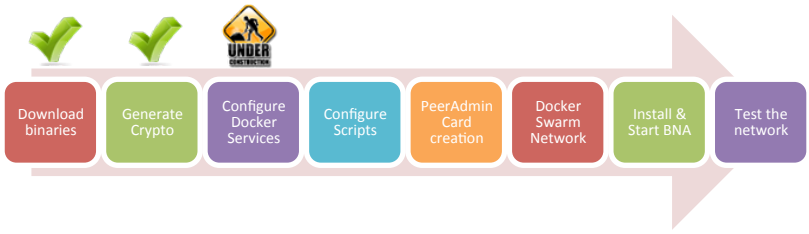
Step 6: Add the new environment variable to each container for using docker swarm as shown below;

Add the above environment variable to all the container services as below:

- ca.org1.example.com
- orderer.example.com:
- peer0.org1.example.com:
- couchdb:
- peer1.org1.example.com:
- couchdb1:

Host#2

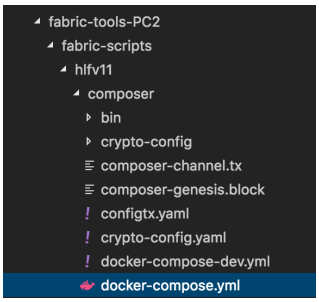
On second host we need only peer2 and corresponding couchdb2 hence in the docker-compose.yml we will add the additional peer2 & couchdb2 services and remove all other services.



Host#2

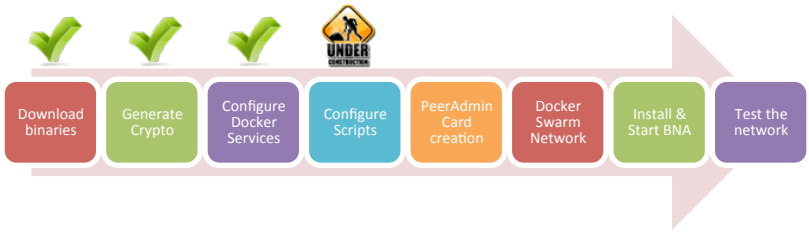
Step 7: Use the **"docker-compose.yml"** file from the provided fabric-tools folder for host#2 i.e "***fabric-tools-PC2***"

On Host#2 replace the content of **"docker-compose.yml"** file with that in the provided **"fabric-tools-PC"** as shown below;



With this we are all set to launch the peers on multiple host. In next task we will make provisions to join the peer on host1 & host2 to the same network

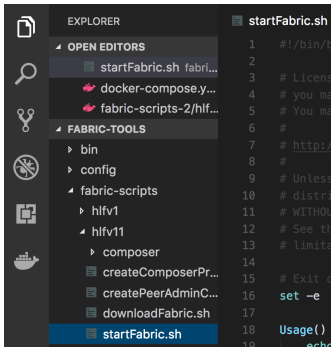
Task 3 is complete!



TASK#4: Configure Scripts – Join Peers to the channel

Host#1

Step 1: Using Visual Studio Code Open the "fabric-tools" folder and open the "startFabric.sh" of hlfv11 for editing



Host#1

Step 2: Channel is already created and peer0 is added to the channel as shown below:

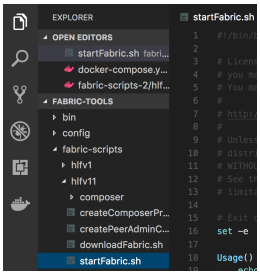
Add following lines below it to add peer1 to the channel

Host#1

```
# # Fetch channel block from orderer
docker exec -e
"CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin
@org1.example.com/msp" peer1.org1.example.com peer
channel fetch config -o orderer.example.com:7050 -c
composerchannel

# # Join peer1.org1.example.com to the channel.
docker exec -e
"CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin
@org1.example.com/msp" peer1.org1.example.com peer
channel join -b composerchannel_config.block
```

Host#2

Step 3: On Host#2 we repeat the same steps as done for adding Peer1 on Host#1. Using Visual Studio Code Open the “fabric-tools” folder and open the “startFabric.sh” of hlfv11 for editing



Host#2

Step 4: As the channel is already created in Host#1 we will fetch these details and add Peer2 to the channel.

On Host#2, replace the following lines shown with lines in next steps.

Add following lines below it to add peer2 to the channel

Host#2

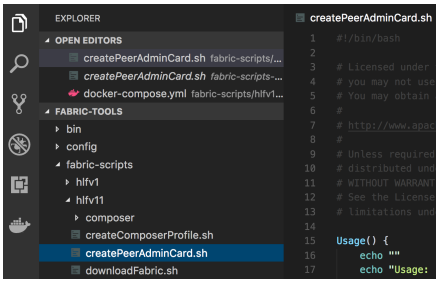
```
# # Fetch channel block from orderer
docker exec -e
"CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin
@org1.example.com/msp" peer2.org2.example.com peer
channel fetch config -o orderer.example.com:7050 -c
composerchannel

# # Join peer1.org1.example.com to the channel.
docker exec -e
"CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin
@org1.example.com/msp" peer1.org1.example.com peer
channel join -b composerchannel_config.block
```

Task 4 is complete!

TASK#5: Peer Admin Card creation

Host#1

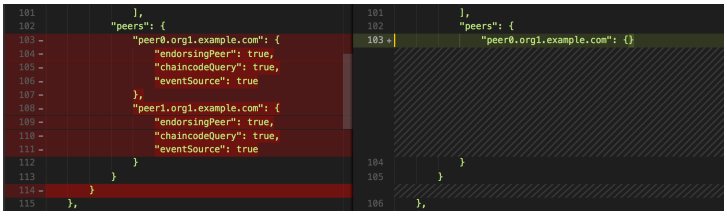
Step 1: Using Visual Studio Code Open the "fabric-tools" folder and open the "createPeerAdminCard.sh" of hlfv11 for editing



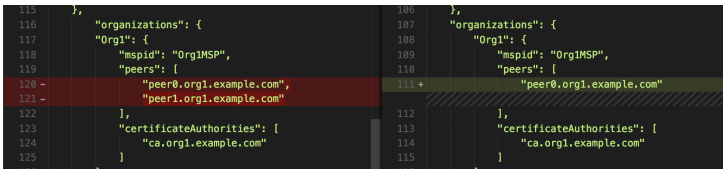
Host#1

Step 2: Update the connection string to add **peer1** as shown in the images below:

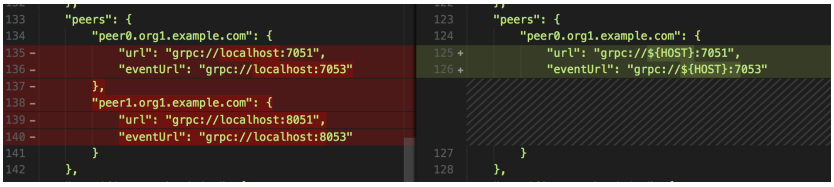
#1:



#2:



#3:

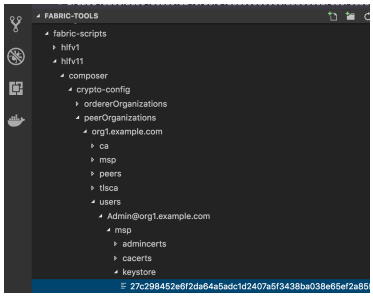


Host#1

Step 3: Update the PRIVATE KEY for the MSP as we did in TASK#3 Step-2.

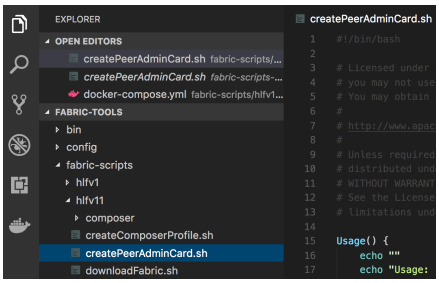
LOCATION for Key:

crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore/



Host#2

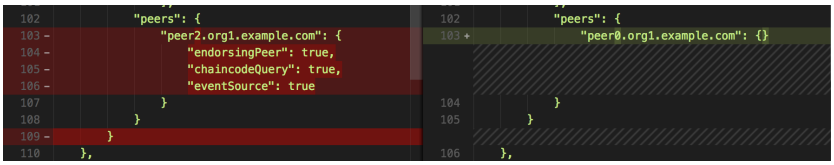
Step 4: We need to repeat similar steps on Host2 for Peer2. Using Visual Studio Code Open the "fabric-tools" folder and open the "createPeerAdminCard.sh" of hlfv11 for editing



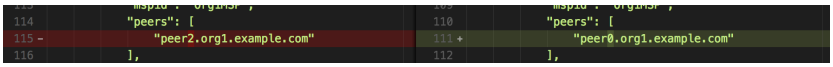
Host#2

Step 5: Update the connection string to remove **peer0** & **peer1** and instead add **peer2** as shown in the images below:

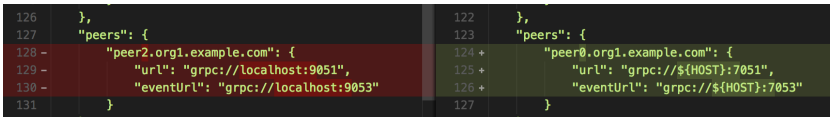
#1:



#2:



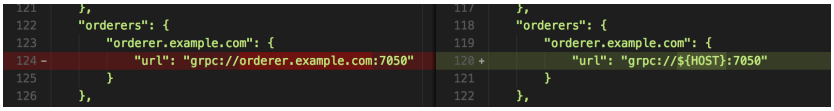
#3:



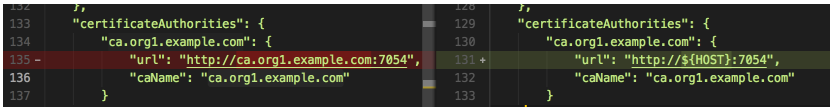
Host#2

Step 6: Orderer & CA in the script is assumes to be on the same localhost. We need to explicitly call it by its name to be identified on the docker swarm network.

For Orderer use **orderer.example.com** instead of ${HOST}



Similiarly use **ca.org1.example.com** instead of ${HOST}

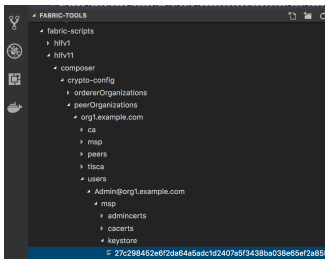


Host#2

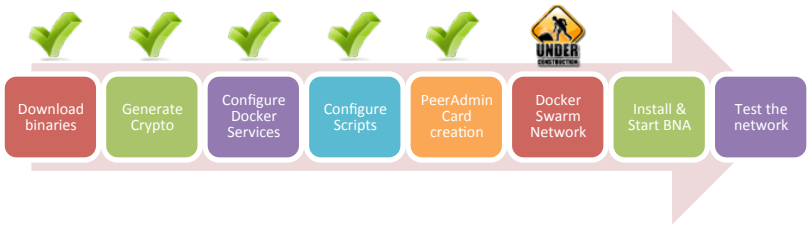
Step 7: Update the PRIVATE KEY for the MSP as we did in Step-3.

LOCATION for Key:

crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore/



Task 5 is complete!



TASK#6: Create Docker Swarm network

**Please ensure docker service is running on both the HOSTs

Host#1 & HOST#2

Step 1: Open the terminal window and CD into the Fabric's root folder directory

```
cd $HOME/fabric-tools
```

Host#1 & HOST#2

Step 2: Shutdown all docker containers using the following command

```
./teardownAllDocker.sh
```

```
For all Docker containers or images (not just Hyperledger Fabric and Composer)
1 - Kill and remove only the containers
2 - Kill and remove the containers and remove all the downloaded images
3 - Quit and not do anything

1) Kill & Remove
2) Remove Images
3) Quit
Please select which option > 1
```

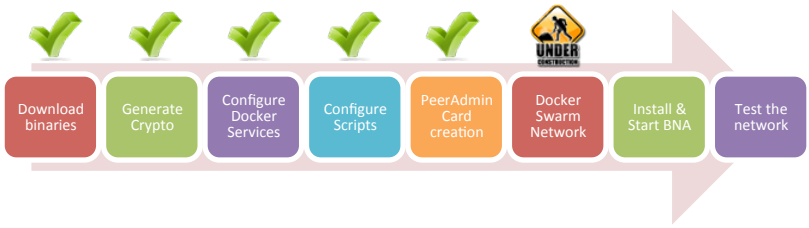
Choose **option 1** and press enter

Host#1

Step 3: Run the following command to start a docker swarm

** Please replace the text in blue below by the IP of Host#1

```
docker swarm init --advertise-addr 192.168.43.5
```




```
Swarm initialized: current node (q53aw3l90uvynkiq8npjml2ol) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3q7mcx6q9i9ri02xpquwwbt3wynews2akkyr18zbvwtuub0yhs-4zkz4ejzgi82wso2t
b4q0h3ny 192.168.56.102:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Host#1

Step 3: We need the other host HOST#2 to join as a manager hence run the following command on HOST#1

```
docker swarm join-token manager
```

On running above command you will get a string that you need to run on HOST#2

```
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3q7mcx6q9i9ri02xpquwwbt3wynews2akkyr18zbvwtuub0yhs-9fsrqjrtiq0xu5yly
b4ybjgea 192.168.56.102:2377
```

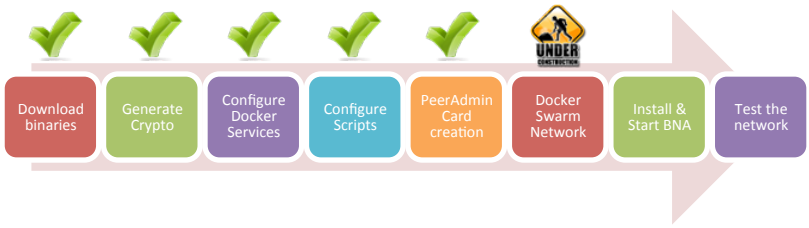
Host#2

Step 4: Use the highlighted command provided as a result of step3 on HOST#2 to join the docker swarm as a manager

```
docker swarm join --token SWMTKN-1-
2hsbwmhd3gxmrzy2o2rpheb3rwukzninrvk1i041z4333okioe-
73tsd5ig3tik0gbstn3qydk5e 192.168.56.102:2377
```

Host#2

Step 5: Use the highlighted command provided as a result of step3 on HOST#2 to join the docker swarm as a manager




```
mustafa@mustafa:~/Desktop/Chapter 5/script$ docker swarm join --token SWMTKN-1-3q7mcx6q9l9ri02xpquwwbt
3wynews2akkyr18zbvwtuub0yhs-9fsrqjrtiq0xu5ylyb4ybjgea 192.168.56.102:2377
This node joined a swarm as a manager.
```

Host#1

Step 6: Now we need to create an Overlay Network. Use the following command to generate the overlay;

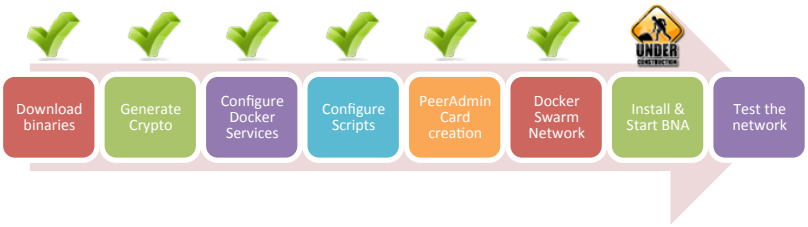
```
docker network create --attachable --driver overlay
composer_hyp-net
```

Host#1 & Host#2

Step 7: Validate that the overlay is created. Use following command to verify;

```
mustafa@mustafa:~/Desktop/Chapter 5/script$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
49d4c61d2ae6        bridge              bridge              local
740a392479c1        composer_default    bridge              local
f6c4m9lb63yu        composer_hyp-net    overlay             swarm
e7a6462034fe        docker_gwbridge     bridge              local
caebf14b9a4f        host                host                local
d4qpx2bwga63        ingress             overlay             swarm
34fa92552941        none                null                local
```

Task 6 is complete!



TASK#7: Deploy and Launch the Network

Host#1 & Host#2
Step 1: Unzip "chapter05.zip" code and CD into the "chapter05" directory

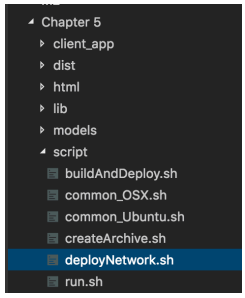
Host#1 & Host#2
Step 2: Run NPM install to install all required dependencies

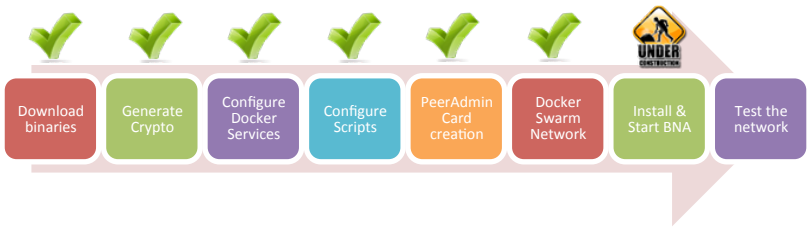
```
npm install
```

Host#1 & Host#2
Step 3: CD in the scripts directory

```
cd script
```

Host#1 & Host#2
Step 4: Using Visual Studio Code Open the "chapter05/script" folder and open the "deployNetwork.sh" file for editing





Host#1

Step 5: Along with this chapter two files "deployNetwork.1.sh" and "deployNetwork.2.sh" is provided in "**chapter5-script**" folder.

On Host#1 use the content of "deployNetwork.1.sh" and paste in the existing "deployNetwork.sh" file.

Host#2

Step 6: On Host#2 use the content of "deployNetwork.2.sh" and paste in the existing "deployNetwork.sh" file.

Host#1

Step 7: Run buildAndDeploy Script inside Chapter5/script folder

```
./buildAndDeploy.sh
```

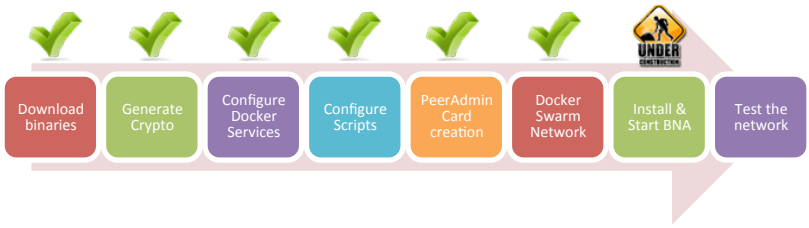
The script will run the "startFabric.sh" and the "createPeerAdminCard.sh" files we created in the above tasks.

Let's understand what happened with this script…

- ✓ Rather than creating a new network bridge the docker has now utilized our overlay we recently created

```
FABRIC_START_TIMEOUT is unset, assuming 10 (seconds)
Network composer_hyp-net is external, skipping
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 sch
eduled on the current node.
```

- ✓ Peer0 and Peer1 docker container is now created along with couchdb and couchdb1 containers




```
Creating orderer.example.com ...
Creating couchdb1 ...
Creating couchdb ...
Creating ca.org1.example.com ...
Creating orderer.example.com
Creating couchdb1
Creating couchdb
Creating couchdb1 ... done
Creating peer0.org1.example.com ...
Creating peer1.org1.example.com ...
Creating peer0.org1.example.com
Creating peer0.org1.example.com ... done
```

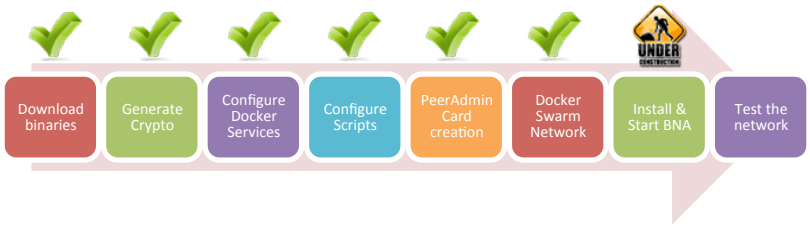
- ✓ 'composer_channel' is created

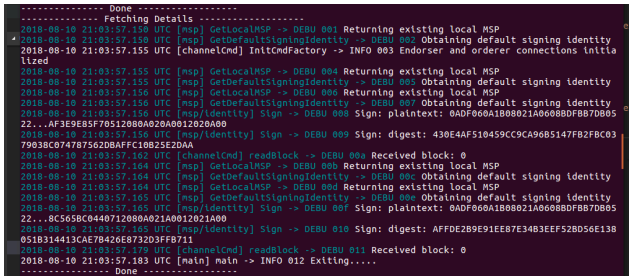
```
2018-08-10 21:03:55.653 UTC [msp] GetLocalMSP -> DEBU 01a Returning existing local MSP
2018-08-10 21:03:55.653 UTC [msp] GetDefaultSigningIdentity -> DEBU 01b Obtaining default signing identity
2018-08-10 21:03:55.653 UTC [msp] GetLocalMSP -> DEBU 01c Returning existing local MSP
2018-08-10 21:03:55.653 UTC [msp] GetDefaultSigningIdentity -> DEBU 01d Obtaining default signing identity
2018-08-10 21:03:55.654 UTC [msp/identity] Sign -> DEBU 01e Sign: plaintext: 0ADF060A1B08021A0608BBFBB7DB05
22...A273CEF6105612080A021A0012021A00
2018-08-10 21:03:55.654 UTC [msp/identity] Sign -> DEBU 01f Sign: digest: CC27347BC88AFB4FC1E594F2514641DC6
7B2D77B941AE10B72B0C370B49B091D
2018-08-10 21:03:55.659 UTC [channelCmd] readBlock -> DEBU 020 Received block: 0
2018-08-10 21:03:55.660 UTC [main] main -> INFO 021 Exiting.....
--------------- Done ----------------
```

- ✓ Peer0 joined channel

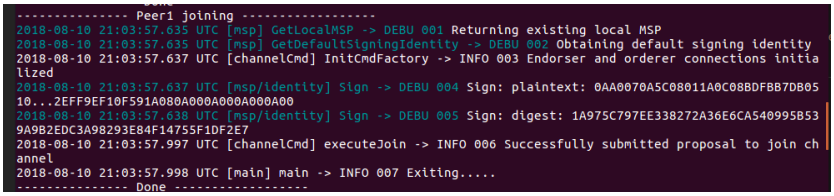
```
--------------- Done ----------------
------------- Peer0 Joining ----------------
2018-08-10 21:03:56.387 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2018-08-10 21:03:56.387 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2018-08-10 21:03:56.389 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initia
lized
2018-08-10 21:03:56.390 UTC [msp/identity] Sign -> DEBU 004 Sign: plaintext: 0AA0070A5C08011A0C08BCFBB7DB05
10...2EFF9EF10F591A080A000A000A000A00
2018-08-10 21:03:56.400 UTC [msp/identity] Sign -> DEBU 005 Sign: digest: DAEEFE616361A220E517B318FB036CD27
BB205B35F6CF888792158C156866E7F
2018-08-10 21:03:56.830 UTC [channelCmd] executeJoin -> INFO 006 Successfully submitted proposal to join ch
annel
2018-08-10 21:03:56.830 UTC [main] main -> INFO 007 Exiting.....
--------------- Done ----------------
```

- ✓ Now as the channel is not empty, before Peer1 joining the channel we need to fetch existing channel blocks

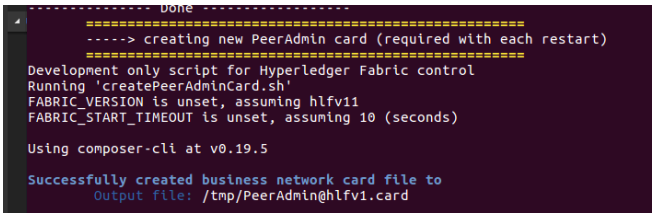


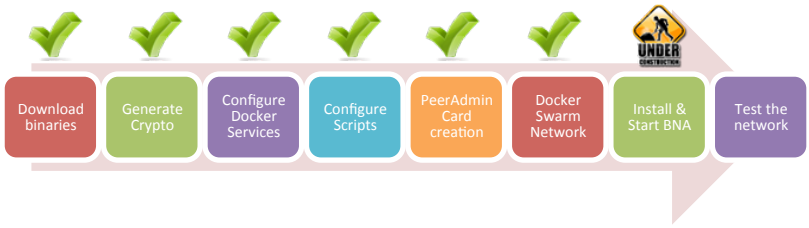


✓ Peer1 joined the channel



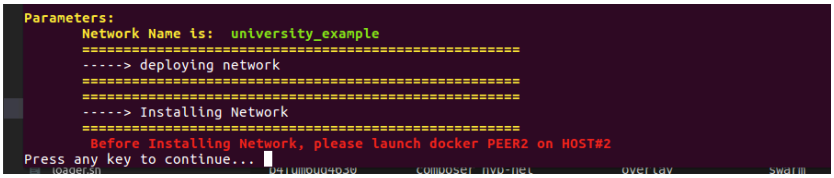
✓ New peerAdminCard is created & imported into the system





Host#1

Step 8: On Host#1 the script will halt. Before installing the network we need to launch the peer2 on host#2



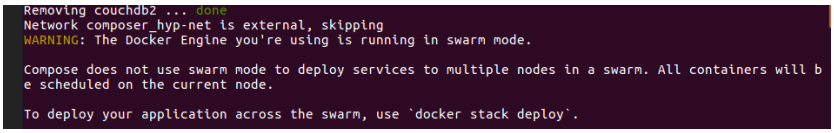
Host#2

Step 9: on Host#2 run buildAndDeploy Script inside Chapter5/script folder

```
./buildAndDeploy.sh
```

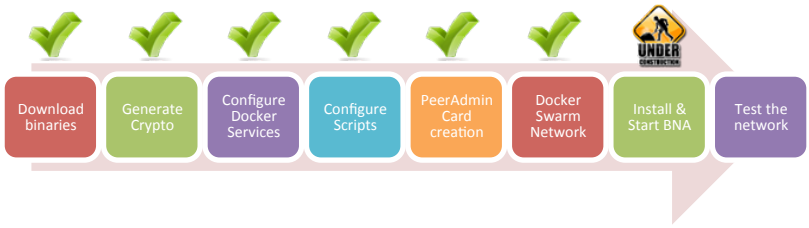
The script will again run the "startFabric.sh" and the "createPeerAdminCard.sh" files for Host#2.

Host#2 should also now utilize the same OVELAY network '***composer_hyp-net***'



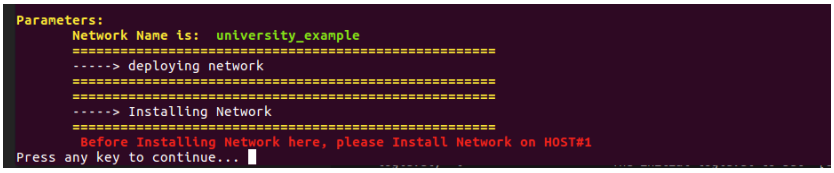
This script will do the following:

- ✓ Fetch exiting channel blocks
- ✓ Enroll and join Peer2 to the channel
- ✓ Create Peer Admin card for the Host#2



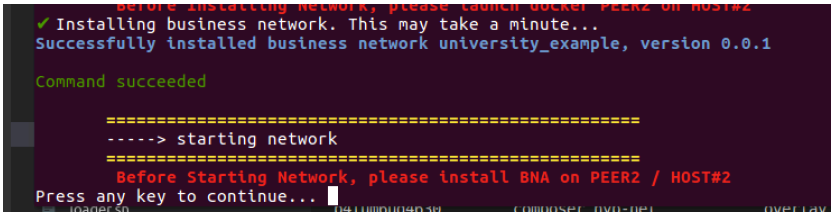
Host#2

Step 10: On Host#2 the script will halt. Before installing the network on Host#2 we install it on Host#1



Host#1

Step 11: On Host#1 press any key to continue installing the network.

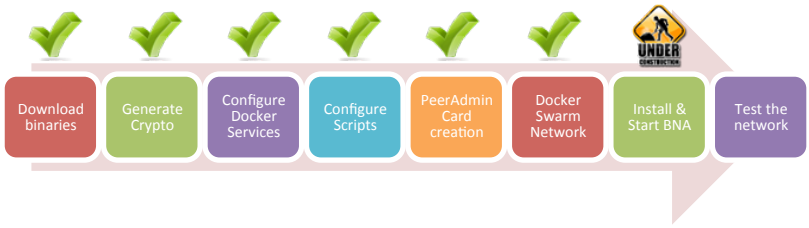


Host#2

Step 12: Now on Host#2 install the network by continuing the script execution. Press any key.

Host#1

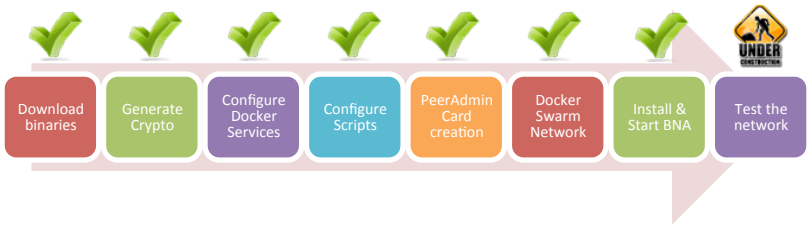
Step 13: Finally we will start the Business Network by continuing the execution of script.




```
Command succeeded

      ======================================================
      -----> pinging admin@university_example card
      ======================================================
The connection to the network was successfully tested: university_example
       Business network version: 0.0.1
       Composer runtime version: 0.19.5
       participant: org.hyperledger.composer.system.NetworkAdmin#admin
       identity: org.hyperledger.composer.system.Identity#b4752fd38625f8458ec6f690e0d7d45e3c11c206c0cba368
e72f575b347af69d

Command succeeded
```

Now we are all done with deployment and starting of business network.

Task 7 is complete!



TASK#8: Testing the network

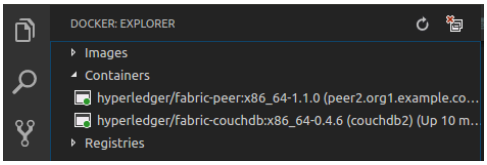
TASK#8.1: Visual Testing docker containers

Host#1

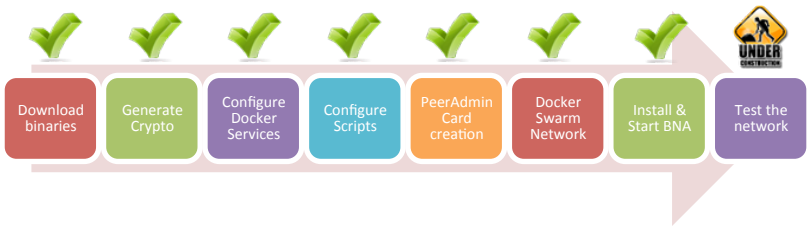
Step 1: On Host#1 launch docker-explorer installed on visual studio code.

Host#2

Step 2: On Host#2 launch docker-explorer installed on visual studio code.



Task 8.1 is complete!



TASK#8.2: couchDB Validation

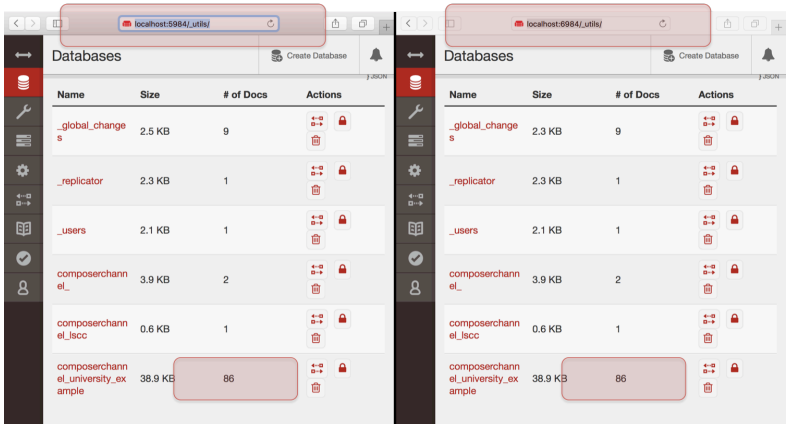
Host#1

Step 1: On Host#1 launch Internet explorer and go to the following URL

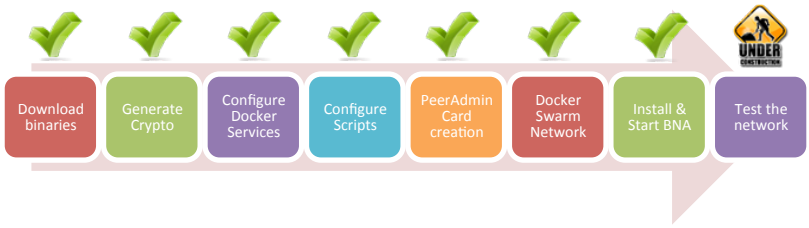
Open couchDB database admin/util page using separate browsers:

For Peer0 : couchDB
URL: localhost:5984/_utils/

For Peer1 : couchDB1
URL: localhost:6984/_utils/



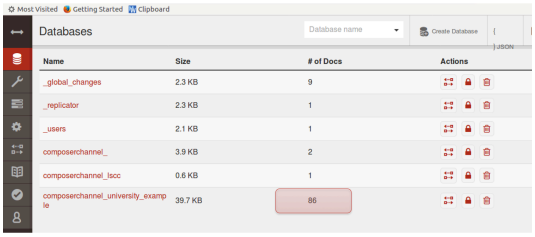
Please Note: Note that both the blocks are in sync as shown highlighted



Host#2

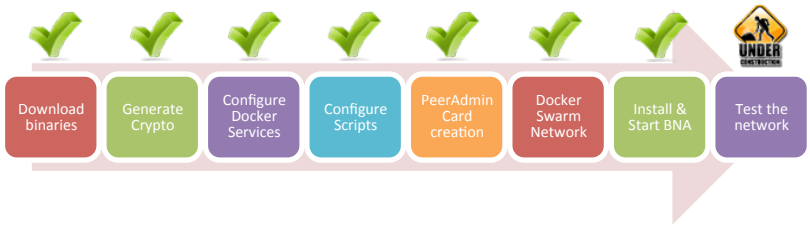
Step 2: On Host#2 open couchDB database admin/util page using separate browsers:

For Peer2 : couchDB
URL: localhost:7984/_utils/



Please Note: Note that both the blocks are in sync as shown highlighted

Task 8.2 is complete!



TASK#8.3: Run UI – Make Transaction & Validate Blocks

Student are encouraged to complete this validation on their own. Few minimal pointers shall be provided which are essential.

Host#1

Step 1: Post deploying the "chapter5" University usecase run the UI using the following command from the 'script' folder

```
./run.sh
```

Step 2: Open couchDB database admin/util page using separate browsers:

Host#1

For Peer0 : couchDB

URL: localhost:5984/_utils/

Host#1

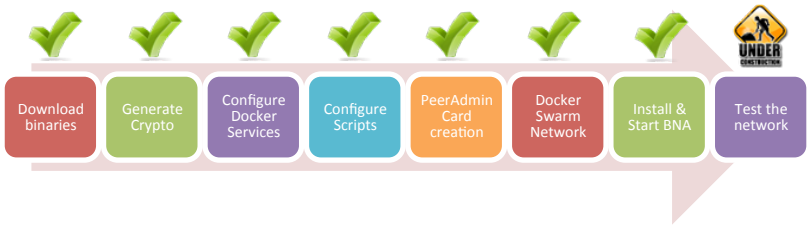
For Peer1 : couchDB1

URL: localhost:6984/_utils/

Host#2

For Peer2: couchDB2

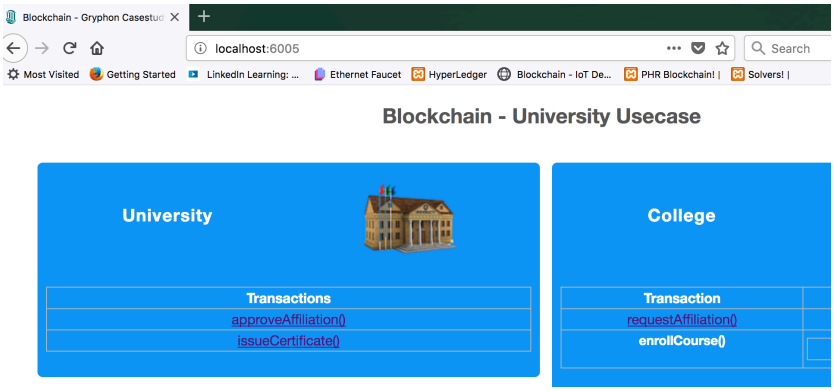
URL: localhost:7984/_utils/



Step 3: Open UI using the URL as shown up after running ./run.sh script in step-1

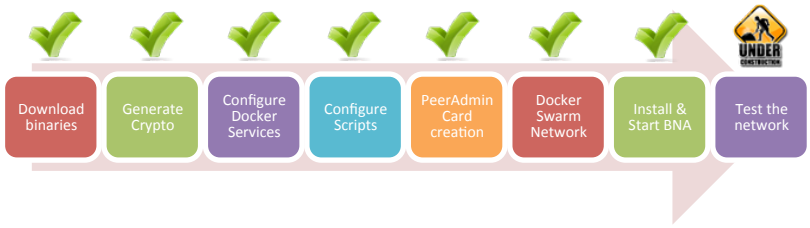
```
Command succeeded

Mustafas-MBP:script mustafahusain$ ./run.sh
Listening locally on port 6005
```

URL: localhost:6005

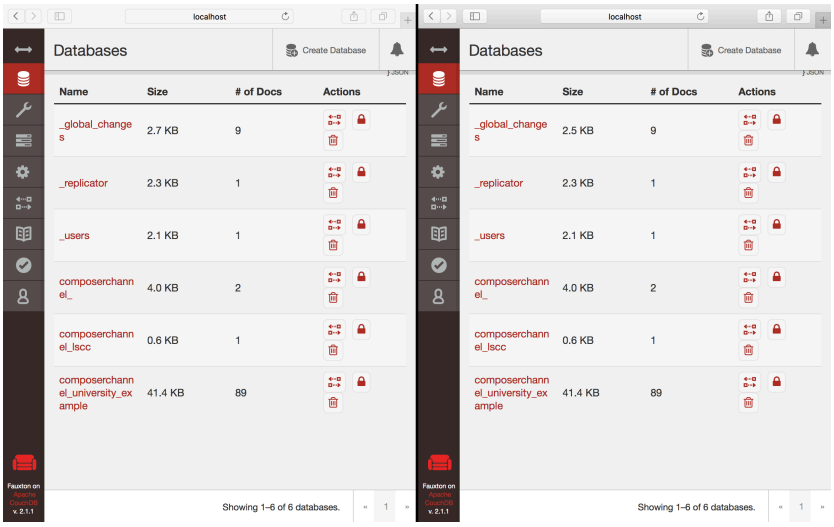


Step 4: Run requestAffiliation() Transaction

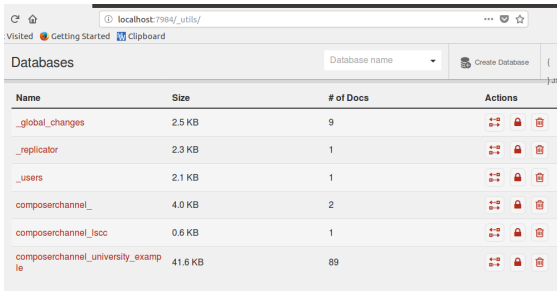
Step 5: Refresh all DB browsers on both hosts and check that the blocks are updated



Host#1



Host#2



Students are further encouraged to explore more test scenarios to learn more.

Task 8.3 is complete!

SUMMARY

In this chapter we

- Created a Docker Swarm Network and an overlay to connect two hosts, both running docker services
- Connected both Hosts as a Manager Node
- Passed and environment variable CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE to match the docker swarm overlay network we created
- Deployed & launched peers and BNA on both hosts
- Finally did a quick test to validate the working