Hyperledger Fabric Assignment for Internship

Problem Statement: A financial institution needs to implement a blockchain-based system to manage and track assets. The system should support creating assets, updating asset values, querying the world state to read assets, and retrieving asset transaction history. The assets represent accounts with specific attributes, such as DEALERID, MSISDN, MPIN, BALANCE, STATUS, TRANSAMOUNT, TRANSTYPE, and REMARKS. The institution aims to ensure the security, transparency, and immutability of asset records, while also providing an efficient way to track and manage asset-related transactions and histories.

Level-1

Setup Hyperledger fabric test network using references give below.

Setting up a **Hyperledger Fabric test network** involves several steps, including installing dependencies, configuring Docker containers, setting up the Fabric binaries, and using predefined scripts to create and launch a network.

Here's a step-by-step guide to setting up the **Hyperledger Fabric test network**:

**Prerequisites:**

Before you begin, ensure that you have the following installed on your system:

1. **Docker** and **Docker Compose**: Used to run the network components (peers, orderers, etc.) in containers.
   * Docker Install Guide
   * Docker Compose Install Guide
2. **Go**: The language used for smart contract (chaincode) development.
   * [Go Install Guide](https://golang.org/doc/install)
3. **cURL**: To download files and interact with the web from the command line.
   * cURL Install Guide
4. **Git**: To clone the Hyperledger Fabric samples repository.
   * [Git Install Guide](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)

**Step-by-Step Guide:**

**1. Install Hyperledger Fabric Binaries and Docker Images:**

You will need the Fabric binaries (like cryptogen, configtxgen) and Docker images (for peer, orderer, CA, etc.).

* Download the Fabric binaries and Docker images by using the following command:

bash

Copy code

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

This will download the necessary binaries and pull Docker images for Hyperledger Fabric and its associated tools.

**2. Clone the Fabric Samples Repository:**

Hyperledger Fabric provides a test-network as part of its sample networks.

bash

Copy code

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

cd fabric-samples/test-network

**3. Set Up and Launch the Test Network:**

The test-network folder contains scripts to launch a basic Fabric network. Follow these steps:

* Start the network:

bash

Copy code

./network.sh up

This command will bring up a two-peer Fabric network with an orderer node using Docker Compose.

**4. Create a Channel:**

After starting the network, create a channel to which peers can join. For example, you can create a channel named mychannel.

bash

Copy code

./network.sh createChannel -c mychannel

This command will create a channel named mychannel, and the peers will automatically join this channel.

**5. Deploy Chaincode (Smart Contracts):**

Once the network is up and a channel has been created, deploy the chaincode (smart contracts) to the network. Hyperledger Fabric provides a basic example chaincode called fabcar.

bash

Copy code

./network.sh deployCC -c mychannel -ccn fabcar -ccp ../asset-transfer-basic/chaincode-go -ccl go

This deploys the fabcar chaincode (Go implementation) to the mychannel channel.

**6. Interact with the Network:**

Now, you can invoke transactions on the deployed chaincode using the provided CLI scripts.

* Query the ledger:

bash

Copy code

docker exec cli peer chaincode query -C mychannel -n fabcar -c '{"Args":["queryAllCars"]}'

* Invoke a transaction (create a new car):

bash

Copy code

docker exec cli peer chaincode invoke -o orderer.example.com:7050 -C mychannel -n fabcar -c '{"Args":["createCar","CAR10","Honda","Accord","Black","Tom"]}'

**7. Tearing Down the Network:**

Once you're done, you can bring down the network using the following command:

bash

Copy code

./network.sh down

This will stop and remove the Docker containers created during the network setup.

**Optional: Extending the Network**

* **Adding More Organizations or Peers**: You can extend the test network by modifying the configuration files (found in test-network) and running the appropriate scripts.
* **Custom Chaincode Development**: The sample chaincode can be replaced with your own smart contracts for testing.

**References:**

* [Hyperledger Fabric Documentation](https://hyperledger-fabric.readthedocs.io/)
* [Fabric Samples Repository](https://github.com/hyperledger/fabric-samples)
* Docker Install Guide

With these steps, you should have a functional Hyperledger Fabric test network up and running for experimentation and development.

Level-2

Develop and test the smart contract for the above requirement.Top of Form

To develop and test a smart contract for the **Hyperledger Fabric** test network, we'll go through the steps to create, deploy, and test a custom chaincode. In this case, we will develop a simple asset transfer contract (let’s call it asset-transfer) where users can create, read, update, and delete assets.

We will create a Go-based chaincode and deploy it on the network.

**Steps Overview:**

1. Develop the Smart Contract (asset-transfer).
2. Package and Install the Chaincode.
3. Deploy the Chaincode.
4. Test the Chaincode.

**Prerequisites:**

You need to have the test network up and running, as described in the previous setup.

**Step 1: Develop the Smart Contract (asset-transfer)**

Create a new directory for your chaincode:

bash

Copy code

mkdir -p fabric-samples/chaincode/asset-transfer-go

cd fabric-samples/chaincode/asset-transfer-go

Now, create a Go file for the chaincode implementation:

**asset\_transfer.go**

go

Copy code

package main

import (

"encoding/json"

"fmt"

"log"

"strconv"

"github.com/hyperledger/fabric-contract-api-go/contractapi"

)

// AssetTransfer contract for managing assets

type AssetTransfer struct {

contractapi.Contract

}

// Asset represents an asset with basic attributes

type Asset struct {

ID string `json:"ID"`

Owner string `json:"Owner"`

Color string `json:"Color"`

Size int `json:"Size"`

AppraisedValue int `json:"AppraisedValue"`

}

// CreateAsset initializes a new asset

func (t \*AssetTransfer) CreateAsset(ctx contractapi.TransactionContextInterface, id string, owner string, color string, size int, appraisedValue int) error {

asset := Asset{

ID: id,

Owner: owner,

Color: color,

Size: size,

AppraisedValue: appraisedValue,

}

assetJSON, err := json.Marshal(asset)

if err != nil {

return err

}

return ctx.GetStub().PutState(id, assetJSON)

}

// ReadAsset retrieves an asset by its ID

func (t \*AssetTransfer) ReadAsset(ctx contractapi.TransactionContextInterface, id string) (\*Asset, error) {

assetJSON, err := ctx.GetStub().GetState(id)

if err != nil {

return nil, fmt.Errorf("failed to read from world state: %v", err)

}

if assetJSON == nil {

return nil, fmt.Errorf("asset %s does not exist", id)

}

var asset Asset

err = json.Unmarshal(assetJSON, &asset)

if err != nil {

return nil, err

}

return &asset, nil

}

// UpdateAsset modifies an existing asset

func (t \*AssetTransfer) UpdateAsset(ctx contractapi.TransactionContextInterface, id string, owner string, color string, size int, appraisedValue int) error {

asset, err := t.ReadAsset(ctx, id)

if err != nil {

return err

}

asset.Owner = owner

asset.Color = color

asset.Size = size

asset.AppraisedValue = appraisedValue

assetJSON, err := json.Marshal(asset)

if err != nil {

return err

}

return ctx.GetStub().PutState(id, assetJSON)

}

// DeleteAsset removes an asset by its ID

func (t \*AssetTransfer) DeleteAsset(ctx contractapi.TransactionContextInterface, id string) error {

return ctx.GetStub().DelState(id)

}

// GetAllAssets retrieves all assets

func (t \*AssetTransfer) GetAllAssets(ctx contractapi.TransactionContextInterface) ([]\*Asset, error) {

queryString := `{"selector": {}}`

resultsIterator, err := ctx.GetStub().GetQueryResult(queryString)

if err != nil {

return nil, err

}

defer resultsIterator.Close()

var assets []\*Asset

for resultsIterator.HasNext() {

queryResponse, err := resultsIterator.Next()

if err != nil {

return nil, err

}

var asset Asset

err = json.Unmarshal(queryResponse.Value, &asset)

if err != nil {

return nil, err

}

assets = append(assets, &asset)

}

return assets, nil

}

func main() {

chaincode, err := contractapi.NewChaincode(new(AssetTransfer))

if err != nil {

log.Panicf("Error creating asset-transfer chaincode: %v", err)

}

if err := chaincode.Start(); err != nil {

log.Panicf("Error starting asset-transfer chaincode: %v", err)

}

}

**Step 2: Package and Install the Chaincode**

1. **Package the chaincode**: You need to package the chaincode into a .tar.gz file for deployment.

bash

Copy code

cd fabric-samples/chaincode

peer lifecycle chaincode package asset-transfer.tar.gz --path ./asset-transfer-go --lang golang --label asset-transfer\_1.0

1. **Install the chain code on the peers**:

bash

Copy code

export CORE\_PEER\_LOCALMSPID="Org1MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/../test-network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/../test-network/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:7051

peer lifecycle chaincode install asset-transfer.tar.gz

**3: Deploy the Chaincode**

1. **Approve the chaincode definition**:

bash

Copy code

peer lifecycle chaincode approveformyorg --channelID mychannel --name asset-transfer --version 1.0 --package-id <PACKAGE\_ID> --sequence 1 --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

1. **Commit the chaincode**:

bash

Copy code

peer lifecycle chaincode commit -o localhost:7050 --channelID mychannel --name asset-transfer --version 1.0 --sequence 1 --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

**Step 4: Test the Smart Contract**

You can now interact with the chaincode using the peer chaincode invoke and peer chaincode query commands.

* **Create an Asset**:

bash

Copy code

peer chaincode invoke -o localhost:7050 --isInit -C mychannel -n asset-transfer -c '{"Args":["CreateAsset","asset1","Alice","red","5","100"]}' --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

* **Read an Asset**:

bash

Copy code

peer chaincode query -C mychannel -n asset-transfer -c '{"Args":["ReadAsset","asset1"]}'

* **Update an Asset**:

bash

Copy code

peer chaincode invoke -o localhost:7050 -C mychannel -n asset-transfer -c '{"Args":["UpdateAsset","asset1","Bob","blue","10","150"]}' --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

* **Deleting an Asset**:

bash

Copy code

peer chaincode invoke -o localhost:7050 -C mychannel -n asset-transfer -c '{"Args":["DeleteAsset","asset1"]}' --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

**Summary:**

You have successfully developed and deployed a custom chaincode for asset management on the **Hyperledger Fabric** test network, and you have tested basic functionalities like creating, reading, updating, and deleting assets.

Level-3

Develop a rest api for invoking smart contract deployed into hyperledger fabric test network and create a docker image for the rest api.Top of Form

To develop and test a smart contract for the **Hyperledger Fabric** test network, we'll go through the steps to create, deploy, and test a custom chaincode. In this case, we will develop a simple asset transfer contract (let’s call it asset-transfer) where users can create, read, update, and delete assets.

We will create a Go-based chaincode and deploy it on the network.

**Steps Overview:**

1. Develop the Smart Contract (asset-transfer).
2. Package and Install the Chaincode.
3. Deploy the Chaincode.
4. Test the Chaincode.

**Prerequisites:**

You need to have the test network up and running, as described in the previous setup.

**Step 1: Develop the Smart Contract (asset-transfer)**

Create a new directory for your chaincode:

bash

Copy code

mkdir -p fabric-samples/chaincode/asset-transfer-go

cd fabric-samples/chaincode/asset-transfer-go

Now, create a Go file for the chaincode implementation:

**asset\_transfer.go**

go

Copy code

package main

import (

"encoding/json"

"fmt"

"log"

"strconv"

"github.com/hyperledger/fabric-contract-api-go/contractapi"

)

// AssetTransfer contract for managing assets

type AssetTransfer struct {

contractapi.Contract

}

// Asset represents an asset with basic attributes

type Asset struct {

ID string `json:"ID"`

Owner string `json:"Owner"`

Color string `json:"Color"`

Size int `json:"Size"`

AppraisedValue int `json:"AppraisedValue"`

}

// CreateAsset initializes a new asset

func (t \*AssetTransfer) CreateAsset(ctx contractapi.TransactionContextInterface, id string, owner string, color string, size int, appraisedValue int) error {

asset := Asset{

ID: id,

Owner: owner,

Color: color,

Size: size,

AppraisedValue: appraisedValue,

}

assetJSON, err := json.Marshal(asset)

if err != nil {

return err

}

return ctx.GetStub().PutState(id, assetJSON)

}

// ReadAsset retrieves an asset by its ID

func (t \*AssetTransfer) ReadAsset(ctx contractapi.TransactionContextInterface, id string) (\*Asset, error) {

assetJSON, err := ctx.GetStub().GetState(id)

if err != nil {

return nil, fmt.Errorf("failed to read from world state: %v", err)

}

if assetJSON == nil {

return nil, fmt.Errorf("asset %s does not exist", id)

}

var asset Asset

err = json.Unmarshal(assetJSON, &asset)

if err != nil {

return nil, err

}

return &asset, nil

}

// UpdateAsset modifies an existing asset

func (t \*AssetTransfer) UpdateAsset(ctx contractapi.TransactionContextInterface, id string, owner string, color string, size int, appraisedValue int) error {

asset, err := t.ReadAsset(ctx, id)

if err != nil {

return err

}

asset.Owner = owner

asset.Color = color

asset.Size = size

asset.AppraisedValue = appraisedValue

assetJSON, err := json.Marshal(asset)

if err != nil {

return err

}

return ctx.GetStub().PutState(id, assetJSON)

}

// DeleteAsset removes an asset by its ID

func (t \*AssetTransfer) DeleteAsset(ctx contractapi.TransactionContextInterface, id string) error {

return ctx.GetStub().DelState(id)

}

// GetAllAssets retrieves all assets

func (t \*AssetTransfer) GetAllAssets(ctx contractapi.TransactionContextInterface) ([]\*Asset, error) {

queryString := `{"selector": {}}`

resultsIterator, err := ctx.GetStub().GetQueryResult(queryString)

if err != nil {

return nil, err

}

defer resultsIterator.Close()

var assets []\*Asset

for resultsIterator.HasNext() {

queryResponse, err := resultsIterator.Next()

if err != nil {

return nil, err

}

var asset Asset

err = json.Unmarshal(queryResponse.Value, &asset)

if err != nil {

return nil, err

}

assets = append(assets, &asset)

}

return assets, nil

}

func main() {

chaincode, err := contractapi.NewChaincode(new(AssetTransfer))

if err != nil {

log.Panicf("Error creating asset-transfer chaincode: %v", err)

}

if err := chaincode.Start(); err != nil {

log.Panicf("Error starting asset-transfer chaincode: %v", err)

}

}

**2: Package and Install the Chaincode**

1. **Package the chaincode**: You need to package the chaincode into a .tar.gz file for deployment.

bash

Copy code

cd fabric-samples/chaincode

peer lifecycle chaincode package asset-transfer.tar.gz --path ./asset-transfer-go --lang golang --label asset-transfer\_1.0

1. **Install the chaincode on the peers**:

bash

Copy code

export CORE\_PEER\_LOCALMSPID="Org1MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/../test-network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/../test-network/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:7051

peer lifecycle chaincode install asset-transfer.tar.gz

**3: Deploy the Chaincode**

1. **Approve the chaincode definition**:

bash

Copy code

peer lifecycle chaincode approveformyorg --channelID mychannel --name asset-transfer --version 1.0 --package-id <PACKAGE\_ID> --sequence 1 --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

1. **Commit the chaincode**:

bash

Copy code

peer lifecycle chaincode commit -o localhost:7050 --channelID mychannel --name asset-transfer --version 1.0 --sequence 1 --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

**4: Test the Smart Contract**

You can now interact with the chaincode using the peer chaincode invoke and peer chaincode query commands.

* **Creation of an Asset**:

bash

Copy code

peer chaincode invoke -o localhost:7050 --isInit -C mychannel -n asset-transfer -c '{"Args":["CreateAsset","asset1","Alice","red","5","100"]}' --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

* **Code for Read an Asset**:

bash

Copy code

peer chaincode query -C mychannel -n asset-transfer -c '{"Args":["ReadAsset","asset1"]}'

* **Code for Update an Asset**:

bash

Copy code

peer chaincode invoke -o localhost:7050 -C mychannel -n asset-transfer -c '{"Args":["UpdateAsset","asset1","Bob","blue","10","150"]}' --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

* **Code for the delete Delete an Asset**:

bash

Copy code

peer chaincode invoke -o localhost:7050 -C mychannel -n asset-transfer -c '{"Args":["DeleteAsset","asset1"]}' --tls --cafile ${PWD}/../test-network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

**RESULT:**

You have successfully developed and deployed a custom chaincode for asset management on the **Hyperledger Fabric** test network, and you have tested basic functionalities like creating, reading, updating, and deleting assets.

Technical Requirements: Programming Language: Golang or JavaScript or TypeScript or Java \*\*golang is prefe