Ira A. Fulton Schools of Engineering
Arizona State University

CSE 598: Engineering Blockchain Applications

Hyperledger Fabric Private Blockchain Chaincode Development Project

Purpose

Welcome to the Hyperledger Fabric Smart Contract Development Project! In this project, you will gain hands-on experience in developing, deploying, and testing a smart contract on the Hyperledger Fabric platform. You will be given a skeleton 'Supply Chain Management' Go file containing an initLedger method and other empty methods. You will need to implement these methods based on the provided tasks and objectives.

Objectives

Learners will be able to:

- Describe the basic blockchain data structure
- Identify an industry that can be improved with the benefits of blockchain
- Describe applications of blockchain technology
- Analyze the viability of a current blockchain application
- Deploy a blockchain application using Hyperledger Fabric.
- Gain hands-on experience in deploying and testing a ‘Supply Chain Management’ smart contract.

Technology Requirements

This project requires the following tools and technologies:

- Hyperledger Fabric and its architecture
- Docker and Docker Compose
- Go: Programming language
- Git

Project Description

Project Methods Descriptions

Here are the descriptions of what each method in the SupplyChainContract is intended for:

1. InitLedger

   This method initializes the ledger with some example products.

2. CreateProduct

   This method creates a new product in the ledger.

3. UpdateProduct

   This method allows updating a product's status, owner, description, and category.

4. TransferOwnership

   This method changes the owner of a product. You need to retrieve the existing product from the ledger using its ID, update the Owner field with the new owner.

5. QueryProduct

   This method retrieves a single product from the ledger by ID.

6. GetAllProducts (Helper Method)

   This method retrieves all products from the ledger.

7. putProduct (Helper Method)

   This helper method is used for inserting or updating a product in the ledger.

8. ProductExists (Helper Method)

   This helper method checks if a product exists in the ledger.

Directions

Setting Up Project

Installing Prerequisites

- Install Go programming language from the Go official website.
- Install Git from its official website.

Setup

Project Skeleton Source Code: CSE598-EBA-Project2

- Clone the project repository to your local machine using the following command:
  git clone https://github.com/pavankramadugu/CSE598-EBA-Project2.git

- Navigate to the Project Directory:
  Change to the project directory: *cd CSE598-EBA-Project2*

- Install Dependencies:
  Install the required Go dependencies for the project: *go mod tidy*

Project Tasks

In the smartcontract.go file, you will find several methods with empty bodies. Your task is to implement these methods according to the specifications provided.

The methods you need to implement include:
- CreateProduct
- UpdateProduct
- TransferOwnership
- QueryProduct

Task 1: Implement InitLedger

Objective:

Initialize the ledger with some example products.

Implementation:

- Create an array of Product structs with predefined values.
- In the provided InitLedger method, there is an example product with the following attributes:

  ID: "p1"
  Name: "Laptop"
  Status: "Manufactured"
  Owner: "CompanyA"
  CreatedAt: The current time
  UpdatedAt: The current time
  Description: "High-end gaming laptop"

Category: "Electronics"

Add a second product to the InitLedger method with the following attributes:

ID: "p2"
Name: "Smartphone"
Status: "Manufactured"
Owner: "CompanyB"
CreatedAt: The current time
UpdatedAt: The current time
Description: "Latest model smartphone"
Category: "Electronics"

Ensure that the InitLedger method initializes the ledger with both the "Laptop" and "Smartphone" products.

Test:

- Assert that the InitLedger method returns a 200 status code.
- Assert that the retrieved product's attributes match the expected values for each product initialized in the ledger.

Task 2: Implement CreateProduct

Objective:

Create a new product in the ledger.

Implementation:

- Create a Product struct with the provided parameters.
- Set the CreatedAt and UpdatedAt fields to the current time.
- Helper methods can be useful.

Test:

- Assert that the product does not exist before creation.
- Assert that the CreateProduct method returns a 200 status code.
- Assert that the product exists after creation.
- Assert that the retrieved product's attributes match the expected values.

Task 3: Implement UpdateProduct

Objective:

Update a product's status, owner, description, and category.

Implementation:

- Update the existing product's relevant fields with the new values.
- Set the UpdatedAt field to the current time.

Test:

- Assert that the UpdateProduct method returns a 200 status code.
- Assert that the updated product's attributes match the expected values.

Task 4: Implement TransferOwnership

Objective:

Change the owner of a product.

Implementation:

- Update the Owner field of an existing product with the new owner.
- Set the UpdatedAt field to the current time.

Test:

- Assert that the UpdateProduct method returns a 200 status code.
- Assert that the updated product's attributes match the expected values.

Task 5: Implement QueryProduct

Objective:

Retrieve a single product from the ledger by ID.

Test:

- Assert that the QueryProduct method returns a 200 status code for an existing product.
- Assert that the retrieved product's attributes match the expected values.

- Assert that the QueryProduct method returns a 500 status code for a non-existent product and also returns an error message saying that `the product does not exist`.

Please implement these methods correctly to ensure that all the tests pass, demonstrating the functionality of your smart contract.

Setup Network and Test Chaincode on it

1. Install the Prerequisites

- Install Docker and Docker Compose from the official website.
- Install jq from its official website.
- Install the latest version of cURL.

2. Setting up Hyperledger Fabric

- Download the necessary binaries and Docker images.

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

- Clone the Hyperledger Fabric samples repository from GitHub.

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

3. Run a test network

Bring up the network along with two example orgs.

- cd into the test-network directory containing the main network.sh script:

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

- run the following command to bring up a fresh test-network:

```
./network.sh down && ./network.sh up
```

- run the following command to see the running docker-containers:

```
docker ps -a
```

- You should see 4 containers with the following names:

```
hyperledger/fabric-tools:latest: 1 container
hyperledger/fabric-orderer:latest: 1 container
hyperledger/fabric-peer:latest: 2 containers
```

- Create a channel and have the two example orgs join it.

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

  You should see the following message at the end of the output:

  ```
  Channel 'mychannel' joined
  ```

4. Navigate to the Project Directory

After making sure that your go file compiles and is ready to test,

Change to the project directory: cd CSE598-EBA-Project2

5. Deploy a smart contract on the network

Go to the folder containing the golang Chaincode and install golang dependencies there,

- Run `go mod vendor`.
- Then return to the current directory test-network.
- Now add the following variables to your cli:

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

- Finally, create the package for the smart contract:

  ```
  peer lifecycle chaincode package supplyChain.tar.gz --path <path-to-code> --lang golang
  --label supplyChain
  ```

  Replace <path-to-code> with the directory path where your smart contract is set up.

- Run the following command to ensure that the package has been created:

  ```
  ls | grep supplyChain.tar.gz
  ```

Install the chaincode package on both the orgs

- First, let's take up the role of org 1:

  export CORE_PEER_TLS_ENABLED=true

  export CORE_PEER_LOCALMSPID="Org1MSP"

  export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

  export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

  export CORE_PEER_ADDRESS=localhost:7051

- Install the chaincode onto the peer (org1):

  peer lifecycle chaincode install supplyChain.tar.gz

- Next, let's take up the role of org 2:

  export CORE_PEER_LOCALMSPID="Org2MSP"

  export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

  export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

  export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

  export CORE_PEER_ADDRESS=localhost:9051

- Install the chaincode onto the peer (org2):

  peer lifecycle chaincode install supplyChain.tar.gz

Approve the chaincode on both the orgs

- First, let's take up the role of org 1:

  export CORE_PEER_TLS_ENABLED=true

  export CORE_PEER_LOCALMSPID="Org1MSP"

export
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.exampl
e.com/peers/peer0.org1.example.com/tls/ca.crt

export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.co
m/users/Admin@org1.example.com/msp

export CORE_PEER_ADDRESS=localhost:7051

- Ensure that the chaincode is installed on org1:

  peer lifecycle chaincode queryinstalled

  Installed chaincodes on peer:

  Package ID:
  supplyChain:1146b4b491871bf18b23dd67dd8cc058655b36cc0e2274f165ed06b796a8f276,
  Label: supplyChain

- Copy the Package ID from the result in the previous command, and set the following variable equal to it:

  CC_PACKAGE_ID=supplyChain:1146b4b491871bf18b23dd67dd8cc058655b36cc0e2274f165
  ed06b796a8f276

- Approve the chaincode for org1:

  peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride
  orderer.example.com --channelID mychannel --name supplyChain --version 1.0 --package-id
  $CC_PACKAGE_ID --sequence 1 --tls true --cafile
  ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
  tlscacerts/tlsca.example.com-cert.pem

- Next, let's take up the role of org 2:

  export CORE_PEER_LOCALMSPID="Org2MSP"

  export
  CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.exampl
  e.com/peers/peer0.org2.example.com/tls/ca.crt

  export
  CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.exampl
  e.com/peers/peer0.org2.example.com/tls/ca.crt

  export
  CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.co
  m/users/Admin@org2.example.com/msp

```
export CORE_PEER_ADDRESS=localhost:9051
```

- Ensure that the chaincode is installed onto org 2:

```
peer lifecycle chaincode queryinstalled

Installed chaincodes on peer:

Package ID:
supplyChain:1146b4b491871bf18b23dd67dd8cc058655b36cc0e2274f165ed06b796a8f276,
Label: supplyChain
```

- Copy the Package ID from the result in the previous command, and set the following variable equal to it:

```
CC_PACKAGE_ID=supplyChain:0d7df43c75eadc05c423f1f842e2c24f8d15cec4c6b94aa9601
1c461e7c60a73
```

- Now approve the chaincode for org2:

```
peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com --channelID mychannel --name supplyChain --version 1.0 --package-id
$CC_PACKAGE_ID --sequence 1 --tls true --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem
```

- Commit the chaincode to the channel

  Check if the chaincode is ready to be committed:

```
peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name supplyChain
--version 1.0 --sequence 1 --tls true --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem --output json
```

- Commit the chaincode to the channel:

```
peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com --channelID mychannel --name supplyChain --version 1.0 --sequence 1
--tls true --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem --peerAddresses localhost:7051 --tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tl
s/ca.crt --peerAddresses localhost:9051 --tlsRootCertFiles
${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tl
s/ca.crt
```

- Check if the chaincode is committed:

```
peer lifecycle chaincode querycommitted --channelID mychannel --name supplyChain --cafile
${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem
```

6. Invoke the chaincode

Now that you have committed the chaincode onto the channel, finally it's time to have some fun!

- Let's invoke various functions of the committed chaincode:

  Call the GetAllProducts function:

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

- As you can see by the empty array, there are no products on the ledger, so let's try adding some.

  Call the initLedger function:

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

- This should add some products to the ledger, let's check it by querying all cars again.

  Call the GetAllProducts function:

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

  ```
  [{ID: "p1", Name: "Laptop", Status: "Manufactured", Owner: "CompanyA", CreatedAt:
  time.Now(), UpdatedAt: time.Now(), Description: "High-end gaming laptop", Category:
  "Electronics"}, {ID: "p2", Name: "Smartphone", Status: "Manufactured", Owner: "CompanyB",
  CreatedAt: time.Now(), UpdatedAt: time.Now(), Description: "Latest model smartphone",
  Category:"Electronics"}]
  ```

Deployment Automation

To automate the deployment of chaincode and reduce steps, a script has been created in the git repository. Use that script after you understand what is happening in each step of the deployment for the subsequent testing.

Script

1. Navigate to Fabric Samples Test Network
Navigate to the Fabric samples test network directory, which is part of the Hyperledger Fabric installation:

- cd path/to/fabric-samples/test-network
- Replace path/to/fabric-samples with the actual path to your Fabric samples directory.

2. Copy Deployment Script
Copy the deployment script deploy_chaincode.sh from your project directory to the Fabric samples test network directory:

- cp path/to/CSE598-EBA-Project2/deploy_chaincode.sh .
- Replace path/to/CSE598-EBA-Project2 with the actual path to your project directory.

3. Execute Deployment Script
Give execute permissions to the deployment script and run it:

Run the deployment script with the path to the Go smart contract directory as an argument, should be replaced with the actual directory path

- chmod +x deploy_chaincode.sh
- ./deploy_chaincode.sh “<path-to-code>”

This script will deploy your chaincode to the Hyperledger Fabric network. Make sure to copy and enter the package ID for each organization when prompted.

Executing Chaincode Operations

Once the SupplyChain smart contract has been successfully deployed, you can interact with it using the following operations to test remaining functionalities as defined in the task requirements.

Submission Directions for Project Deliverables

The autograder is not intended for debugging or incremental testing of your Supply Chain smart contract. It should be used only when you believe your smart contract is fully functional and ready for grading. For testing individual functions and overall functionality, utilize the locally deployed Hyperledger network.

Deliverables

- You must submit a Go file for the Supply Chain Management smart contract named **“smartcontract.go”**.

Submitting to Gradescope

Your submissions will be graded automatically. After completing your work, you must submit to Gradescope to receive credit for the course. You will have unlimited submission attempts, but the course team will only review the latest submission.

1. Go to the Canvas Assignment, “**Submission: Hyperledger Fabric Private Blockchain and Chaincodes Project**”.
2. Click the "**Load Submission…in new window**" button.
3. Once in Gradescope, select the project titled "**Submission: Hyperledger Fabric Private Blockchain and Chaincodes Project**" and a pop-up window will appear.
4. In the pop-up window, submit your “**smartcontract.go**” file.
5. If needed: to resubmit the project in Gradescope:
    a. Return to the Canvas submission and open Gradescope.
    b. You will be navigated to the "Autograder Results" page (if it is not your first submission).
    c. Click the "**Resubmit**" button on the bottom right corner of the page and repeat the process from Step 3.

Your submission will be reviewed by the course team and then, after the due date has passed, your score will be populated from Gradescope into your Canvas grade.

Evaluation

This project carries a total of 210 points. The autograder will directly invoke the methods and events of your smart contract deployed on the Hyperledger Fabric network as per the provided instructions. The evaluation of your project hinges entirely on the precise and correct implementation of your smart contract methods. Your grade, along with details of which methods/events passed or failed, will be the output. To aid in debugging and resolving issues, code logs are available to you on the Gradescope UI.

- 42 points: If the InitLedger method is correct.
- 42 points: If the CreateProduct method is correct.
- 42 points: If the UpdateProduct method is correct.
- 42 points: If the TransferOwnership method is correct.
- 42 points: If the QueryProduct method is correct.