DOP: /   /2023   DOS: / / 2023

Experiment No: 05

Aim: To deploy a chain code on a permissioned blockchain.

Theory:

Hyperledger Fabric

Hyperledger Fabric is a blockchain framework that provides a modular and scalable architecture for building enterprise-grade distributed ledger applications. Chaincode is a term used in Hyperledger Fabric to refer to the smart contracts that run on the network. These chaincodes are written in a programming language such as Go, JavaScript, or Java.

Hyperledger Fabric's permissioned blockchain is designed to provide fine-grained access control, where participants must be authenticated and authorized before they can access the network. To deploy chaincode on Hyperledger Fabric, we need to follow a few steps.

Steps:

1. Set up the environment: First, we need to set up the development environment for Hyperledger Fabric. We need to install the required software, including Docker, Docker Compose, Go, and Node.js. Here's how to do it:

a. Install Docker: Download and install Docker from the official website: https://www.docker.com/get-started

b. Install Docker Compose: Download and install Docker Compose from the official website: https://docs.docker.com/compose/install/

c. Install Go: Download and install Go from the official website: https://golang.org/dl/

d. Install Node.js: Download and install Node.js from the official website: https://nodejs.org/en/download/

2. Create a network: We need to create a network of peers and orderers that will form the blockchain network. We can use tools such as Hyperledger Fabric's test-network or create a custom network. Follow the official documentation to set up the network. Here is an example of creating a test network using the Hyperledger Fabric CLI:

a. Download the test-network: Download the test-network by cloning the repository from GitHub using the following command in your terminal or command prompt:

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

cd fabric-samples/test-network

b. Start the network: Start the network by running the following command:

`./network.sh up createChannel -ca`

This command will start the network, create a channel, and generate the required cryptographic material for the network. It may take a few minutes for the network to start up.

c. Verify the network: Verify that the network is running by running the following command:

`./network.sh list`

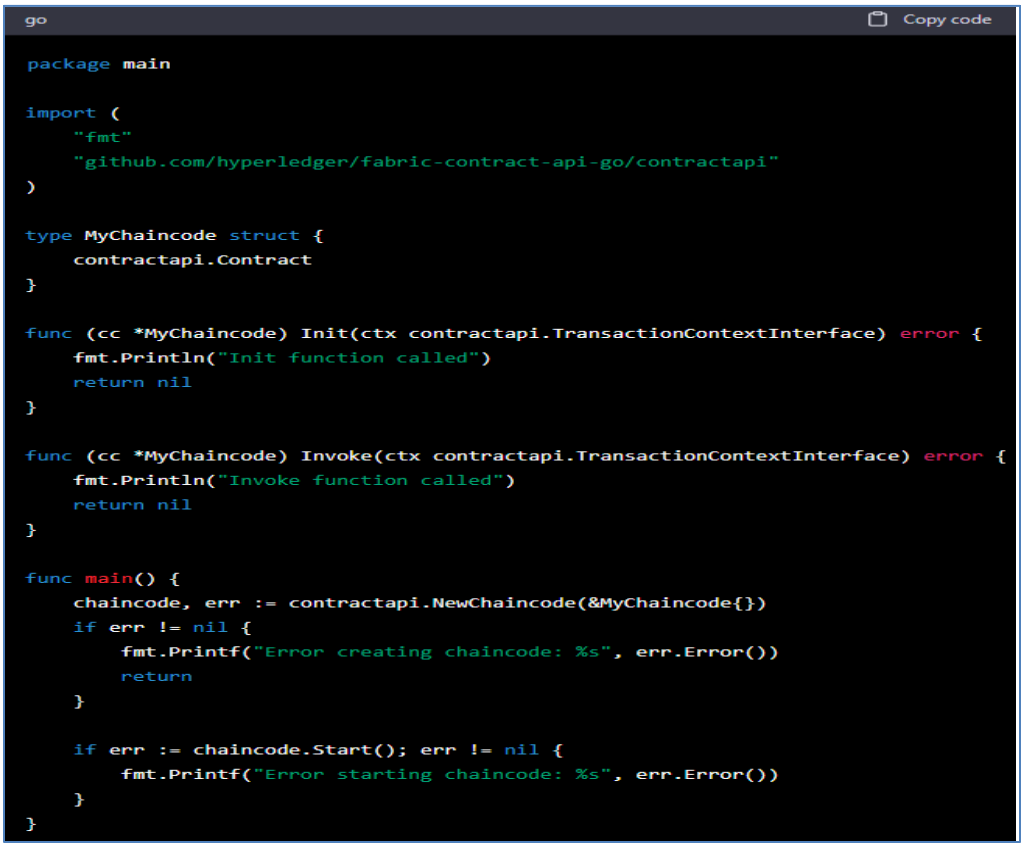
This command will list the running containers and should show a list of containers with names starting with "peer".

3. Write the chaincode: We need to write the chaincode in a programming language supported by Hyperledger Fabric. We can use a text editor or an Integrated Development Environment (IDE) to write the code. Here is an example of a simple chaincode written in Go:

a. Create a new directory: Create a new directory for the chaincode using the following command: `mkdir -p chaincode/mychaincode`

b. Create a new file: Create a new file named "mychaincode.go" in the chaincode/mychaincode directory using the following command: `touch chaincode/mychaincode/mychaincode.go`

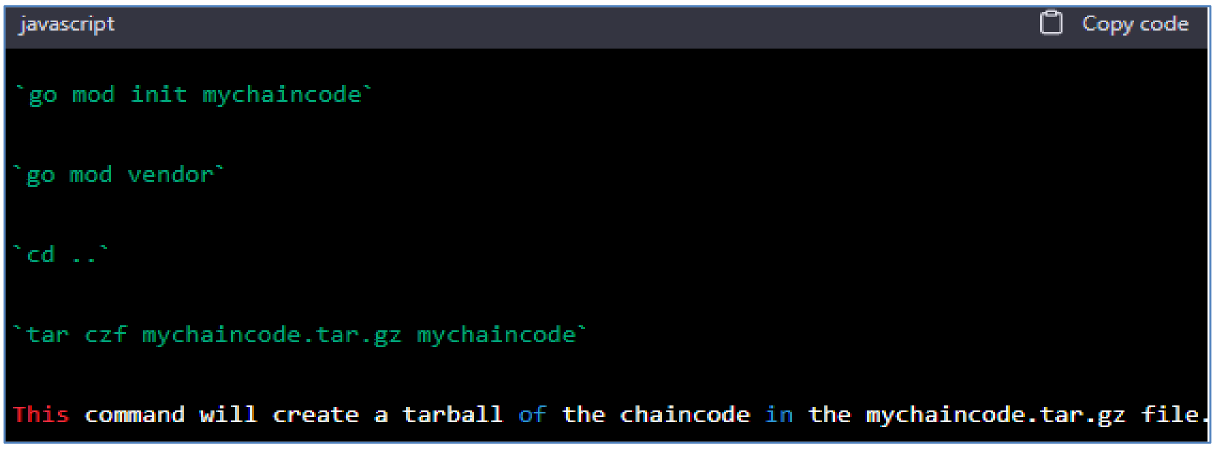
c. Write the chaincode code.go file in a text editor or IDE and write the sample chaincode in Go language. Here's an example.



```go
package main

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

type MyChaincode struct {
    contractapi.Contract
}

func (cc *MyChaincode) Init(ctx contractapi.TransactionContextInterface) error {
    fmt.Println("Init function called")
    return nil
}

func (cc *MyChaincode) Invoke(ctx contractapi.TransactionContextInterface) error {
    fmt.Println("Invoke function called")
    return nil
}

func main() {
    chaincode, err := contractapi.NewChaincode(&MyChaincode{})
    if err != nil {
        fmt.Printf("Error creating chaincode: %s", err.Error())
        return
    }

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

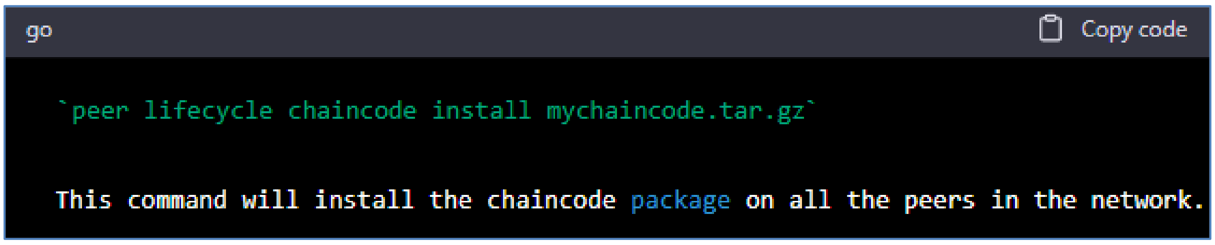
4. Package and install the chaincode:Once we have written the chaincode, we need to package it into a binary format and install it on the peers. Follow these steps:

a. Package the chaincode: Package the chaincode



```javascript
`go mod init mychaincode`

`go mod vendor`

`cd ..`

`tar czf mychaincode.tar.gz mychaincode`

This command will create a tarball of the chaincode in the mychaincode.tar.gz file.
```

b. Install the chaincode: Install the chaincode on the peers using the following command:



```go
  `peer lifecycle chaincode install mychaincode.tar.gz`

  This command will install the chaincode package on all the peers in the network.
```

c. Approve the chaincode: Approve the chaincode definition using the following command:

`peer lifecycle chaincode approveformyorg --channelID mychannel --name mychaincode --version 1.0 --package-id PACKAGE_ID --sequence 1`

Replace PACKAGE_ID with the package ID generated in the previous step. This command will approve the chaincode definition for your organization.

d. Commit the chaincode: Commit the chaincode definition using the following command:

`peer lifecycle chaincode commit -o localhost:7050 --channelID mychannel --name mychaincode --version 1.0 --sequence 1 --tls true --cafile /path/to/orderer/ca.crt --peerAddresses localhost:7051 --tlsRootCertFiles /path/to/peer/tls/ca.crt`

5. Test the chaincode:

Once the chaincode is deployed and running, we can test it by invoking its functions. Follow these steps:

a. Open the Hyperledger Fabric CLI: Open a new CLI to the test-network directory:

`cd fabric-samples/test-network`

b. Set environment variables: Set the environment variables for the peer and channel by running the following commands:

export PATH=${PWD}/../bin:$PATH

export FABRIC_CFG_PATH=$PWD/../config/

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

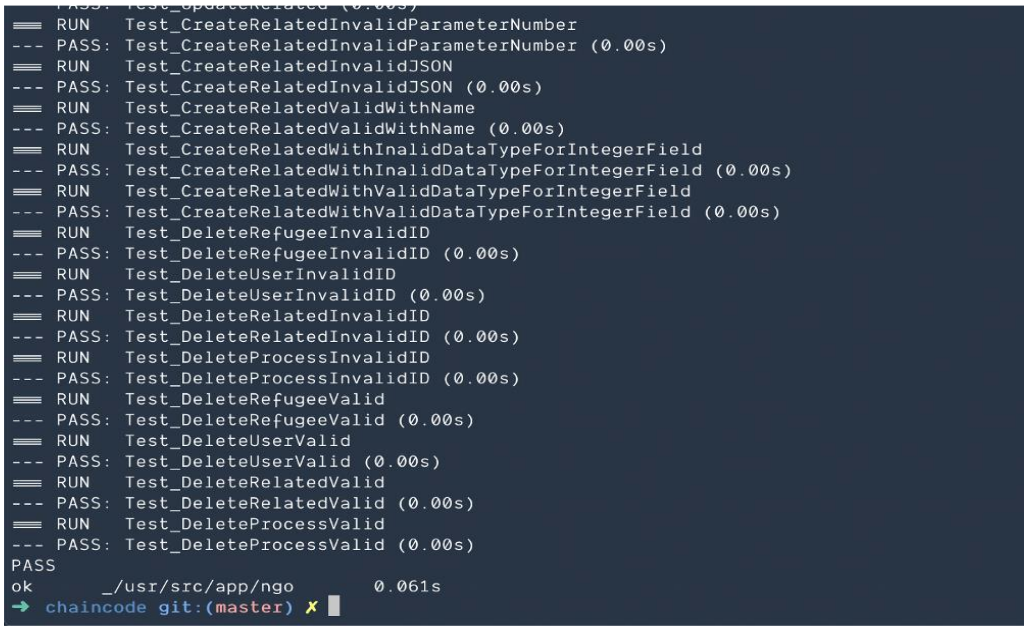
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

export CORE_PEER_ADDRESS=localhost:7051

export CHANNEL_NAME=mychannel

c. Invoke the chaincode: Invoke the chaincode using the following command:

`peer chaincode invoke -o localhost:7050 --tls true --cafile /path/to/orderer/ca.crt -C $CHANNEL_NAME -n mychaincode --peerAddresses localhost:7051 --tlsRootCertFiles /path/to/peer/tls/ca.crt -c '{"function":"Invoke","Args":[]}'`



```
=== RUN   Test_CreateRelatedInvalidParameterNumber
--- PASS: Test_CreateRelatedInvalidParameterNumber (0.00s)
=== RUN   Test_CreateRelatedInvalidJSON
--- PASS: Test_CreateRelatedInvalidJSON (0.00s)
=== RUN   Test_CreateRelatedValidWithName
--- PASS: Test_CreateRelatedValidWithName (0.00s)
=== RUN   Test_CreateRelatedWithInalidDataTypeForIntegerField
--- PASS: Test_CreateRelatedWithInalidDataTypeForIntegerField (0.00s)
=== RUN   Test_CreateRelatedWithValidDataTypeForIntegerField
--- PASS: Test_CreateRelatedWithValidDataTypeForIntegerField (0.00s)
=== RUN   Test_DeleteRefugeeInvalidID
--- PASS: Test_DeleteRefugeeInvalidID (0.00s)
=== RUN   Test_DeleteUserInvalidID
--- PASS: Test_DeleteUserInvalidID (0.00s)
=== RUN   Test_DeleteRelatedInvalidID
--- PASS: Test_DeleteRelatedInvalidID (0.00s)
=== RUN   Test_DeleteProcessInvalidID
--- PASS: Test_DeleteProcessInvalidID (0.00s)
=== RUN   Test_DeleteRefugeeValid
--- PASS: Test_DeleteRefugeeValid (0.00s)
=== RUN   Test_DeleteUserValid
--- PASS: Test_DeleteUserValid (0.00s)
=== RUN   Test_DeleteRelatedValid
--- PASS: Test_DeleteRelatedValid (0.00s)
=== RUN   Test_DeleteProcessValid
--- PASS: Test_DeleteProcessValid (0.00s)
PASS
ok      _/usr/src/app/ngo       0.061s
➜ chaincode git:(master) ✗
```

Conclusion: In this lab experiment, we learned how to deploy a chaincode on a permissioned blockchain in Hyperledger Fabric. We started by setting up a Hyperledger Fabric network using the test-network script, then we wrote a sample chaincode in Go language and packaged it into a binary format. We installed the chaincode on the peers, approved its definition, and committed it to the channel. Finally, we tested the chaincode by invoking its functions using the Hyperledger Fabric CLI. This lab experiment provides a basic understanding of how to deploy a chaincode on a permissioned blockchain in Hyperledger Fabric and can be used as a starting point for developing more complex chaincodes and applications.