PROJECT 2 - Hyperledger Fabric Private Blockchain - Supply Chain Smart Contract

This project demonstrates the development, deployment, and testing of a private blockchain-based Supply Chain Management system on the Hyperledger Fabric platform. The application is implemented as a smart contract in Go, with a focus on managing and tracking product ownership, status, and other attributes across the supply chain.

Overview

The project utilizes Hyperledger Fabric's modular architecture to implement a smart contract for supply chain operations. The smart contract includes methods to initialize a ledger, create new products, update product details, transfer ownership, and query product information.

Key tasks involved:

1. Setting up a private blockchain network using Hyperledger Fabric.
2. Developing the smart contract logic in Go, with core functionality for supply chain management.
3. Deploying and testing the chaincode on the network.

Features and Functionality

Smart Contract Core Methods

1. **InitLedger**: Initializes the ledger with predefined products. Example entries include:
   - **Product 1**:
     - ID: `p1`
     - Name: `Laptop`
     - Status: `Manufactured`
     - Owner: `CompanyA`
     - Description: `High-end gaming laptop`
     - Category: `Electronics`
   - **Product 2**:
     - ID: `p2`
     - Name: `Smartphone`
     - Status: `Manufactured`
     - Owner: `CompanyB`
     - Description: `Latest model smartphone`
     - Category: `Electronics`

2. **CreateProduct**: Adds a new product to the ledger with attributes such as ID, name, status, owner, description, and category. Automatically timestamps the creation and update times.
3. **UpdateProduct**: Updates existing product details, including status, owner, description, and category, with the timestamp reflecting the most recent update.
4. **TransferOwnership**: Transfers ownership of a product to a new entity. Retrieves the existing product using its ID and updates the owner field.
5. **QueryProduct**: Fetches product details by ID. Returns appropriate error messages if the product does not exist.
6. **GetAllProducts** (Helper Method): Retrieves all products stored in the ledger.
7. **putProduct** (Helper Method): Handles product insertions and updates in the ledger.
8. **ProductExists** (Helper Method): Verifies whether a product exists in the ledger.

Project Setup

Technology Requirements

- **Hyperledger Fabric**: Blockchain platform for private and permissioned networks.
- **Go Programming Language**: Used for developing the chaincode.
- **Docker & Docker Compose**: Essential for running Hyperledger Fabric networks.
- **Git**: For version control.

Development Environment

1. **Install Go**: Download and install Go from official site.
2. **Install Docker**: Install Docker and Docker Compose from official site.
3. **Clone Project Repository**: Use Git to clone the skeleton project.
4. `git clone https://github.com/pavankramadugu/CSE598-EBA-Project2.git`
5. `cd CSE598-EBA-Project2`
6. `go mod tidy`

Deployment Steps

Set Up Hyperledger Fabric

1. Download the necessary binaries and Docker images:
2. `curl -sSL https://raw.githubusercontent.com/hyperledger/fabric/master/scripts/bootstrap.sh | bash -s`
3. Clone the Hyperledger Fabric samples repository:
4. `git clone https://github.com/hyperledger/fabric-samples`
5. Start the network:
6. `cd fabric-samples/test-network`
7. `./network.sh down && ./network.sh up`

8. Create a channel and add organizations:

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

Deploy the Chaincode

1. Package the chaincode:

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

3. Install the chaincode on peers for both organizations:

```
4. export CORE_PEER_LOCALMSPID="Org1MSP"
5. export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
6. export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
7. export CORE_PEER_ADDRESS=localhost:7051
8.
9. peer lifecycle chaincode install supplyChain.tar.gz
```

10. Approve and commit the chaincode:

```
11.  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
```

12. Test the deployment:

```
13.  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
```

14. Invoke chaincode methods to verify functionality:
    - Initialize the ledger:

    -
      ```
      peer chaincode invoke -C mychannel -n supplyChain -c '{"function":"initLedger","Args":[]}'
      ```
    - Query all products:

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

Testing and Validation

1. Tested all methods in the chaincode for expected outcomes.
2. Verified successful ledger initialization with predefined products.
3. Validated error handling for invalid or non-existent product queries.

Learning Outcomes

- Gained expertise in setting up and managing Hyperledger Fabric networks.
- Developed and deployed a fully functional private blockchain application for supply chain management.
- Learned to interact with chaincode using CLI commands and validate blockchain operations.

This documentation maintains the integrity of the original project details while streamlining it for GitHub. Let me know if you'd like further refinements!