

Blockchain-Based Asset Management System using Hyperledger Fabric

This project is a complete, end-to-end blockchain application built for a financial institution, using Hyperledger Fabric. It provides a secure, transparent, and immutable ledger for tracking and managing financial accounts.

The system is composed of a Hyperledger Fabric network, a smart contract (chaincode) written in Go, and a containerized REST API for client interaction, fulfilling all requirements of the internship assignment.

Table of Contents

1. Problem Statement
2. Project Architecture
3. Technologies Used
4. Prerequisites
5. Step-by-Step Installation and Execution Guide
 - Step 1: Environment Setup
 - Step 2: Start the Blockchain Network (Terminal 1)
 - Step 3: Deploy the Smart Contract (Terminal 1)
 - Step 4: Configure and Start the REST API (Terminal 2)
 - Step 5: Test the Live Application (Terminal 3)
6. Bringing Down the Application

Problem Statement

A financial institution requires a blockchain solution to manage and track assets representing client accounts. The system must ensure the security, transparency, and immutability of these asset records.

The core functionalities required are:

- Asset creation
- Updating asset values
- Querying the world state to read assets
- Retrieving the transaction history for an asset

Each asset represents an account with the following attributes:

- DEALERID

- MSISDN
- MPIN
- BALANCE
- STATUS
- TRANSAMOUNT
- TRANSTYPE
- REMARKS

Project Architecture

The project consists of three main components as specified in the assignment levels:

1. **Hyperledger Fabric Network:** A test network configured with two organizations (Org1, Org2) and a Raft ordering service. This provides the distributed ledger infrastructure.
2. **Go Smart Contract (Chaincode):** The business logic of the application, deployed on the Fabric network. It defines the functions that can interact with the ledger, such as CreateAsset, ReadAsset, and GetAssetHistory.
3. **Go REST API:** A client-facing web server that exposes simple HTTP endpoints to invoke the smart contract. It uses the Fabric Gateway SDK to communicate with the blockchain and is containerized using Docker.

Technologies Used

- **Blockchain:** Hyperledger Fabric v2.5
- **Programming Language:** Go (for both the chaincode and the REST API)
- **Containerization:** Docker & Docker Compose
- **Environment:** Ubuntu / Windows Subsystem for Linux (WSL2)

Prerequisites

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

- Docker and Docker Compose
- Go programming language (version 1.21 or higher)
- Git
- curl (for testing the API)

Step-by-Step Installation and Execution Guide

The application must be started in a specific order: first the blockchain network, and then the API server. This requires **at least two separate terminals**.

Step 1: Environment Setup

1. Clone this repository to your local machine.
2. # Replace with your actual repository URL
3. `git clone [https://github.com/kernelfatima/ Hyperledger-Fabric-Management-System.git]`
4. `cd fabric-internship-project`
5. The fabric-samples directory contains the scripts needed to run the test network. Download the required Hyperledger Fabric binaries and Docker images.
6. `cd fabric-samples/test-network`
7. `./network.sh prereq`

Step 2: Start the Blockchain Network (Terminal 1)

1. Open your **first terminal** window.
2. Navigate to the test-network directory:
3. `cd ~/fabric-internship-project/fabric-samples/test-network`
4. Bring down any old network instances to ensure a clean start:
5. `./network.sh down`
6. Start the Fabric network and create the channel using Certificate Authorities.
7. `./network.sh up createChannel -ca`

Step 3: Deploy the Smart Contract (Terminal 1)

1. In the same terminal, deploy the Go smart contract. Since this is a fresh network, the sequence number must be 1.
2. `./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go -ccv 1.0 -ccs 1`

```
kernel@fatima:~/fabric-internship-project/fabric-samples/test-network$ ./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go -ccv 1.1 -ccs 1
Using docker and docker-compose
Deploying chaincode on channel 'mychannel'
executing with the following
- CHANNEL_NAME: mychannel
- CC_NAME: basic
- CC_SRC_PATH: ../asset-transfer-basic/chaincode-go
- CC_SRC_LANGUAGE: go
- CC_VERSION: 1.1
- CC_SEQUENCE: 1
- CC_END_POLICY: NA
- CC_COLL_CONFIG: NA
- CC_INIT_FCN: NA
- DELAY: 3
- MAX_RETRY: 5
- VERBOSE: false
executing with the following
- CC_NAME: basic
- CC_SRC_PATH: ../asset-transfer-basic/chaincode-go
- CC_SRC_LANGUAGE: go
- CC_VERSION: 1.1
Vendors Go dependencies at ../asset-transfer-basic/chaincode-go
~/fabric-internship-project/fabric-samples/asset-transfer-basic/chaincode-go ~/fabric-internship-project/fabric-samples/test-network
Finished vendoring Go dependencies
+ '[' false = true ']'
+ peer lifecycle chaincode package basic.tar.gz --path ../asset-transfer-basic/chaincode-go --lang golang --label basic_1.1
+ res=0
Chaincode is packaged
Installing chaincode on peer0.org1...
Using organization 1
+ peer lifecycle chaincode queryinstalled --output json
+ jq -r 'try (.installed_chaincodes[].package_id)'
+ grep '^basic_1.1:39b0ac278864904208997236ff03f245fb9d4ad9150879d1b94589652188fa81|022|tbasic_1.1' >
+ test 1 -ne 0
+ peer lifecycle chaincode install basic.tar.gz
+ res=0
2025-10-05 06:43:12.062 UTC 0001 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Installed remotely: response:<status:200 payload:"\njbasic_1.1:39b0ac278864904208997236ff03f245fb9d4ad9150879d1b94589652188fa81|022|tbasic_1.1" >
2025-10-05 06:43:12.063 UTC 0002 INFO [cli.lifecycle.chaincode] submitInstallProposal -> Chaincode code package identifier: basic_1.1:39b0ac2788649042089972
```

```
kernel@fatima:~/fabric-internship-project/fabric-samples/test-network$ ./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go -ccv 1.1 -ccs 1
Attempting to check the commit readiness of the chaincode definition on peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name basic --version 1.1 --sequence 1 --output json
+ res=0
{
  "approvals": {
    "Org1MSP": true,
    "Org2MSP": true
  }
}
Checking the commit readiness of the chaincode definition successful on peer0.org2 on channel 'mychannel'
Using organization 1
+ peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile /home/kernel@fatima/fabric-internship-project/fabric-samples/test-network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --channelID mychannel --name basic --peerAddresses localhost:7051 --tlsRootCertFiles /home/kernel@fatima/fabric-internship-project/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/tlsca/tlsca.org1.example.com-cert.pem --peerAddresses localhost:9051 --tlsRootCertFiles /home/kernel@fatima/fabric-internship-project/fabric-samples/test-network/organizations/peerOrganizations/org2.example.com/tlsca/tlsca.org2.example.com-cert.pem --version 1.1 --sequence 1
+ res=0
2025-10-05 06:43:51.492 UTC 0001 INFO [chaincodeCmd] ClientWait -> txid [a8338dc7c49cca2471c6bb3eb84212baf3cab02bea1424b00c63b28e2d01ecc] committed with status (VALID) at localhost:9051
2025-10-05 06:43:51.494 UTC 0002 INFO [chaincodeCmd] ClientWait -> txid [a8338dc7c49cca2471c6bb3eb84212baf3cab02bea1424b00c63b28e2d01ecc] committed with status (VALID) at localhost:7051
Chaincode definition committed on channel 'mychannel'
Using organization 1
Querying chaincode definition on peer0.org1 on channel 'mychannel'...
Attempting to Query committed status on peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'mychannel':
Version: 1.1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vsc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org1 on channel 'mychannel'
Using organization 2
Querying chaincode definition on peer0.org2 on channel 'mychannel'...
Attempting to Query committed status on peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'mychannel':
Version: 1.1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vsc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org2 on channel 'mychannel'
Chaincode initialization is not required
kernel@fatima:~/fabric-internship-project/fabric-samples/test-network$ cd ../financial-api
```

Leave this terminal open. It is now running your live blockchain network.

Step 4: Configure and Start the REST API (Terminal 2)

1. Open a new, second terminal window.
2. **Important:** The API code needs the IP address of your WSL instance to connect to the Fabric peer. Find your IP address by running:

3. `ip addr show eth0 | grep "inet\s" | awk '{print $2}' | cut -d/ -f1`

Copy the IP address it returns (e.g., 172.25.1.38).

4. Navigate to the financial-api directory and open the main.go file to update the endpoint.

5. `cd ~/fabric-internship-project/fabric-samples/financial-api`
6. `nano main.go`
7. In the editor, find the `peerEndpoint` constant and

```

kernel@fatima:~/fabri
Query chaincode definition successful on peer0.org2 on channel 'mychannel'
kernel@fatima:~/fabric-internship-project/fabric-samples/test-network$ cd ../financial-api
kernel@fatima:~/fabric-internship-project/fabric-samples/financial-api$ docker build -t financial-api .
[+] Building 3.2s (15/15) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 597B
=> [internal] load metadata for docker.io/library/golang:1.24
=> [internal] load metadata for docker.io/library/alpine:3.18
=> [internal] load dockerignore
=> => transferring context: 2B
=> [builder 1/6] FROM docker.io/library/golang:1.24@sha256:2c5f7a0c252a17cf6aa38ddee15caa0f485ee29410a6ea64cddb62eea2b07bdf
=> => resolve docker.io/library/golang:1.24@sha256:2c5f7a0c252a17cf6aa38ddee15caa0f485ee29410a6ea64cddb62eea2b07bdf
=> [internal] load build context
=> => transferring context: 111B
=> [stage-1 1/3] FROM docker.io/library/alpine:3.18@sha256:de0eb0b3f2a47baleb89389859a9bd88b28e82f5826b6969ad604979713c2d4f
=> => resolve docker.io/library/alpine:3.18@sha256:de0eb0b3f2a47baleb89389859a9bd88b28e82f5826b6969ad604979713c2d4f
=> [stage-1 2/3] RUN apk add --no-cache ca-certificates
=> CACHED [builder 2/6] WORKDIR /app
=> CACHED [builder 3/6] COPY go.mod go.sum ./
=> CACHED [builder 4/6] RUN go mod download
=> CACHED [builder 5/6] COPY
=> CACHED [builder 6/6] RUN CGO_ENABLED=0 GOOS=linux go build -o /server
=> CACHED [stage-1 3/3] COPY --from=builder /server /server
=> => exporting to image
=> => exporting layers
=> => exporting manifest sha256:2057dd615fad2db5d5072fbc4d7f9e4e2e0c92ef35388bc6a968f8e3db048f5
=> => exporting config sha256:5f99aed415f2670179229777fa3c9357108f4198a80b23229a428453e976c01ed
=> => exporting attestations manifest sha256:b99f703969797a81e72cafd8d218f963d2135e93665cc071c9480716a0d18fb3de
=> => exporting manifest list sha256:cf2cc8a761b99e4c91c6967579112315124f53cab835fedcc3e903aaccb9e8
=> => naming to docker.io/library/financial-api:latest
=> => unpacking to docker.io/library/financial-api:latest
kernel@fatima:~/fabric-internship-project/fabric-samples/financial-api$ docker run --rm -p 8080:8080 --name financial-api \
-v ${PWD}/../test-network:/test-network \
financial-api
2025/10/05 06:45:40 ===== application-golang starts =====
2025/10/05 06:45:40 Starting server on port 8080
2025/10/05 06:46:13 --> Submit Transaction: CreateAsset, for dealer DEALER001
2025/10/05 06:46:15 --> Evaluate Transaction: ReadAsset, for dealer DEALER001
2025/10/05 06:48:45 --> Evaluate Transaction: ReadAsset, for dealer DEALER001

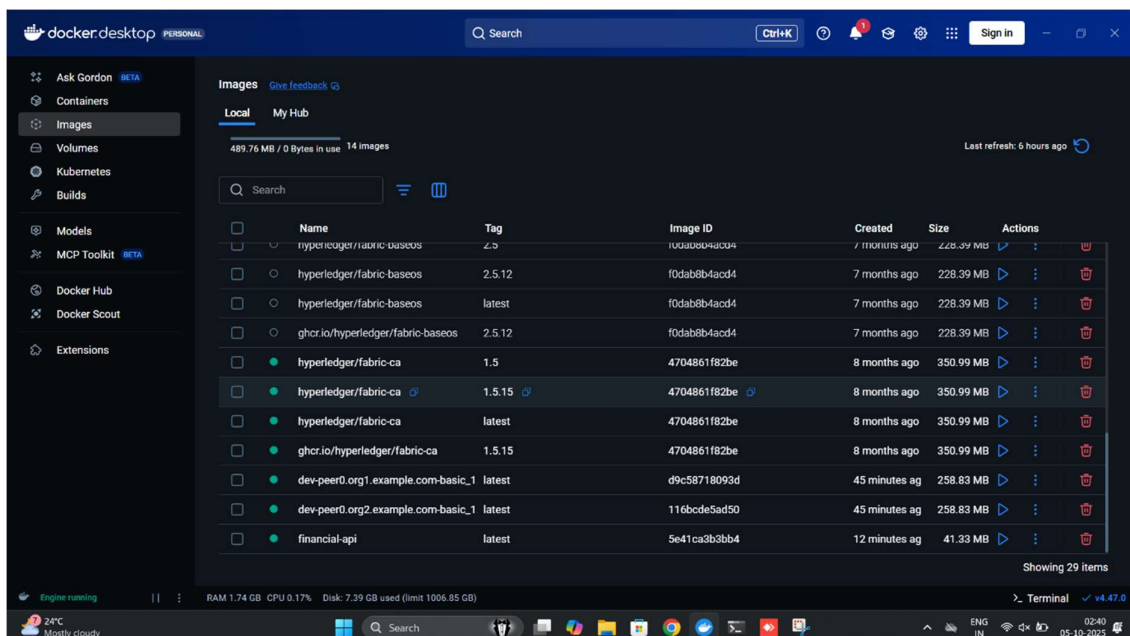
```

Replace the placeholder with your actual IP address.

8. `// Example:`
9. `peerEndpoint = "172.25.1.38:7051"`

Save and exit the editor (`Ctrl + X, Y, Enter`).

10. Build the Docker image for the API.



11. `docker build -t financial-api .`
12. Run the API container.
13. `docker run --rm -p 8080:8080 --name financial-api \`
14. `-v ${PWD}/../test-network:/test-network \`
15. `financial-api`

You should see the log Starting server on port 8080. **Leave this terminal open.** It is now running your live API server.

```

kernel@fatima:~/fabric-internship-project/fabric-samples/test-network$ cd ../financial-api
kernel@fatima:~/fabric-internship-project/fabric-samples/financial-api$ docker build -t financial-api .
[+] Building 3.2s (15/15) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 597B
=> [internal] load metadata for docker.io/library/golang:1.24
=> [internal] load metadata for docker.io/library/alpine:3.18
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [builder 1/6] FROM docker.io/library/golang:1.24@sha256:2c5f7a8c252a17cf6aa30ddee15caa0f485ee29418a6ea64cddb62eea2b07bdf
=> => resolve docker.io/library/golang:1.24@sha256:2c5f7a8c252a17cf6aa30ddee15caa0f485ee29418a6ea64cddb62eea2b07bdf
=> [internal] load build context
=> => transferring context: 111B
=> [stage-1 1/3] FROM docker.io/library/alpine:3.18@sha256:de0eb0b3f2a47ba1eb89389859a9bd88b28e82f5826b6969ad604979713c2d4f
=> => resolve docker.io/library/alpine:3.18@sha256:de0eb0b3f2a47ba1eb89389859a9bd88b28e82f5826b6969ad604979713c2d4f
=> CACHED [stage-1 2/3] RUN apk add --no-cache ca-certificates
=> CACHED [builder 2/6] WORKDIR /app
=> CACHED [builder 3/6] COPY go.mod go.sum ./
=> CACHED [builder 4/6] RUN go mod download
=> CACHED [builder 5/6] COPY . .
=> CACHED [builder 6/6] RUN CGO_ENABLED=0 GOOS=linux go build -o /server
=> CACHED [stage-1 3/3] COPY --from=builder /server /server
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:2057dd615fad2db5d5072fbc4df70e44e2e0c92ef35380bcb6a968f8e3db048f5
=> => exporting config sha256:5f99a6d415f257d179229974fa5c9357100f4198a8b23229a428453e976c015d
=> => exporting attestation manifest sha256:b99f70396707e81e72caf8d318f963d2135d93665cc671c9408716a0d18fb3de
=> => exporting manifest list sha256:cf2cc8a751b99e4ca91c69675791112315124f53cab835fedcc3e983aeecc9e8
=> => naming to docker.io/library/financial-api:latest
=> => unpacking to docker.io/library/financial-api:latest
kernel@fatima:~/fabric-internship-project/fabric-samples/financial-api$ docker run --rm -p 8080:8080 --name financial-api \
-v ${PWD}/../test-network:/test-network \
financial-api
2025/10/05 06:45:40 ===== application-golang starts =====
2025/10/05 06:45:40 Starting server on port 8080
2025/10/05 06:46:13 --> Submit Transaction: CreateAsset, for dealer DEALER001
2025/10/05 06:46:15 --> Evaluate Transaction: ReadAsset, for dealer DEALER001
2025/10/05 06:48:45 --> Evaluate Transaction: ReadAsset, for dealer DEALER001

```

Step 5: Test the Live Application (Terminal 3)

1. Open a **third, new terminal**.
2. Use `curl` to interact with your running API.

Create an Asset

- **Endpoint:** `POST /assets`
- **Command:**
- `curl -X POST http://localhost:8080/assets -d '{`
- `"DEALERID": "DEALER001",`
- `"MSISDN": "9876543210",`
- `"MPIN": "1234",`
- `"BALANCE": 5000,`

- "STATUS": "ACTIVE",
- "TRANSAMOUNT": 5000,
- "TRANSTYPE": "CREDIT",
- "REMARKS": "Initial deposit via API"
- }
- **Success Response:** Asset DEALER001 created successfully

Read the Asset

- **Endpoint:** GET /assets?dealerid=<id>
- **Command:**
- curl http://localhost:8080/assets?dealerid=DEALER001
- **Success Response (JSON):**
- {"DEALERID":"DEALER001","MSISDN":"9876543210","MPIN":"1234","BALANCE":5000,"STATUS":"ACTIVE","TRANSAMOUNT":5000,"TRANSTYPE":"CREDIT","REMARKS":"Initial deposit via API"}

```
kernel@fatima:~$ # Create an asset
curl -X POST http://localhost:8080/assets -d '{
  "DEALERID": "DEALER001",
  "MSISDN": "9876543210",
  "MPIN": "1234",
  "BALANCE": 5000,
  "STATUS": "ACTIVE",
  "TRANSAMOUNT": 5000,
  "TRANSTYPE": "CREDIT",
  "REMARKS": "Initial deposit via API"
}'

# Read the asset
curl http://localhost:8080/assets?dealerid=DEALER001
Asset DEALER001 created successfully{"DEALERID":"DEALER001","MSISDN":"9876543210","MPIN":"1234","BALANCE":5000,"STATUS":"ACTIVE","TRANSAMOUNT":5000,"TRANSTYPE":"CREDIT","REMARKS":"Initial deposit via API"}
kernel@fatima:~$
```

```
kernel@fatima:~$ # Create an asset
curl -X POST http://localhost:8080/assets -d '{
  "DEALERID": "DEALER001",
  "MSISDN": "9876543210",
  "MPIN": "1234",
  "BALANCE": 5000,
  "STATUS": "ACTIVE",
  "TRANSAMOUNT": 5000,
  "TRANSTYPE": "CREDIT",
  "REMARKS": "Initial deposit via API"
}'

# Read the asset
curl http://localhost:8080/assets?dealerid=DEALER001
Asset DEALER001 created successfully{"DEALERID":"DEALER001","MSISDN":"9876543210","MPIN":"1234","BALANCE":5000,"STATUS":"ACTIVE","TRANSAMOUNT":5000,"TRANSTY
kernel@fatima:~$
```

Bringing Down the Application

1. In **Terminal 2** (running the API), press Ctrl + C to stop the API server container.
2. In **Terminal 1** (running the blockchain), run the down script to stop and remove all network components.
3. `./network.sh down`