Experiment No-9

Title: Setup and Configuration of a Private Blockchain Network using Hyperledger Fabric.

Aim: To set up and configure a private blockchain network using **Hyperledger Fabric**, and test transactions between peers.

Theory:

- **Private Blockchain**: A blockchain where access is restricted to selected participants.
- **Hyperledger Fabric**:
    - An open-source, permissioned blockchain framework.
    - Developed by the Linux Foundation.
    - Used for enterprise applications like supply chain, banking, and healthcare.

Key Components of Hyperledger Fabric

1. **Peer Nodes** – Maintain the ledger and run smart contracts (chaincode).
2. **Orderer Node** – Orders transactions and ensures consensus.
3. **Channel** – Private communication path between organizations.
4. **Chaincode** – Smart contracts in Fabric (written in Go, Node.js, or Java).
5. **Certificate Authority (CA)** – Provides digital identities to participants.

Key Characteristics of dApps :

1. **Permissioned** – Only approved members can join.
2. **Modular Architecture** – Consensus, identity, and policies are configurable.
3. **Privacy** – Supports private channels for sensitive data.
4. **Smart Contracts (Chaincode)** – Can be written in common programming languages.
5. **High Performance** – Supports thousands of transactions per second.

Steps of Execution:

1. Install prerequisites (Docker, Docker-Compose, Go, Node.js, Fabric binaries).
2. Download Hyperledger Fabric samples.
3. Generate crypto materials (certificates and keys).
4. Configure network with peers, orderer, and channel.
5. Start the Fabric network using Docker.
6. Deploy a sample chaincode.
7. Test transactions (query and update ledger).

Stepwise Procedure :

1. Install Prerequisites

- Install **Docker** and **Docker-Compose**.
- Install **cURL, Go, and Node.js**.
- Verify installation using:

Program:

docker --version

go version

node -v

2. Download Fabric Samples

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

cd fabric-samples/test-network

3. Generate Certificates and Genesis Block

./network.sh down

./network.sh up

- This generates **crypto materials** and starts the network.

4. Create a Channel

./network.sh createChannel -c mychannel

5. Deploy Chaincode (Smart Contract)

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

6. Interact with Chaincode

- **Query all assets:**

peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'

Create a new asset:

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com \

--tls --cafile $ORDERER_CA -C mychannel -n basic \

-c '{"Args":["CreateAsset","asset6","blue","30","Tom","400"]}'

7. Verify Transactions

- **Query the asset again:**

peer chaincode query -C mychannel -n basic -c '{"Args":["ReadAsset","asset6"]}'

Program (Sample Chaincode in Go – Asset Transfer)

```
package main
import (
    "encoding/json"
    "fmt"
    "github.com/hyperledger/fabric-contract-api-go/contractapi"
)
type SmartContract struct {
    contractapi.Contract
}
type Asset struct {
    ID             string `json:"ID"`
    Color          string `json:"color"`
    Size           int    `json:"size"`
    Owner          string `json:"owner"`
    AppraisedValue int    `json:"appraisedValue"`
}
func (s *SmartContract) CreateAsset(ctx contractapi.TransactionContextInterface, id string,
color string, size int, owner string, value int) error {
    asset := Asset{id, color, size, owner, value}
    assetJSON, _ := json.Marshal(asset)
    return ctx.GetStub().PutState(id, assetJSON)
}
func (s *SmartContract) ReadAsset(ctx contractapi.TransactionContextInterface, id string)
(*Asset, error) {
    assetJSON, err := ctx.GetStub().GetState(id)
    if err != nil || assetJSON == nil {
        return nil, fmt.Errorf("Asset not found")
    }
    var asset Asset
    _ = json.Unmarshal(assetJSON, &asset)
    return &asset, nil
}
func main() {
    chaincode, _ := contractapi.NewChaincode(new(SmartContract))
    chaincode.Start()
}
```

Key Points :

1. Hyperledger Fabric is **modular and permissioned**.
2. Identity management is handled by **Certificate Authority (CA)**.
3. Smart contracts are called **Chain code**.
4. The **Ordered** ensures consensus in the network.
5. Channels provide **private communication** for organizations.

Conclusion :

In this experiment, we successfully set up a private blockchain network using Hyperledger Fabric. We created a channel, deployed chain code, and performed transactions. This demonstrated how Fabric provides a secure and permissioned blockchain system for enterprise applications

Viva Questions:

1. What is the difference between public and private blockchains?
2. What is Hyperledger Fabric used for?
3. What is a peer node in Fabric?
4. What is the role of the Orderer in Hyperledger Fabric?
5. What is Chaincode?
6. Why do we use channels in Fabric?
7. What is the advantage of using Fabric over Ethereum for enterprise solutions?