1. Install Docker Container,Node.js and HyperledgerFabric and Etherum Network

1. **Install Docker:** Go to the Docker website and download the Docker Desktop application for your operating system. Follow the installation instructions provided on the website.
2. **Install Node.js:** Visit the Node.js website and download the installer for your operating system. Follow the installation instructions provided on the website.
3. **Install Hyperledger Fabric:**
   - Download the Hyperledger Fabric samples and binaries from the official GitHub repository: https://github.com/hyperledger/fabric-samples
   - Follow the instructions provided in the repository to set up your Hyperledger Fabric network. This typically involves setting up the prerequisites, generating the cryptographic materials, and starting the network using Docker Compose.

     ```
     git clone https://github.com/hyperledger/fabric-samples.git
     cd fabric-samples
     curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.2.2
     ```
4. **Install Ethereum Network:**
   - You can set up a private Ethereum network using tools like Ganache or Geth.
   - **Ganache:** Download and install Ganache from the Truffle Suite website (https://www.trufflesuite.com/ganache) and follow the installation instructions.
   - **Geth**: Install Geth, the Go implementation of the Ethereum protocol, by following the instructions on the Ethereum website (https://ethereum.org/en/developers/docs/nodes-and-clients/geth/).

2. Create and deploy a blockchain network using Hyperledger Fabric SDK for Java

```
public import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.concurrent.TimeoutException;
import org.hyperledger.fabric.gateway.Contract;
import org.hyperledger.fabric.gateway.ContractException;
import org.hyperledger.fabric.gateway.Gateway;
import org.hyperledger.fabric.gateway.Network;
import org.hyperledger.fabric.gateway.Wallet;
import org.hyperledger.fabric.gateway.Wallets;
class Sample {
public static void main(String[] args) throws IOException {
// Load an existing wallet holding identities used to access the
network.
Path walletDirectory = Paths.get("wallet");
Wallet wallet = Wallets.newFileSystemWallet(walletDirectory);
// Path to a common connection profile describing the network.
Path networkConfigFile = Paths.get("connection.json");
// Configure the gateway connection used to access the network.
Gateway.Builder builder = Gateway.createBuilder()
.identity(wallet, "user1")
.networkConfig(networkConfigFile);
// Create a gateway connection
try (Gateway gateway = builder.connect()) {
// Obtain a smart contract deployed on the network.
Network network = gateway.getNetwork("mychannel");
Contract contract = network.getContract("fabcar");
// Submit transactions that store state to the ledger.
byte[] createCarResult = contract.createTransaction("createCar")
.submit("CAR10", "VW", "Polo", "Grey", "Mary");
lOMoAR cPSD|27947560
20
System.out.println(new String(createCarResult,
StandardCharsets.UTF_8));
// Evaluate transactions that query state from the ledger.
byte[] queryAllCarsResult =
contract.evaluateTransaction("queryAllCars");
System.out.println(new String(queryAllCarsResult,
StandardCharsets.UTF_8));
} catch (ContractException | TimeoutException |
InterruptedException e) {
e.printStackTrace();}}}
```

4. Deploy an asset-transfer app using blockchain within hyperledger fabric

```
'use strict';

const { Contract } = require('fabric-contract-api');

class AssetTransfer extends Contract {

    async InitLedger(ctx) {
        console.info('============= START : Initialize Ledger ===========');
        const assets = [
            {
                ID: 'CAR001',
                Owner: 'Tom',
                Model: 'Toyota Corolla',
            },
            {
                ID: 'CAR002',
                Owner: 'Jerry',
                Model: 'Honda Civic',
            },
        ];

        for (const asset of assets) {
            asset.docType = 'asset';
            await ctx.stub.putState(asset.ID,
Buffer.from(JSON.stringify(asset)));
            console.info(`Asset ${asset.ID} initialized`);
        }
        console.info('============= END : Initialize Ledger ===========');
    }

    async CreateAsset(ctx, id, owner, model) {
        console.info('============= START : CreateAsset ===========');

        const asset = {
            ID: id,
            Owner: owner,
            Model: model,
            docType: 'asset',
        };

        await ctx.stub.putState(id, Buffer.from(JSON.stringify(asset)));
        console.info(`Asset ${id} created`);

        console.info('============= END : CreateAsset ===========');
    }

    async TransferAsset(ctx, id, newOwner) {
        console.info('============= START : TransferAsset ===========');

        const assetBuffer = await ctx.stub.getState(id);
        if (!assetBuffer || assetBuffer.length === 0) {
            throw new Error(`Asset ${id} not found`);
        }
        const asset = JSON.parse(assetBuffer.toString());
        asset.Owner = newOwner;

        await ctx.stub.putState(id, Buffer.from(JSON.stringify(asset)));
        console.info(`Asset ${id} transferred to ${newOwner}`);

        console.info('============= END : TransferAsset ===========');
    }

    async QueryAsset(ctx, id) {
        console.info('============= START : QueryAsset ===========');
        const assetAsBytes = await ctx.stub.getState(id); // get the asset from chaincode state
        if (!assetAsBytes || assetAsBytes.length === 0) {
            throw new Error(`Asset ${id} does not exist`);
        }
        console.log(assetAsBytes.toString());
        console.info('============= END : QueryAsset ===========');
        return assetAsBytes.toString();
    }

}

module.exports = AssetTransfer;
```

5.Build a web app that tracks fitness club rewards using Hyperledger Fabric
And
10 Crete a web app that track fitness club rewards using hyperledger fabric

Step 1: Design the Data Model

Define the data structure for the fitness club rewards system. For example, you might have entities like members, rewards, transactions, etc. Each member may have a wallet to store reward points.

Step 2: Write Chaincode (Smart Contract)

Write chaincode to manage reward transactions. This includes functions for:

- Creating member profiles
- Issuing rewards
- Redeeming rewards
- Querying member balances, transaction history, etc.

Step 3: Set up Hyperledger Fabric Network

Set up a Hyperledger Fabric network with channels, peers, orderers, and a Certificate Authority (CA). Install and instantiate the chaincode on the network.

Step 4: Develop the Web Application

Develop a web application using a framework like Express.js (Node.js) for the backend and React.js for the frontend. You'll need to implement features such as:

- User authentication and authorization
- Displaying member profiles, reward balances, transaction history, etc.
- Issuing and redeeming rewards
- Performing transactions securely using Hyperledger Fabric SDK

Step 5: Interact with the Network

Use Hyperledger Fabric SDK (Node.js SDK) in your backend to interact with the Fabric network. Implement functions to invoke chaincode methods for issuing, redeeming rewards, querying member balances, etc.

Step 6: Implement User Interface

Build a user-friendly interface using React.js for the frontend. Implement features like member registration, login, profile management, reward redemption, etc.

Step 7: Test and Deploy

Test your web application thoroughly to ensure that it functions correctly. Deploy the backend and frontend components to a server or cloud platform. Make sure to secure your application and data properly.

```
const { Gateway, Wallets } = require("fabric-network");
const path = require("path");
const fs = require("fs");

async function queryMemberBalance(memberId) {
  const walletPath = path.resolve(__dirname, "wallet");
  const wallet = await Wallets.newFileSystemWallet(walletPath);
  const gateway = new Gateway();
  const connectionProfile = path.resolve(__dirname, "connection.json");
  const connectionOptions = {
    wallet,
    identity: "user1",
    discovery: { enabled: true, asLocalhost: true },
  };
 try {
    await gateway.connect(connectionProfile, connectionOptions);
    const network = await gateway.getNetwork("mychannel");
    const contract = network.getContract("reward-contract");

    const result = await contract.evaluateTransaction(
      "QueryMemberBalance",
      memberId
    );
    return result.toString();
  } finally {
    gateway.disconnect();}}
```

6.Create a "Hello World" example of a car auction network using the Hyperledger Fabric Node SDK
And
7..To create a "Hello World" example of a car auction network using the Hyperledger Fabric Node SDK and IBM Blockchain Starter Plan

```
"use strict";

const { Contract } = require("fabric-contract-api");

class CarAuction extends Contract {
  async InitLedger(ctx) {
    console.info("============= START : Initialize Ledger ===========");
    const cars = [
      {
        id: "CAR001",
        make: "Toyota",
        model: "Corolla",
        owner: "Tom",
        price: 10000,
      },
      {
        id: "CAR002",
        make: "Honda",
        model: "Civic",
        owner: "Jerry",
        price: 12000,
      },
    ];

    for (const car of cars) {
      car.docType = "car";
      await ctx.stub.putState(car.id, Buffer.from(JSON.stringify(car)));
      console.info(`Car ${car.id} initialized`);
    }
    console.info("============= END : Initialize Ledger ===========");
  }

  async ListCar(ctx, id, make, model, owner, price) {
    console.info("============= START : ListCar ===========");

    const car = {
      id: id,
      make: make,
      model: model,
      owner: owner,
      price: price,
      docType: "car",
    };
    await ctx.stub.putState(id, Buffer.from(JSON.stringify(car)));
    console.info(`Car ${id} listed for auction`);

    console.info("============= END : ListCar ===========");
  }

  async PlaceBid(ctx, carId, bidder, bidPrice) {
    console.info("============= START : PlaceBid ===========");

    const carAsBytes = await ctx.stub.getState(carId);
    if (!carAsBytes || carAsBytes.length === 0) {
      throw new Error(`Car ${carId} not found`);
    }
    const car = JSON.parse(carAsBytes.toString());

    if (bidPrice <= car.price) {
      throw new Error("Bid price must be higher than the current price");
    }

    car.price = bidPrice;
    car.owner = bidder;

    await ctx.stub.putState(carId, Buffer.from(JSON.stringify(car)));
    console.info(`Bid placed for Car ${carId} by ${bidder} at ${bidPrice}`);

    console.info("============= END : PlaceBid ===========");
  }

  async CloseAuction(ctx, carId) {
    console.info("============= START : CloseAuction ===========");

    const carAsBytes = await ctx.stub.getState(carId);
    if (!carAsBytes || carAsBytes.length === 0) {
      throw new Error(`Car ${carId} not found`);
    }
    const car = JSON.parse(carAsBytes.toString());

    console.info(
      `Auction for Car ${carId} closed. Winner: ${car.owner}, Price: ${car.price}`
    );

    console.info("============= END : CloseAuction ===========");
    return car;
  }
}

module.exports = CarAuction;
```

8.Create a Crypto-currency Wallet

```
import java.security.*;
import java.security.spec.ECGenParameterSpec;
public class CryptoWallet {
private PrivateKey privateKey;
private PublicKey publicKey;
public CryptoWallet() {
generateKeyPair();
}
public void generateKeyPair() {
try {
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("EC");
SecureRandom random = SecureRandom.getInstanceStrong();
keyGen.initialize(ecSpec, random);
KeyPair keyPair =
keyGen.generateKeyPair();privateKey =
keyPair.getPrivate();
publicKey = keyPair.getPublic();
} catch (Exception
e) {
e.printStackTrace(
);
}}
public static void main(String[] args) {
CryptoWallet wallet = new
CryptoWallet();
System.out.println("Private Key: " +
wallet.privateKey);System.out.println("Public Key:
" + wallet.publicKey);
}}
```

9. Install ethreum network and hyperledger fabric network

Ethereum Network Installation:

1. Install Ethereum Client (Geth or Parity):
    - You can install Geth (Go Ethereum) or Parity client on your machine.
    - Follow the installation instructions provided by the Ethereum client you choose.
2. Run Ethereum Node:
    - After installation, run an Ethereum node on your local machine or connect to a public testnet like Ropsten, Rinkeby, or Kovan.
    - Configure your node to sync with the Ethereum blockchain network.
3. Install Wallet Software:
    - You may want to install a wallet software such as MetaMask or MyEtherWallet to interact with the Ethereum network and manage your accounts.
4. Smart Contract Development:
    - Develop smart contracts using Solidity, the programming language for Ethereum smart contracts.
    - Compile and deploy your smart contracts to the Ethereum network.

Hyperledger Fabric Network Installation:

1. Prerequisites:
    - Install Docker and Docker Compose.
    - Download the Hyperledger Fabric binaries and Docker images.
2. Set Up Network Configuration:
    - Set up your network configuration by defining organizations, peers, orderers, channels, and chaincode (smart contracts).
    - Define your network topology and consensus mechanism (e.g., Solo, Raft, Kafka).
3. Start the Network:
    - Use Docker Compose to start your Hyperledger Fabric network.
    - Bring up the orderer, peers, and other network components defined in your configuration files.
4. Create and Join Channels:
    - Create channels and let peers join them as per your network design.

    - Define policies and anchor peers.
5. Install and Instantiate Chaincode:
    - Install chaincode onto peers and instantiate it on channels.
    - Use CLI commands or SDKs to manage chaincode lifecycle (install, approve, commit).