upGrad

#LifeKoKaroLift

SOLUTION IMPLEMENTATION PART - 1 (CHAINCODE)

upGrad

Course:

Topic: Solution Implementation Part -1 (Chaincode)

Instructor: Jaywardhan Sawale

RECAP

upGrad

1. Overview of components of Hyperledger Fabric

2. Designing Hyperledger Fabric network

3. Deployment of Fabric network
    - Network deployment
    - Chaincode installation
    - Adding and removing an organisation from the network

4. Use of external certificate authority

upGrad

TODAY'S AGENDA

- Introduction
- Structuring smart contract application
- Libraries used for smart contract development
- Developing smart contracts
- Hands-on exercise

INTRODUCTION

upGrad

- Chaincode, in Hyperledger Fabric, is a piece of code that contains business logic.
- Data stored by a chaincode can be accessed only by that chaincode.
- In some cases, a chaincode can access data of another chaincode based on appropriate permissions.
- Data is stored and accessed in key-value pairs.

CHAINCODE BASICS

upGrad

- Irrespective of language of choice, a chaincode needs to implement chaincode interface.

- This interface comprises methods that are called during transaction executions.
  - Init method is called during chaincode instantiation or upgrade.
  - Invoke method is called to make a transaction proposal.

SMART CONTRACT PACKAGE

upGrad

Each Smart Contract package is, from a node perspective, a NPM module.

- package.json
    - This needs to import the fabric-contract-api and fabric-shim npm modules
    - The 'start' script must be set to fabric-chaincode-node-start - this has to be present for the peer to call the node module to start
    - It is recommended to have a 'start:dev' script that can be used for development (details on this later)
    - A 'main' entry to point to your index.js file contain the exports of the node module

SMART CONTRACT PACKAGE

upGrad

- index.js
    - It is mandatory to have a contracts element exported that is a array of classes.
    - Each of these classes must extend the Contract class from the fabric-contract-api module
    - Optionally, a custom serializer may be defined to control how data is converted for transmission between chaincode, peer and ultimately client applications.

SMART CONTRACT DEPLOYMENT

upGrad

- Deployed in peer using cli command
- A docker image is built for this chaincode, the package.json and code copied and command npm install run executed.
- After installation as part of the bootstrap process, constructor is run. Constructor is used to set names for chaincode and optional setup of other functionalities like error monitoring.
- Chaincode is instantiated. During this process init() function is called.
- *init()* and *update()* functions are abstracted out. They are used for any setup initialization and migrations that might be required.

Node.js LIBRARIES

upGrad

- Two node.js libraries are used for chaincode development:
  - *fabric-contract-api*
    - High-level API to implement chaincode
    - Provides a contract interface
  - *fabric-shim*
    - Low-level API to implement smart contracts
    - Provides a shim interface

fabric-contract-api

upGrad

- This library is used to implement a class that extends contract class and a constructor.
- Each contract instance may have functions as required. These functions of the class are directly invokable from a client.
- The node module must export an array of one or more contract classes in the contracts property. Each of these class must extend the correct type. At runtime each of these will have a single instance created, and will persist for the lifetime of the chaincode container.

Transaction context

upGrad

- First parameter of all the functions is transaction context
- It gives information specific to the transaction that is currently being executed.
- Mainly 2 objects present
  - Stub API used to manage world state (ctx.stub)
  - Client identity (ctx.identity)
- Each contract has create context method that can be overridden

fabric-contract-api

upGrad

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

// Main Chaincode Class
class UserChainCode extends Contract {
    constructor() {
        super('org.upstac.user');
    }

    /* ****** Contract Functions - Starts ***** */
    // This is a basic user defined function used at the time of instantiating the smart contract
    // to print the success message on console
    async instantiate(ctx) {
        console.log('User ChainCode Instantiated');
    }
}
```

fabric-shim

upGrad

- This library has 2 functions that can be implemented in a class:
  - Init
  - Invoke

- This process needs to be started separately.

fabric-shim

upGrad

```
const shim = require('fabric-shim');
const Chaincode = class {
    async Init(stub) {
        await stub.putState(key, Buffer.from(aStringValue));
        return shim.success(Buffer.from('Initialized Successfully!'));
    }
    async Invoke(stub) {
        let oldValue = await stub.getState(key);
        let newValue = oldValue + delta;
        await stub.putState(key, Buffer.from(newValue));
        return shim.success(Buffer.from(newValue.toString()));
    }
};
shim.start(new Chaincode());
```

Working with iterators

upGrad

- The fabric-shim api provides capability to retrieve blocks of information.
  - History of a key
  - Set of keys and their values in a given range
  - Values of rich query (performed when using couchDB)

- APIs for these tasks return a set of data that needs to be iterated upon to retrieve final information

Working with iterators

upGrad

- History
  - getHistoryForKey
- Rich query
  - getQueryResult
  - getQueryResultWithPagination
- Range queries
  - getStateByPartialCompositeKey
  - getStateByPartialCompositeKeyWithPagination
  - getStateByRange
  - getStateByRangeWithPagination

Considerations

upGrad

- Functions should not create random variables or use functions dependent on time zones
- A function should not have multiple read writes happening to same world state key (variable)
  - If multiple times a state key is being updated in function, then caution has to be taken about read and write state of data.
- Global variables
- Simultaneous read and writes

Poll 1 (15 seconds)

Which node.js libraries are used for chaincode development?

WALK-THROUGH OF UPSTAC CHAINCODE

(Code Demonstration)

upGrad

HANDS-ON EXERCISE

- Update UPSTAC application chaincode to delete user data.

In this class, you learnt

upGrad

- Introduction
- Structuring smart contract application
- Libraries used for smart contract development
- Developing smart contracts
- Hands-on exercise

HOMEWORK

upGrad

- Write Asset Transfer Chaincode.

NEXT STEPS

- Fabric SDK
  - Identity creation
  - Interacting with smart contracts
- Express.js
- UPSTAC application API server (a walkthrough)

upGrad

#RahoAmbitious

Thank You!