Understanding Chaincode

Lets look at our first chaincode application and deploy it to our basic-network.

WE'LL COVER THE FOLLOWING

Lets start with a simple chaincode to ensure we understand the basic concepts.

We will use a simple Land Record(landrec) chaincode which will do CRUD operations on basic land records.

We will be using javascript(node) as the language for our chaincode. GO is also supported though.

We will use the `fabric-contract-api` node package to write our chaincode.

How chaincode works

Chaincode application is simple. It stores and retrieves data from underlying world-state databases and can hold any logic. Since couch db is a key-value based db, it is quite simple. Based on the data it changes in the world-state the fabric peer itself generates the transaction with “changed” data.

The `fabric-contract-api` requires our chaincode class to extend the `Contract` class.

- The `async initLedger(ctx)` method is called whenever a chaincode is initialized.
- We can add more methods depending on what functionality we want our chaincode to expose. There are a couple of important methods passed in context object that allow us to store and retrieve data from the key value store.

Look at the sample code below. There are two functions supported by this chaincode. CreateLand and QueryLand. Once this chaincode is deployed, any application can call these methods given that it calls it signed with a valid user's private key whose certificate is issued by our CA. Important lines are highlighted

```
'use strict';

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

class LandRec extends Contract {

    async initLedger(ctx) {
        console.info('============= START : Initialize Ledger ===========');
        // todo
        console.info('============= END : Initialize Ledger ===========');
    }

    async queryLand(ctx, landNumber) {
        const landAsBytes = await ctx.stub.getState(landNumber); // get the land from chainco
        if (!landAsBytes || landAsBytes.length === 0) {
            throw new Error(`${landNumber} does not exist`);
        }
        console.log(landAsBytes.toString());
        return landAsBytes.toString();
    }

    async createLand(ctx, landNumber, address, location, owner) {
        console.info('============= START : Create Land ===========');

        const land = {
            docType: 'land',
            address,
            location,
            owner,
        };

        await ctx.stub.putState(landNumber, Buffer.from(JSON.stringify(land)));
        console.info('============= END : Create Land ===========');
    }
}
```

For more details on this visit: https://fabric-shim.github.io/release-1.4/tutorial-deep-dive-contract-interface.html

After we have written our chaincode, we will:

1. Deploy chaincode on peer(s)
2. Instantiate chaincode on peer(s) and define an endorsement policy.

Both, 1 and 2 are done using a cli tool provided by hyperledger fabric. This cli tool is preinstalled in our fabric-cli container to make environment setup easier for us. The fabric cli container is also based off the official hyperledger fabric tools image.

Lets do this practically to understand better! In the next lesson, we will deploy our chaincode.