Deploy Your Chaincode

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

WE'LL COVER THE FOLLOWING

Folder Structure for `Chaincode` Folder

Please note that `infra-basic-network` has all the stuff we used earlier to bring up a network. We will use the same network now and deploy our landrec chaincode on it.

Now lets look at important files inside the `chaincode` folder.

1. `lib/landrec.js`: This contains the chaincode logic which reads and writes values to key value-based data store (state db). This defines valid transactions on the ledger.

2. `index.js`: This is used to expose the landrec chaincode module.

3. `docker-compose.yml`: This defines the `fabric-tools` container from which we will deploy our chaincode. This container will access the chaincode as mounted volume and will connect to same docker network as our fabric network components.

Environment for Step 2

```
'use strict';

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

class LandRec extends Contract {

    async initLedger(ctx) {
        console.info('============= START : Initialize Ledger ===========');
        const lands = [
            {
                address: '4545 45th St E, Redmond WA, 98052',
                location: '[{lat: 3534.3, lon: 2423}, {lat: 3534.3, lon: 2341}, {lat: 3534.3,
                owner: 'John',
            },
            {
                address: '356 8th St E, Redmond WA, 98052',
                location: '[{lat: 3534.3, lon: 2423}, {lat: 3534.3, lon: 2341}, {lat: 3534.3,
                owner: 'Jane',
            },
            {
                address: '999 156th Ave NE, Redmond WA, 98052',
                location: '[{lat: 3534.3, lon: 2423}, {lat: 3534.3, lon: 2341}, {lat: 3534.3,
                owner: 'Huma',
            },
        ];

        for (let i = 0; i < lands.length; i++) {
            lands[i].docType = 'land';
            await ctx.stub.putState('LAND' + i, Buffer.from(JSON.stringify(lands[i])));
            console.info('Added <--> ', lands[i]);
        }
        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 ===========');
    }
```

```
    async queryAllLands(ctx) {
        const startKey = 'LAND0';
        const endKey = 'LAND999';

        const iterator = await ctx.stub.getStateByRange(startKey, endKey);

        const allResults = [];
        while (true) {
            const res = await iterator.next();

            if (res.value && res.value.value.toString()) {
                console.log(res.value.value.toString('utf8'));

                const Key = res.value.key;
                let Record;
                try {
                    Record = JSON.parse(res.value.value.toString('utf8'));
                } catch (err) {
                    console.log(err);
                    Record = res.value.value.toString('utf8');
                }
                allResults.push({ Key, Record });
            }
            if (res.done) {
                console.log('end of data');
                await iterator.close();
                console.info(allResults);
                return JSON.stringify(allResults);
            }
        }
    }

    async changeLandOwner(ctx, landNumber, newOwner) {
        console.info('============= START : changeLandOwner ===========');

        const landAsBytes = await ctx.stub.getState(landNumber); // get the land from chainco
        if (!landAsBytes || landAsBytes.length === 0) {
            throw new Error(`${landNumber} does not exist`);
        }
        const land = JSON.parse(landAsBytes.toString());
        land.owner = newOwner;

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

}

module.exports = LandRec;
```

Running the Environment

1. Run Fabric Network

- *As done in step 1*

  When you run the above environment, you might see your previous environment running with all docker containers still up - if that is the

environment running with all docker containers still up - if that is the case you can skip this step. If its a new environment, we need to bring up the network again on which we will deploy the chaincode. We will bring up the same network as we did in step 1. This time, you can use a script that will execute all the commands we did in step 1. The script is named `exercise-1.sh` and placed inside the `infra-basic-network` directory. Execute the following to run it:

```
cd /usercode/infra-basic-network && ./exercise-1.sh
```

2. Launch Tools Container

Docker compose command to bring up the tools container.

```
cd /usercode/chaincode && docker-compose up -d cli
```

3. Install chaincode

Here we will use `docker exec` command to execute a command in `cli` container. The command is `peer chaincode install` and takes the name, version, path and language of the chaincode it will install on the peer. Note that the path refers to path of chaincode as mounted on the cli/tools container.

If we had multiple peers on our network, we would need to run the same command for all peers.

For full details on peer chaincode tooling visit: https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/peerchaincode.html

```
docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" \
            -e "CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/cr
       cli peer chaincode install \
            -n landrec \
            -v 1.0 \
            -p "/opt/gopath/src/github.com" \
            -l "node"
```

4. Instantiate chaincode

Here we will use `docker exec` command to execute a command in `cli` container. The command is `peer chaincode instantiate` and it takes the orderer node address, channel name, chaincode name, chaincode version,

orderer node address, channel name, chaincode name, chaincode version,

chaincode language and chaincode initialization args as parameters. These args will be passed to the init method of our chaincode contract.

The last argument here is the `endorsement policy`. Here, the endorsement policy is defined as `OR` between org1 and org2 members. This means, as long as a peer from either org1 or org2 endorses, our transaction proposal will satisfy the endorsement policy.

See the link when you want to understand more about endorsement policies.

Execute the command below to instantiate chaincode (It might take a while).

```
docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" \
            -e "CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/cr
       cli peer chaincode instantiate \
            -o orderer.example.com:7050 \
            -C mychannel \
            -n landrec \
            -l "node" \
            -v 1.0 \
            -c '{"Args":[]}' \
            -P "OR ('Org1MSP.member','Org2MSP.member')"
```

5. Test chaincode invoke

Here we will use `docker exec` command to execute a command in `cli` container. The command is `peer chaincode invoke` and takes the orderer, channel, chaincode name to invoke and a string with function name and arguments to invoke in the chaincode.

This is the cli command to invoke chaincode. In actual systems we would need to invoke chaincode from other system components. The Hyperledger Fabric provides SDKs in different languages to do this. In next step, we will use node sdk to invoke chaincode deployed on fabric network.

```
docker exec -e "CORE_PEER_LOCALMSPID=Org1MSP" \
            -e "CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/cr
       cli peer chaincode invoke \
            -o orderer.example.com:7050 \
            -C mychannel \
            -n landrec \
            -c '{"function":"initLedger","Args":[]}'
```

In the next chapter, we will create a full-fledged client application.