Extending a Blockchain Network

This tutorial demonstrates how a blockchain network is extended by adding a third organization.

Quick Setup

1. Run the quick setup.

```
chaincode> ./quick-setup.sh blue-coin blue-coin 1.0
```

2. Generate initial coins for org1.

```
chaincode> docker exec cli0.org1 peer chaincode invoke \
  -o orderer.example.com:7050 \
  -C mychannel -n blue-coin \
  -c '{"function":"generateInitialCoin","Args":["Org1MSP"]}'
```

   This command is performed to give an initial content to couchdb. When the network is extended to include org3 later, we will verify if the couchdb's of org3 will be able to replicate this content.

3. Confirm that the blue coins are saved by opening a browser and checking the contents of couchdb of peer0.org1.example.com

```
browser> http://localhost:5984/_utils/
```

Extending the Blockchain Network

The steps involved in extending the blockchain network is very simlar to starting a blockchain network.

The steps involve creation of the blockchain artifacts, certificates and private keys for peers, and certificate authority for org3.

It also creates a customized `docker-compose-org3.yml` file based on the `docker-compose-org3-template.yml` file.

Lastly, it starts the necessary docker containers to extend the blockchain network.

1. Generate blockchain artifacts, certificates and private keys, and `docker-compose-org3.yml` file.

```
network> ./generate-and-replace-for-extended-network.sh
```

2. Confirm that the the subfolder `config` now contains `org3.json`.

```
network> ls config
```

3. Confirm that the the subfolder `org3-artifacts/crypto-config` is not empty.

```
network> ls org3-artifacts/crypto-config
```

4. Confirm that there is a `docker-compose-org3.yml` file.

```
network> ls docker-compose-org3.yml
```

5. Extend the blockchain network.

```
network> ./extend-network.sh
```

6. Confirm that the necessary docker containers for org3 are up.

```
network> docker ps
```

The following docker containers should be up:

- Org3
  - Peers
    - peer0.org3.example.com
    - peer1.org3.example.com
  - CouchDB
    - couchdb0.org3
    - couchdb1.org3
  - Certificate Authority
    - ca3.example.com

7. Start the CLI docker containers needed to access the peers of org3.

```
network> ./extend-cli.sh
```

8. Confirm that the necessary docker containers are up.

```
network> docker ps
```

The following additional docker containers should be up:

- Org3
  - CLI
    - cli0.org3
    - cli1.org3

Installing and Upgrading Chaincode

1. Install the chaincode found in the subfolder `blue-coin`.

```
chaincode> ./install-chaincode.sh 1 0 blue-coin blue-coin 2.0
chaincode> ./install-chaincode.sh 1 1 blue-coin blue-coin 2.0
chaincode> ./install-chaincode.sh 2 0 blue-coin blue-coin 2.0
chaincode> ./install-chaincode.sh 2 1 blue-coin blue-coin 2.0
chaincode> ./install-chaincode.sh 3 0 blue-coin blue-coin 2.0
chaincode> ./install-chaincode.sh 3 1 blue-coin blue-coin 2.0
```

Notice that instead of version `1.0`, a new version `2.0` is indicated in the parameter. Even if there are no changes made in the chaincode, a new version number is used since org3 will be added to the existing network.

The script copied the chaincode found in the subfolder `blue-coin` to the following peers:

- peer0.org1.example.com
- peer1.org1.example.com
- peer0.org2.example.com
- peer1.org2.example.com
- peer0.org3.example.com
- peer1.org3.example.com

Each peer refers to this copy as `blue-coin`.

2. Upgrade the chaincode `blue-coin`.

```
chaincode> ./upgrade-chaincode.sh 1 0 blue-coin 2.0
```

The `upgrade-chaincode.sh` script accepts two parameters:

- `org index` - index of organization
- `peer index` - index of peer
- `chaincode name` - name of the chaincode
- `chaincode version` - version of the chaincode

The `org index` and `peer index` determine which peer is used to upgrade the chaincode. For example, if `org index` is 1 and `peer index` is 0 then the chaincode is instantiated through

peer0.org1.example.com.

The `chaincode name` and `chaincode version` should be the same name and version used when the chaincode is installed.

The script uses the CLI of peer0.org1.example.com (i.e., cli0.org1) to upgrade the chaincode.

Take note that the target of the upgrade of a chaincode is a channel and not a particular peer. We just used peer0.org1.example.com to perform the upgrade. However, we could have used any of the other peers where the `blue-coin` chaincode was previously installed.

Note: Upgrade will create an additional docker container for peer0.org1.example.com since we used the CLI for this peer to instantiate the chaincode.

This will take several minutes.

3. Confirm that an additonal docker container for peer0.org1.example.com is created.

```
chaincode> docker ps
```

Notice that a docker container with the following name is created:

- dev-peer0.org1.example.com-blue-coin-2.0

This docker container, which we refer to as a chaincode container, contains the running chaincode `blue-coin` and is owned by the docker container peer0.org1.example.com. The number `2.0` pertains to the version of the chaincode.

Confirm that the Blockchain is Replicated to Org3

1. Confirm that the blue coins are replicated in org3 by opening a browser and checking the contents of couchdb of peer0.org3.example.com and peer1.org3.example.com

```
browser> http://localhost:9984/_utils/
browser> http://localhost:10984/_utils/
```

Test the Chaincode

1. Generate initial coins for org2 and org3.

```
chaincode> docker exec cli0.org2 peer chaincode invoke \
  -o orderer.example.com:7050 \
  -C mychannel -n blue-coin \
  -c '{"function":"generateInitialCoin","Args":["Org2MSP"]}'

chaincode> docker exec cli0.org3 peer chaincode invoke \
  -o orderer.example.com:7050 \
```

```
  -C mychannel -n blue-coin \
  -c '{"function":"generateInitialCoin","Args":["Org3MSP"]}'
```

2. Enroll users for org1 and org3.

```
bc-client> node enrollAdmin.js 1
bc-client> node enrollUser.js 1 1
bc-client> node enrollAdmin.js 3
bc-client> node enrollUser.js 3 1
```

3. Transfer 50 blue coins from org1 to org2.

```
bc-client> node transferCoinWithBlueCoinManager.js 1 1 \
  Org1MSP Org2MSP 50
```

4. Transfer 150 blue coins from org3 to org2.

```
bc-client> node transferCoinWithBlueCoinManager.js 3 1 \
  Org3MSP Org2MSP 150
```

Simultaneous Transfer

1. Open a second blue coin client terminal.

2. For the first terminal, type the following command. DO NOT press enter yet.

```
bc-client #1> node transferCoinWithBlueCoinManager.js 1 1 \
  Org1MSP Org2MSP 5
```

   This will transfer blue coins from org1 to org2.

3. For the second terminal, type the following command. DO NOT press enter yet.

```
bc-client #2> node transferCoinWithBlueCoinManager.js 3 1 \
  Org3MSP Org2MSP 5
```

   This will transfer blue coins from org2 to org1.

4. Press enter on both terminals.

   IMPORTANT: Try to make the gap between the two presses as short as possible (e.g., less than 1 second).

This will make the two function calls be part of the same block.

Notice that only one of them is successful while the other one has an `MVCC_READ_CONFLICT` status.

The reason for this is both function calls perform an update on the same world state keys (i.e., `Org2MSP`).

Quick Extended Setup

Similar to `quick-setup.sh`, a similar script is created to extend the network called `quick-extended-setup.sh`.

Take note that `quick-extended-setup.sh` should be run only if there is an existing blockchian network. In a typical scenario, `quick-setup.sh` is executed followed by `quick-extended-setup.sh`.

1. Run the quick setup.

```
chaincode> ./quick-setup.sh blue-coin blue-coin 1.0
```

2. Run the quick extended setup.

```
chaincode> ./quick-extended-setup.sh blue-coin blue-coin 2.0
```

The `quick-extended-setup.sh` script accepts three parameters:

- `folder name` - location of the chaincode
- `chaincode name` - name of the chaincode
- `chaincode version` - version of the chaincode

Notice that the version passed in `quick-extended-setup.sh` is 2.0.