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

**Expected Output:**
```

Quick setup for chaincode blue-coin is complete.

```
```

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"]}'
```

Expected Output:

```
... UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke
successful. result: status:200 payload:"
{\"status\":200,\"message\":\"Successfully generated blue
coins\",\"payload\":{\"mspId\":\"Org1MSP\",\"amt\":500}}"
```

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/
```

Expected Output:

```
_replicator
_users
mychannel_
mychannel_blue-coin
mychannel_lscc
```

4. In the web management of couchdb, click `mychannel_blue-coin` and click the entry with the key `Org1MSP`.

   Expected Output:

```
{
"_id": "Org1MSP",
"_rev": "<may differ>",
"amt": 500,
"mspId": "Org1MSP",
"~version": "<may differ>"
}
```

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
```

   Expected Output:

```
Artifacts, certificates, private keys, and docker-compose-org3.yml file are
generated.
```

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

```
network> ls config
```

Expected Output:

```
channel.tx  genesis.block  Org1MSPanchors.tx  Org2MSPanchors.tx  org3.json
```

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

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

Expected Output:

```
ordererOrganizations  peerOrganizations
```

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

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

Expected Output:

```
docker-compose-org3.yml
```

5. Extend the blockchain network.

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

Expected Output:

```
Blockchain network is extended.
```

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

```
network> docker ps --format "table {{.Ports}}\t{{.Names}}"
```

Expected Output:

```
PORTS                                              NAMES
0.0.0.0:11051->7051/tcp, 0.0.0.0:11053->7053/tcp   peer0.org3.example.com
0.0.0.0:12051->7051/tcp, 0.0.0.0:12053->7053/tcp   peer1.org3.example.com
0.0.0.0:9054->7054/tcp                             ca3.example.com
4369/tcp, 9100/tcp, 0.0.0.0:9984->5984/tcp         couchdb0.org3
4369/tcp, 9100/tcp, 0.0.0.0:10984->5984/tcp        couchdb1.org3
                                                  dev-peer0.org1.example.com-
blue-coin-1.0
                                                  cli1.org2
                                                  cli0.org2
                                                  cli1.org1
                                                  cli0.org1
0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp     peer1.org1.example.com
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp     peer0.org1.example.com
0.0.0.0:10051->7051/tcp, 0.0.0.0:10053->7053/tcp   peer1.org2.example.com
0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp     peer0.org2.example.com
0.0.0.0:8054->7054/tcp                             ca2.example.com
4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp         couchdb1.org1
4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp         couchdb0.org1
0.0.0.0:7050->7050/tcp                             orderer.example.com
0.0.0.0:7054->7054/tcp                             ca1.example.com
4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp         couchdb1.org2
4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp         couchdb0.org2
```

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
```

Expected Output:

```
:
:
Creating cli1.org3 ... done
CLI for peers are up.
```

8. Confirm that the necessary docker containers are up.

```
network> docker ps --format "table {{.Ports}}\t{{.Names}}"
```

Expected Output:

```
PORTS                                              NAMES
                                                 cli1.org3
                                                 cli0.org3
0.0.0.0:11051->7051/tcp, 0.0.0.0:11053->7053/tcp   peer0.org3.example.com
0.0.0.0:12051->7051/tcp, 0.0.0.0:12053->7053/tcp   peer1.org3.example.com
0.0.0.0:9054->7054/tcp                             ca3.example.com
4369/tcp, 9100/tcp, 0.0.0.0:9984->5984/tcp         couchdb0.org3
4369/tcp, 9100/tcp, 0.0.0.0:10984->5984/tcp        couchdb1.org3
                                                 dev-peer0.org1.example.com-
blue-coin-1.0
                                                 cli1.org2
                                                 cli0.org2
                                                 cli1.org1
                                                 cli0.org1
0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp     peer1.org1.example.com
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp     peer0.org1.example.com
0.0.0.0:10051->7051/tcp, 0.0.0.0:10053->7053/tcp   peer1.org2.example.com
0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp     peer0.org2.example.com
0.0.0.0:8054->7054/tcp                             ca2.example.com
4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp         couchdb1.org1
4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp         couchdb0.org1
0.0.0.0:7050->7050/tcp                             orderer.example.com
0.0.0.0:7054->7054/tcp                             ca1.example.com
4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp         couchdb1.org2
4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp         couchdb0.org2
```

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
```

Expected Output:

```
:
:
Installation of chaincode blue-coin 2.0 TO peer0 of org1 is complete.
```

```
:
:
Installation of chaincode blue-coin 2.0 TO peer1 of org1 is complete.
```

```
:
:
Installation of chaincode blue-coin 2.0 TO peer0 of org2 is complete.
```

```
:
:
Installation of chaincode blue-coin 2.0 TO peer1 of org2 is complete.
```

```
:
:
Installation of chaincode blue-coin 2.0 TO peer0 of org3 is complete.
```

```
:
:
Installation of chaincode blue-coin 2.0 TO peer1 of org3 is complete.
```

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
```

Expected Output:

```
:
:
Upgrade of chaincode blue-coin 2.0 TO blue-coin is complete.
```

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 --format "table {{.Ports}}\t{{.Names}}"
```

Expected Output:

```
PORTS                                              NAMES
                                                   dev-peer0.org1.example.com-
blue-coin-2.0
                                                   cli1.org3
                                                   cli0.org3
0.0.0.0:11051->7051/tcp, 0.0.0.0:11053->7053/tcp   peer0.org3.example.com
0.0.0.0:12051->7051/tcp, 0.0.0.0:12053->7053/tcp   peer1.org3.example.com
0.0.0.0:9054->7054/tcp                             ca3.example.com
4369/tcp, 9100/tcp, 0.0.0.0:9984->5984/tcp         couchdb0.org3
4369/tcp, 9100/tcp, 0.0.0.0:10984->5984/tcp        couchdb1.org3
                                                   dev-peer0.org1.example.com-
blue-coin-1.0
                                                   cli1.org2
                                                   cli0.org2
                                                   cli1.org1
                                                   cli0.org1
0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp     peer1.org1.example.com
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp     peer0.org1.example.com
0.0.0.0:10051->7051/tcp, 0.0.0.0:10053->7053/tcp   peer1.org2.example.com
0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp     peer0.org2.example.com
0.0.0.0:8054->7054/tcp                             ca2.example.com
4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp         couchdb1.org1
4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp         couchdb0.org1
0.0.0.0:7050->7050/tcp                             orderer.example.com
0.0.0.0:7054->7054/tcp                             ca1.example.com
4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp         couchdb1.org2
4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp         couchdb0.org2
```

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/
```

Expected Output:

```
{
"_id": "Org1MSP",
```

```
"_rev": "<may differ>",
"amt": 500,
"mspId": "Org1MSP",
"~version": "<may differ>"
}
```

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"]}'
```

Expected Output:

```
... UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke
successful. result: status:200 payload:"
{\"status\":200,\"message\":\"Successfully generated blue
coins\",\"payload\":{\"mspId\":\"Org2MSP\",\"amt\":500}}"
```

```
... UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke
successful. result: status:200 payload:"
{\"status\":200,\"message\":\"Successfully generated blue
coins\",\"payload\":{\"mspId\":\"Org3MSP\",\"amt\":500}}"
```

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
```

Expected Output:

```
Enrolling Administrator admin of org1...
Administrator admin of org1 enrolled successfully.
```

```
Registering User user1... of org1
User user1 of org1 registered successfully.
Enrolling User user1... of org1
User user1 of org1 enrolled successfully.
```

```
Enrolling Administrator admin of org3...
Administrator admin of org3 enrolled successfully.
```

```
Registering User user1... of org3
User user1 of org3 registered successfully.
Enrolling User user1... of org3
User user1 of org3 enrolled successfully.
```

3. Transfer 50 blue coins from org1 to org2.

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

Expected Output:

```
response: {
    "status": 200,
    "message": "Transferred successfully the amount of 50 blue coins from
Org1MSP to Org2MSP",
    "payload": {}
}
transferCoin invocation successful.
Invoked by user1 of org1.
txId: d31a073907c3942ad98c5868890b1a0427df6698db04bbff6da0507e2fcc2b51
status: VALID  blockNo: 7
```

4. Transfer 150 blue coins from org3 to org2.

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

Expected Output:

```
response: {
    "status": 200,
    "message": "Transferred successfully the amount of 150 blue coins from
Org3MSP to Org2MSP",
    "payload": {}
}
transferCoin invocation successful.
Invoked by user1 of org3.
txId: ca817ff8650a0d8a2f22b11a3b9c7ed38a6cb9e1fc86f80e2bcbe4edfe40a830
status: VALID  blockNo: 8
```

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).

   Expected Output:

   ```
   transferCoin invocation successful.
   Invoked by user1 of org1.
   txId: 67787a86177ac2aa10684d84f87134833edd1b094ab86cb8e66cb5c3f741a737
   status: MVCC_READ_CONFLICT  blockNo: 12
   response: {
       "status": 500,
       "message": "Error: Peer localhost:7051 has rejected transaction
   \"67787a86177ac2aa10684d84f87134833edd1b094ab86cb8e66cb5c3f741a737\"
   with code \"MVCC_READ_CONFLICT\""
   }
   ```

```
transferCoin invocation successful.
Invoked by user1 of org3.
txId: d8614c7d586c4cc03e39130707095f67e71a3e0d713daa56817f23263df8cb17
status: VALID  blockNo: 12
response: {
    "status": 200,
    "message": "Transferred successfully the amount of 5 blue coins from
Org3MSP to Org2MSP",
    "payload": {}
}
```

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
```

Expected Output:

```
Quick setup for chaincode blue-coin is complete.
```

2. Run the quick extended setup.

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

Expected Output:

```
Quick extended setup for chaincode $CHAINCODE_NAME is complete.
```

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.