

# Starting a Blockchain Network and Deploying a Chaincode

---

Open the **network**, **wallet**, **chaincode**, and **manager** terminals.

We will be using **Git Bash** terminal to run several scripts.

The scripts are located in different subfolders.

Instead of moving from one subfolder to another, we will just open several terminals, each one is dedicated to run scripts in a particular subfolder.

1. Open a **Git Bash** terminal.
2. Go to the root directory of the training workspace.

```
> cd /c/blockchain-training
```

**Note:** Command may differ depending where you placed your training workspace.

3. Go to the **blockchain-tutorial** subfolder.

```
> cd blockchain-tutorial
```

4. Go to the **network** subfolder.

```
> cd network
```

In this tutorial, this will be referred to the **network** terminal.

You will know that a command should be executed in the **network** terminal when you see the following in the instruction:

```
network>
```

5. Open the terminals for the other subfolders. To do this, follow the steps above. However, instead of going to the **network** subfolder, go to the respective subfolders of each terminal.

Please refer to the information below regarding the details of each terminal.

Terminal Name	Command	Prompt
---------------	---------	--------

---

Terminal Name	Command	Prompt
network	<code>cd network</code>	network>
wallet	<code>cd wallet</code>	wallet>
chaincode	<code>cd chaincode</code>	chaincode>
user manager	<code>cd manager/user</code>	user-mgr>
transaction manager	<code>cd manager/transaction</code>	tx-mgr>
blue coin manager	<code>cd manager/blue-coin</code>	bc-mgr>
blue coin REST API server	<code>cd rest-api-server/blue-coin</code>	bc-rest>

## Clear the Blockchain Setup

Just in case you have run this tutorial before, this will remove any running docker containers and files related to the tutorial.

1. Ensure that there are no existing blockchain network by stopping all docker containers.

```
network> ./stop-network.sh
```

The `stop-network.sh` script stops all docker containers. In addition, it removes all blockchain artifacts, certificates and private keys.

2. Confirm that there are no running docker containers.

```
network> docker ps
```

3. Confirm that the the subfolder `config`, which will contain the blockchain artifacts, is empty.

```
network> ls config
```

4. Confirm that the the subfolder `crypto-config`, which will contain the certificates and private keys, is empty.

```
network> ls crypto-config
```

5. Ensure that there are no wallet contents.

```
wallet> ./clear-wallet.sh
```

6. Confirm that the `wallet` subfolder contains no subfolder like `org1`, `org2`, etc.

```
wallet> ls
```

## Starting the Blockchain Setup

The setup involves creation of the blockchain artifacts, certificates and private keys for peers, orderer, and certificate authority.

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

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

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

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

2. Confirm that the the subfolders `config` and `crypto-config` are not anymore empty.

```
network> ls config  
network> ls crypto-config
```

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

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

4. Start the blockchain network.

```
network> ./start-network.sh
```

5. Confirm that the necessary docker containers are up.

```
network> docker ps
```

The following docker containers should be up:

- Orderer
  - orderer.example.com
- Org1

- Peers
  - peer0.org1.example.com
  - peer1.org1.example.com
- CouchDB
  - couchdb0.org1
  - couchdb1.org1
- Certificate Authority
  - ca1.example.com
- Org2
  - Peers
    - peer0.org2.example.com
    - peer1.org2.example.com
  - CouchDB
    - couchdb0.org2
    - couchdb1.org2
  - Certificate Authority
    - ca2.example.com

6. Start the CLI docker containers needed to access the peers.

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

7. Confirm that the necessary docker containers are up.

```
network> docker ps
```

The following additional docker containers should be up:

- Org1
  - CLI
    - cli0.org1
    - cli1.org1
- Org2
  - CLI
    - cli0.org2
    - cli1.org2

## Install and Instantiate Chaincode

1. Check the subfolders under the `chaincode` subfolder.

```
chaincode> ls
```

Notice that there are subfolders:

- `blue-coin-no-acl`
- `blue-coin`
- `blue-coin-utxo`

Each of this subfolder contains a chaincode. Although all of these three (3) chaincodes demonstrate the implementation of cryptocurrency in a Hyperledger Fabric blockchain, the three have their differences.

2. Install the chaincode found in the subfolder `blue-coin-no-acl`.

```
chaincode> ./install-chaincode.sh blue-coin-no-acl blue-coin
```

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

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

The `folder name` is one of the three chaincode subfolders mentioned in the previous step.

The `chaincode name` can be any name. In this tutorial, we use the name `blue-coin`. You may choose any other name, however, you need to adjust some of the scripts since they are configured to use the name `blue-coin`.

The script copied the chaincode found in the subfolder `blue-coin-no-acl` to the following peers:

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

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

3. Check that there are no additional docker containers created yet.

```
chaincode> docker ps
```

The installation of the chaincode will eventually create an additional docker container for each peer installed with the chaincode.

An additional container for a peer is created once the following happens:

- chaincode gets instantiated
- CLI of peer is used to instantiate, invoke, or query a chaincode

4. Instantiate the chaincode `blue-coin`.

```
chaincode> ./instantiate-chaincode.sh blue-coin
```

The `instantiate-chaincode.sh` script accepts a parameter:

- `chaincode name` - name of the chaincode

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

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

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

**Note:** Instantiation 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.

5. Confirm that an additional 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-1.0`

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 `1.0` pertains to the version of the chaincode.

You may treat this newly created chaincode container as an extension of the docker container `peer0.org1.example.com`. This is used everytime the said peer needs to execute or query the chaincode `blue-coin`.

Note that there are no chaincode containers created for the other peers. As mentioned in the previous steps, an additional docker container is created once the chaincode is instantiated (which we have done already) and once an `instantiate`, `execute`, or `query` is done through a peer. Since `peer0.org1.example.com` is used to do the instantiation, a chaincode container is created for this peer.

The other peers will have its own chaincode container once an `execute` or `query` is done to the chaincode. This will be shown in the succeeding steps.

## Test the Chaincode

1. 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 is an `invoke` command that invokes the `generateInitialCoin` function.

This provides `500` blue coins to `org1`.

Although the org names in the tutorial are `org1`, `org2`, etc., we use as their identifier their membership service provider (MSP) ID (e.g., `Org1MSP`, `Org2MSP`).

2. 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/
```

3. In the web management of couchdb, click `mychannel-blue-coin` and click the entry with the key `Org1MSP`.

Notice that the following value is saved:

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

4. Confirm that the blue coin of `org1` is replicated in the couchdb of the other peers.

Follow the previous steps but change the port in the URL from `5984` to the following:

- `6984` - for `peer1.org1.example.com`
- `7984` - for `peer0.org2.example.com`
- `8984` - for `peer1.org2.example.com`

5. Query the balance of `org1`.

```
chaincode> docker exec cli0.org1 peer chaincode query -C mychannel -n blue-coin -c '{"function":"getBalance","Args":["Org1MSP"]}'
```

This is a `query` command that queries using the `getBalance` function.

Notice that it shows the same blue coins shown in the previous steps.

6. Query again the balance of `org1` but this time using the CLI of `peer1.org1.example.com`.

```
chaincode> docker exec cli1.org1 peer chaincode query -C mychannel -n blue-coin -c '{"function":"getBalance","Args":["Org1MSP"]}'
```

Since this is the first time that an invoke or query command is used in the CLI of peer.org1.example.com, it will take some time before a response is shown. This is due to a chaincode container for the said peer is being created.

7. Confirm that an additional docker container for peer1.org1.example.com is created.

```
chaincode> docker ps
```

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

- dev-peer1.org1.example.com-blue-coin-1.0

8. Query again the balance of org1 but this time using the CLIs of peer0.org2.example.com and peer1.org2.example.com.

```
chaincode> docker exec cli0.org2 peer chaincode query -C mychannel -n blue-coin -c '{"function":"getBalance","Args":["Org1MSP"]}'
```

```
chaincode> docker exec cli1.org2 peer chaincode query -C mychannel -n blue-coin -c '{"function":"getBalance","Args":["Org1MSP"]}'
```

9. Confirm that additional docker containers for peer0.org2.example.com and peer1.org2.example.com are created.

```
chaincode> docker ps
```

Notice that docker containers with the following names are created:

- dev-peer0.org2.example.com-blue-coin-1.0
- dev-peer1.org2.example.com-blue-coin-1.0

## Quick Setup

Since clearing of blockchain setup, starting blockchain setup, and installing and instantiating the chaincode are done frequently in this tutorial, a [quick-setup.sh](#) script is created that performs all the said processes.

1. Run the quick setup.

```
chaincode> ./quick-setup.sh
```