Starting a Blockchain Network and Deploying a Chaincode

This tutorial demonstrates how a blockchain network is setup.

In addition, it shows how a chaincode is deployed.

Open the network, wallet, chaincode, client, and REST API server 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 as a `network` terminal.

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

   ```
   network>
   ```
5. Throughout this training, you need to 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.

Note that you DO NOT need to open NOW all the terminals listed below. You can just open the terminals as needed.

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

Expected Output:

```
:
:
Blockchain network is stopped.
```

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
```

Expected Output:

```
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

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

```
network> ls config
```

Expected Output:

```

```

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

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

Expected Output:

```

```

5. Ensure that there are no wallet contents.

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

Expected Output:

```
Wallet is cleared.
```

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

```
wallet> ls
```

Expected Output:

```
clear-wallet.sh
```

Starting the Blockchain Setup

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

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
```

Expected Output:

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

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

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

Expected Output:

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

```
ordererOrganizations  peerOrganizations
```

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

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

Expected Output:

```
docker-compose.yml
```

4. Start the blockchain network.

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

Expected Output:

```
:
:
Blockchain network is started.
```

5. Confirm that the necessary docker containers are up.

```
network> docker ps
```

Expected Output:

```
CONTAINER ID        IMAGE                         COMMAND
CREATED              STATUS              PORTS
NAMES
6c3c1f8ef883        hyperledger/fabric-peer      "peer node start"
About a minute ago   Up About a minute   0.0.0.0:10051->7051/tcp,
0.0.0.0:10053->7053/tcp   peer1.org2.example.com
a5a027b16ab2        hyperledger/fabric-peer      "peer node start"
About a minute ago   Up About a minute   0.0.0.0:7051->7051/tcp,
0.0.0.0:7053->7053/tcp     peer0.org1.example.com
1e69ae7ca977        hyperledger/fabric-peer      "peer node start"
About a minute ago   Up About a minute   0.0.0.0:9051->7051/tcp,
0.0.0.0:9053->7053/tcp     peer0.org2.example.com
aab89fa9bef2        hyperledger/fabric-peer      "peer node start"
About a minute ago   Up About a minute   0.0.0.0:8051->7051/tcp,
0.0.0.0:8053->7053/tcp     peer1.org1.example.com
bd5be3ab6456        hyperledger/fabric-couchdb   "tini -- /docker-ent…"
About a minute ago   Up About a minute   4369/tcp, 9100/tcp, 0.0.0.0:8984-
>5984/tcp         couchdb1.org2
673fad5128f8        hyperledger/fabric-orderer   "orderer"
About a minute ago   Up About a minute   0.0.0.0:7050->7050/tcp
orderer.example.com
22c71d61c365        hyperledger/fabric-ca        "sh -c 'fabric-ca-se…"
About a minute ago   Up About a minute   0.0.0.0:7054->7054/tcp
ca1.example.com
d68da73af49b        hyperledger/fabric-ca        "sh -c 'fabric-ca-se…"
About a minute ago   Up About a minute   0.0.0.0:8054->7054/tcp
ca2.example.com
57afeeff4a4f        hyperledger/fabric-couchdb   "tini -- /docker-ent…"
About a minute ago   Up About a minute   4369/tcp, 9100/tcp, 0.0.0.0:7984-
>5984/tcp         couchdb0.org2
```

```
b11df4aeb9ec        hyperledger/fabric-couchdb   "tini -- /docker-ent…"
About a minute ago   Up About a minute   4369/tcp, 9100/tcp, 0.0.0.0:5984-
>5984/tcp         couchdb0.org1
a97d23066f22        hyperledger/fabric-couchdb   "tini -- /docker-ent…"
About a minute ago   Up About a minute   4369/tcp, 9100/tcp, 0.0.0.0:6984-
>5984/tcp         couchdb1.org1
```

Depending on the width of your `Git Bash` terminal, the output may be cluttered due to the length of information displayed.

6. Limit the information displayed by the docker command to make the output more readable.

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

Expected Output:

```
PORTS                                              NAMES
0.0.0.0:10051->7051/tcp, 0.0.0.0:10053->7053/tcp   peer1.org2.example.com
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp     peer0.org1.example.com
0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp     peer0.org2.example.com
0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp     peer1.org1.example.com
4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp         couchdb1.org2
0.0.0.0:7050->7050/tcp                             orderer.example.com
0.0.0.0:7054->7054/tcp                             ca1.example.com
0.0.0.0:8054->7054/tcp                             ca2.example.com
4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp         couchdb0.org2
4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp         couchdb0.org1
4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp         couchdb1.org1
```

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

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

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

Expected Output:

```
:
:
Creating cli0.org1 ... done
Creating cli1.org1 ... done
Creating cli0.org2 ... done
Creating cli1.org2 ... 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.org2
                                                 cli0.org2
                                                 cli1.org1
                                                 cli0.org1
0.0.0.0:10051->7051/tcp, 0.0.0.0:10053->7053/tcp   peer1.org2.example.com
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp     peer0.org1.example.com
0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp     peer0.org2.example.com
0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp     peer1.org1.example.com
4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp         couchdb1.org2
0.0.0.0:7050->7050/tcp                             orderer.example.com
0.0.0.0:7054->7054/tcp                             ca1.example.com
0.0.0.0:8054->7054/tcp                             ca2.example.com
4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp         couchdb0.org2
4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp         couchdb0.org1
4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp         couchdb1.org1
```

The following additional docker containers should be up:

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

Installing and Instantiating Chaincode

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

```
chaincode> ls
```

Expected Output:

```
blue-coin         blue-coin-utxo  install-chaincode.sh      quick-extended-
setup.sh  upgrade-chaincode.sh
blue-coin-no-acl  hyperledger     instantiate-chaincode.sh  quick-setup.sh
view-chaincode-logs.sh
```

Notice that there are three (3) 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 Hyerledger Fabric blockchain, the three have their differences.

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

```
chaincode> ./install-chaincode.sh 1 0 blue-coin-no-acl blue-coin 1.0
chaincode> ./install-chaincode.sh 1 1 blue-coin-no-acl blue-coin 1.0
chaincode> ./install-chaincode.sh 2 0 blue-coin-no-acl blue-coin 1.0
chaincode> ./install-chaincode.sh 2 1 blue-coin-no-acl blue-coin 1.0
```

Expected Output:

```
:
:
Installation of chaincode blue-coin 1.0 TO peer0 of org1 is complete.
```

```
:
:
Installation of chaincode blue-coin 1.0 TO peer1 of org1 is complete.
```

```
:
:
Installation of chaincode blue-coin 1.0 TO peer0 of org2 is complete.
```

```
:
:
Installation of chaincode blue-coin 1.0 TO peer1 of org2 is complete.
```

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

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

The `org index` and `peer index` determine in which docker container a chaincode is installed. For example, if `org index` is 1 and `peer index` is 0 then the chaincode is installed in peer0.org1.example.com.

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 `chaincode version` usually starts with 1.0. This is increased to a higher number when a chaincode is upgraded.

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

Expected Output:

```
PORTS                                              NAMES
                                                 cli1.org2
                                                 cli0.org2
                                                 cli1.org1
                                                 cli0.org1
0.0.0.0:10051->7051/tcp, 0.0.0.0:10053->7053/tcp   peer1.org2.example.com
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp     peer0.org1.example.com
0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp     peer0.org2.example.com
0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp     peer1.org1.example.com
4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp         couchdb1.org2
0.0.0.0:7050->7050/tcp                             orderer.example.com
0.0.0.0:7054->7054/tcp                             ca1.example.com
0.0.0.0:8054->7054/tcp                             ca2.example.com
4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp         couchdb0.org2
4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp         couchdb0.org1
4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp         couchdb1.org1
```

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

An additional container for a peer will be 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 1 0 blue-coin 1.0
```

Expected Output:

```
:
:
Instantiation of chaincode blue-coin 1.0 TO blue-coin is complete.
```

The `instantiate-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 instantiate 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 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: Instantation 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 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-1.0
                                                   cli1.org2
                                                   cli0.org2
                                                   cli1.org1
                                                   cli0.org1
0.0.0.0:10051->7051/tcp, 0.0.0.0:10053->7053/tcp   peer1.org2.example.com
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp     peer0.org1.example.com
0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp     peer0.org2.example.com
0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp     peer1.org1.example.com
4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp         couchdb1.org2
0.0.0.0:7050->7050/tcp                             orderer.example.com
0.0.0.0:7054->7054/tcp                             ca1.example.com
0.0.0.0:8054->7054/tcp                             ca2.example.com
4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp         couchdb0.org2
4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp         couchdb0.org1
4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp         couchdb1.org1
```

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 additional 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.og1.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"]}'
```

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 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/
```

Expected Output:

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

3. 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>"
}
```

4. Confirm that the blue coins 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 (`http://localhost:6984/_utils/`)
   - `7984` - for peer0.org2.example.com (`http://localhost:7984/_utils/`)
   - `8984` - for peer1.org2.example.com (`http://localhost:8984/_utils/`)

5. Query the balance of org1.

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

   Expected Output:

```
{"status":200,"message":"Balance retrieved successfully","payload":
{"amt":500,"mspId":"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"]}'
```

Expected Output:

```
{"status":200,"message":"Balance retrieved successfully","payload":
{"amt":500,"mspId":"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 additonal docker container for peer1.org1.example.com is created.

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

Expected Output:

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

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"]}'
```

Expected Output:

```
{"status":200,"message":"Balance retrieved successfully","payload":
{"amt":500,"mspId":"Org1MSP"}}
```

```
{"status":200,"message":"Balance retrieved successfully","payload":
{"amt":500,"mspId":"Org1MSP"}}
```

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

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

Expected Output:

```
PORTS                                              NAMES
                                                   dev-peer1.org2.example.com-
blue-coin-1.0
                                                   dev-peer0.org2.example.com-
blue-coin-1.0
                                                   dev-peer1.org1.example.com-
blue-coin-1.0
                                                   dev-peer0.org1.example.com-
blue-coin-1.0
                                                   cli1.org2
                                                   cli0.org2
                                                   cli1.org1
                                                   cli0.org1
0.0.0.0:10051->7051/tcp, 0.0.0.0:10053->7053/tcp   peer1.org2.example.com
0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp     peer0.org1.example.com
0.0.0.0:9051->7051/tcp, 0.0.0.0:9053->7053/tcp     peer0.org2.example.com
0.0.0.0:8051->7051/tcp, 0.0.0.0:8053->7053/tcp     peer1.org1.example.com
```

```
4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp         couchdb1.org2
0.0.0.0:7050->7050/tcp                             orderer.example.com
0.0.0.0:7054->7054/tcp                             ca1.example.com
0.0.0.0:8054->7054/tcp                             ca2.example.com
4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp         couchdb0.org2
4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp         couchdb0.org1
4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp         couchdb1.org1
```

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 blue-coin-no-acl blue-coin 1.0
```

Expected Output:

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

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

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