Utilize NodeJS Client SDK

In the previous tutorial, the chaincode functions are called through the CLI of the peers.

Using the CLI is a quick way of testing the functions in a chaincode.

However, if the chaincode functions need to be called inside another program, the CLI approach becomes impractical.

This tutorial demonstrates the use of a Hyperledger Fabric NodeJS Client SDK that allows a NodeJS program to call the chaincode functions.

Take note that the chaincode itself is written in NodeJS. It is just a coincidence that we will be using a NodeJS program, which will serve as a chaincode client, to call the chaincode functions.

In a different situation, the chaincode could have been written in another programming language (e.g., Golang) while the chaincode client is written in NodeJS.

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.
   ```

   Notice that the chaincode we are using is found in the `blue-coin` subfolder and not the one found in `blue-coin-no-acl`.

   These two versions of the blue coin chaincode are basically the same except that the one in the `blue-coin` subfolder uses access control list (ACL). This means that call to most of the functions in the chaincode is limited.

   For example, the `transferCoin` function can be called only by the organization that owns the blue coins that will be transferred.

Examine the Client Folder

1. List the files of the `client/blue-coin` subfolder.

   ```
   bc-client> ls
   ```

   The folder contains several NodeJS programs. They are listed below and grouped according to their function:

   - Enroll Administrator
     - `enrollAdmin.js`
     - `enrollAdminWithUserManager.js`
   - Enroll User
     - `enrollUser.js`
     - `enrollUserWithUserManager.js`
   - Generate Initial Coin
     - `generateInitialCoin.js`
     - `generateInitialCoinWithTransactionManager.js`
     - `generateInitialCoinWithBlueCoinManager.js`

- Get Balance
  - `getBalanceWithBlueCoinManager.js`
- Transfer Coin
  - `transferCoinWithBlueCoinManager.js`
- Get Transaction History
  - `getTransactionHistoryWithBlueCoinManager.js`

NodeJS programs that are grouped under the same function have different implementations but yield the same results.

For example, `generateInitialCoin.js`, `generateInitialCoinWithTransactionManager.js`, and `generateInitialCoinWithBlueCoinManager.js` are programs that allow the generation of initial coins. However, their differences are only in the implementations as described below:

- `generateInitialCoin.js` - utilizes only the NodeJS SDK Client for Hypereledger Fabric
- `generateInitialCoinWithTransactionManager.js` - utilizes `TransactionManager.js` in order to abstract the complexity in the use of the NodeJS SDK Client
- `generateInitialCoinWithBlueCoinManager.js` - utilizes `BlueCoinManager.js` in order to make it more readable

Enroll the Bootstrap Administrator

1. Confirm that the `wallet` subfolder does not container other subfolders such as `org` and `org2`.

```
wallet> ls
```

Expected Output:

```
clear-wallet.sh
```

You may also use a file explorer program (e.g., Windows Explorer) to view the contents of the `wallet` subfolder. Later, we will inspect further the contents of the `wallet` subfolder using a file explorer.

2. Open the following source in an IDE.

```
IDE> blockchain-tutorial/client/blue-coin/enrollAdmin.js
```

This program enrolls the bootstrap adminstrator of a particular organization. A bootsrap administrator is needed to register new users through a certificate authority.

It utilizes two Hyperledger Fabric NodeJS Client SDK:

- `fabric-ca-client` - to communicate with a certificate authority (CA) server
- `fabric-network` - to manage the wallet

```
const FabricCAServices = require('fabric-ca-client')
const { FileSystemWallet, X509WalletMixin } = require('fabric-network')
```

The program uses the following syntax.

Syntax:

```
bc-client> node enrollAdmin.js <org index>
```

Example:

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

In th example above, the bootstrap administrator of org1 is enrolled.

3. Install the necessary packages.

```
user-mgr> npm install

tx-mgr> npm install

bc-client> npm install
```

4. Enroll the bootstrap administrator of org1.

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

Expected Output:

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

5. Using a file explorer, check if there are subfolders and/or files created in the `wallet` subfolder.

   The `wallet` subfolder now contains the following:

   - `org1`
     - `admin`
       - `<long hex no.>-priv` - private key
       - `<long hex no.>-pub` - public key
       - `admin` - certificate

6. Enroll the bootstrap administrator of org2.

```
bc-client> node enrollAdminWithUserManager.js 2
```

Expected Output:

```
Enrolling Administrator admin of org2...
Administrator admin of org2 enrolled successfully.
```

Notice that we use `enrollAdminWithUserManager.js` instead of `enrollAdmin.js`. As discussed both have the same result but the former abstracts the complexity of the NodeJS SDK.

7. Check again the contents of the `wallet` subfolder.

   The `wallet` subfolder now contains subfolders and files for both org1 and org2:

   - `org1`

        - admin
            - <long hex no.>-priv - private key
            - <long hex no.>-pub - public key
            - admin - certificate
    - org2
        - admin
            - <long hex no.>-priv - private key
            - <long hex no.>-pub - public key
            - admin - certificate

Register and Enroll other Users

1. Open the following source in an IDE.

```
IDE> blockchain-tutorial/client/blue-coin/enrollUser.js
```

This program registers and enrolls a user of a particular organization. A bootsrap administrator for each organization is needed (which was enrolled in the previous steps) before new users can be registered and enrolled.

It utilizes two Hyperledger Fabric NodeJS Client SDK:

- fabric-ca-client - to communicate with the certificate authority (CA) server
- fabric-network - to manage the wallet and connect to the blockchain network

```
const FabricCAServices = require('fabric-ca-client')
const { FileSystemWallet, X509WalletMixin, Gateway } = require('fabric-network')
```

The program uses the following syntax.

Syntax:

```
bc-client> node enrollUser.js <org index> <user index>
```

Example:

```
bc-client> node enrollUser.js 1 2
```

In th example above, user2 of org1 is registered and enrolled.

2. Enroll user1 of org1.

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

Expected Output:

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

3. Using a file explorer, check if there is a subfolder `user1` created under `wallet/org1`.

   The `wallet` subfolder now contains the following:

   - `org1`
     - `user1`
       - `<long hex no.>-priv` - private key
       - `<long hex no.>-pub` - public key
       - `user1` - certificate

4. Enroll `user1` of org2.

   ```
   bc-client> node enrollUserWithUserManager.js 2 1
   ```

   Expected Output:

   ```
   Registering User user1... of org2
   User user1 of org2 registered successfully.
   Enrolling User user1... of org2
   User user1 of org2 enrolled successfully.
   ```

   Notice that `enrollUserWithUserManager.js` is used instead of `enrollUser.js`. As discussed, they produce the same result but are implemented differently.

5. Using a file explorer, check if there is a subfolder `user1` created under `wallet/org2`.

   The `wallet` subfolder now contains the following:

   - `org2`
     - `user1`
       - `<long hex no.>-priv` - private key
       - `<long hex no.>-pub` - public key
       - `user1` - certificate

Generate Initial Coins

1. Open the following source in an IDE.

   ```
   IDE> blockchain-tutorial/client/blue-coin/generateInitialCoin.js
   ```

   This program calls the `generateInitialCoin` function of the chaincode.

   It utilizes the Hyperledger Fabric NodeJS Client SDK`fabric-network` to manage the wallet and connect to the blockchain network.

   ```
   const { FileSystemWallet, Gateway } = require('fabric-network');
   ```

   The program uses the following syntax.

   Syntax:

   ```
   bc-client> node generateInitialCoin.js <org index> <user index> <mspId>
   ```

Example:

```
bc-client> node generateInitialCoin.js 1 2 Org1MSP
```

In th example above, `user2` of org1 is used to call the chaincode function `generateInitialCoin`. The parameter `Org1MSP` is passed as a parameter to the `generateInitialCoin` chaincode function.

Take note that a user (e.g., `user2`) should be registered and enrolled first before it is used to call any function.

2. Generate initial coins for org1.

```
bc-client> node generateInitialCoin.js 1 1 Org1MSP
```

Expected Output:

```
generateInitialCoin invocation successful.
Invoked by user1 of org1.
txId: 1b0daeec8ed8e908ad1951db5aa0085318f3bcb978cd52d4148ca8625b946c30  status: VALID
blockNo: 2
response: {
    "status": 200,
    "message": "Successfully generated blue coins",
    "payload": {
        "mspId": "Org1MSP",
        "amt": 500
    }
}
```

Note: The value of `txId` may be different in your output. In addition, the message `generateInitialCoin...` may come after the `response: { ...` output.

Just in case the NodeJS program does not exit, press `Ctrl+C` to exit.

3. Generate initial coins for org2.

```
bc-client> node generateInitialCoinWithTransactionManager.js 2 1 Org2MSP
```

Expected Output:

```
response: {
    "status": 200,
    "message": "Successfully generated blue coins",
    "payload": {
        "mspId": "Org2MSP",
        "amt": 500
    }
}
generateInitialCoin invocation successful.
Invoked by user1 of org2.
txId: ef0b45d6720ec876c816f71e81f1a80405e44a14e789d0e760f46ae6b272dccf  status: VALID
blockNo: 3
```

Notice that `generateInitialCoinWithTransactionManager.js` is used instead of `generateInitialCoin.js`. As discussed, they produce the same result but are implemented differently.

A third implemention of the program `generateInitialCoinWithBlueCoinManager.js` can also be used.

Test `BlueCoinManager.js`-based Programs

Several programs are created under the `client/blue-coin` subfolder that utilizes the `BlueCoinManager.js` program. You may test all of these to see their effects.

1. Examine the program details in the table below.

Purpose	Syntax	Example
Generate Initial Coin	`node generateInitialCoinWithBlueCoinManager.js <org index> <user index> <mspId>`	`node generateInitialCoinWithBlueCoinManager.js 1 2 Org1MSP`
Get Balance	`node getBalanceWithBlueCoinManager.js <org index> <user index> <mspId>`	`node getBalanceWithBlueCoinManager.js 1 2 Org1MSP`
Transfer Coin	`node transferCoinWithBlueCoinManager.js <org index> <user index> <srcMspId> <dstMspId> <amt>`	`node transferCoinWithBlueCoinManager.js 1 2 Org1MSP Org2MSP 210`
Get Transaction History	`node getTransactionHistoryWithBlueCoinManager.js <org index> <user index> <mspId>`	`node getTransactionHistoryWithBlueCoinManager.js 1 2 Org1MSP`

2. Get the balance of both org1 and org2.

```
bc-client> node getBalanceWithBlueCoinManager.js 1 1 Org1MSP

bc-client> node getBalanceWithBlueCoinManager.js 2 1 Org2MSP
```

Expected Output:

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

```
response: {
    "status": 200,
    "message": "Balance retrieved successfully",
    "payload": {
        "amt": 500,
        "mspId": "Org2MSP"
    }
}
```

Confirm that both org1 and org2 have 500 blue coins.

3. Try to transfer blue coins from org2 to org1 using user from org1.

```
bc-client> node transferCoinWithBlueCoinManager.js 1 1 Org2MSP Org1MSP 200
```

Expected Output:

```
response: {
    "status": 500,
    "message": "Error: No valid responses from any peers. 1 peer error responses:\n
peer=peer0.org1.example.com, status=500, message=transaction returned with failure: Error: The
parameter srcMspId should be the same as the caller's mspId: Org1MSP"
}
```

This results to an error since the chaincode implements an ACL. It does not allow an organization to transfer blue coins if it does not own the said blue coins.

In this case, `user1` of org1 attempts to transfer 200 blue coins from org2 to org1. This is prohibited by the chaincode.

4. Transfer blue coins from org2 to org1 using user from org2.

```
bc-client> node transferCoinWithBlueCoinManager.js 2 1 Org2MSP Org1MSP 200
```

Expected Output:

```
transferCoin invocation successful.
Invoked by user1 of org2.
txId: 35b9d825618a3865f1d5a95814ffd57187263e05bd30f7ac7be127b5558b73ac  status: VALID
blockNo: 4
response: {
    "status": 200,
    "message": "Transferred successfully the amount of 200 blue coins from Org2MSP to
Org1MSP",
    "payload": {}
}
```

The transfer is now successful since `user1` of org2 is used to transfer blue coins from org2 to org1.

Update `BlueCoinManager.js`

1. Open the following source in an IDE.

```
IDE> blockchain-tutorial/manager/blue-coin/BlueCoinManager.js
```

2. Edit the `BlueCoinManager` class to include the method `getAllAbove`.

3. In the `client/blue-coin` subfolder, create a program called `getAllAboveWithBlueCoinManager.js`

The program should use the following syntax.

Syntax:

```
bc-client> node getAllAboveWithBlueCoinManager.js <org index> <user index> <val>
```

Example:

```
bc-client> node getAllAboveWithBlueCoinManager.js 2 1 250
```

In the example above, `user1` of org2 is used to call the function `getAllAbove` to get all organizations that have blue coins above 250.