Lab 12.1: User Registration and Enrollment

Let’s now enroll CA server administrators and import their identity into a wallet from the client application. First, create the `enrollUser.js` file in the `applications/balance_transfer` folder. In this file, import all the necessary modules, including `fabric-ca-client`.

```
'use strict';

const fs = require('fs');
const path = require('path');

const FabricCAServices = require('fabric-ca-client');
const { Wallets } = require('fabric-network');

const testNetworkRoot = path.resolve(require('os').homedir(),
'go/src/github.com/hyperledger/fabric-samples/test-network');

async function main() {
   try {
       // Logic for user enrollment would go here.
   } catch (error) {
       console.error(`Failed to enroll user: ${error}`);
       process.exit(1);
   }
}

main().then(() => {
   console.log('User enrollment completed successfully.');
}).catch((e) => {
   console.log('User enrollment exception.');
   console.log(e);
   console.log(e.stack);
   process.exit(-1);
});
```

As you can see, we are going to use the FabricCAServices class to interact with Fabric CA. The `FabricCAServices` object can be initialized using the Fabric CA server URL. We have already defined this URL for `test-network` in the connection profile, so let's simply read it.

```
       let args = process.argv.slice(2);

       const identityLabel = args[0];
       const orgName = identityLabel.split('@')[1];
       const orgNameWithoutDomain = orgName.split('.')[0];

       let connectionProfile = JSON.parse(fs.readFileSync(
           path.join(testNetworkRoot,
               'organizations/peerOrganizations',
               orgName,
               `/connection-${orgNameWithoutDomain}.json`), 'utf8')
       );

       const ca = new
FabricCAServices(connectionProfile['certificateAuthorities'][`ca.${orgName}`]
.url);
```

Now, we can enroll a server administrator, using the `ca` object. `FabricCAServices` provides the `enroll` method that accepts the EnrollmentRequest object as its single parameter. `EnrollmentRequest` should contain the enrollment ID and the secret of a registered user and can also include additional information for customizing the enrollment certificate.

The enroll method returns the enrollment certificate along with a private key for the user. This information can be saved in the wallet and used to act on behalf of the user in the network.

Note: A private key is generated on the application side and is not transferred in the network.

Let's now compose an enrollment request. To make the application more flexible, we will use command-line arguments to pass an identity label and enrollment credentials. First of all, we should check whether a user has enrolled previously. To do that, initialize the wallet and check if the specified label is found.

```
       const wallet = await Wallets.newFileSystemWallet('./wallet');
```

```
        let identity = await wallet.get(identityLabel);
        if (identity) {
            console.log(`An identity for the ${identityLabel} user already
exists in the wallet`);
            return;
        }
```

After making sure that the user label is not taken, parse other necessary command-line arguments and compose an enrollment request.

```
        const enrollmentID = args[1];
        const enrollmentSecret = args[2];

        let enrollmentRequest = {
            enrollmentID: enrollmentID,
            enrollmentSecret: enrollmentSecret,
        };
```

Now, you can invoke the `enroll` method and save the user identity in the wallet.

```
        const enrollment = await ca.enroll(enrollmentRequest);

        const orgNameCapitalized =
orgNameWithoutDomain.charAt(0).toUpperCase() + orgNameWithoutDomain.slice(1);
        identity = {
            credentials: {
                certificate: enrollment.certificate,
                privateKey: enrollment.key.toBytes(),
            },
            mspId: `${orgNameCapitalized}MSP`,
            type: 'X.509',
        };

        await wallet.put(identityLabel, identity);
```

Now, we can easily enroll the Fabric CA server administrator and act on this user's behalf in the network.

1) Update the `dependencies` block in the `package.json` file.

```
    "dependencies": {
      "fabric-network": "^2.2.4",
        "fabric-ca-client": "^2.2.4"
    }
```

2) Run `npm install` to update the `node_modules` folder.

    ```
    # npm install
    ```

3) Enroll the `Org1`'s Fabric CA server administrator, using the bootstrapping credentials.

    ```
    # node enrollUser.js 'CAAdmin@org1.example.com' admin adminpw
    ```

    You should receive a success message from the application. You should also find a newly created `CAAdmin@org1.example.com` identity in the `wallet` folder.

As mentioned above, the enrollment process can be initiated only for registered identities. An identity can be registered only by another identity with proper authority. The identity performing a register request should also be currently enrolled. That is why the Fabric CA server requires at least one bootstrap identity to start.

The `FabricCAServices` class provides the `register` method for user registration in Fabric CA. Similar to `enroll`, `register` accepts the `RegisterRequest` object with user-specific information. However, there is also one more parameter—the `User` object representing a registrar.

Let's implement a registration workflow in the `registerUser.js` file. Create the file, import all required modules, and parse the first command-line argument. We expect it to be the registrar label, as it is necessary to check whether a registrar is enrolled.

```
'use strict';

const fs = require('fs');
const path = require('path');

const FabricCAServices = require('fabric-ca-client');
const { Wallets, Gateway } = require('fabric-network');

const testNetworkRoot = path.resolve(require('os').homedir(),
'go/src/github.com/hyperledger/fabric-samples/test-network');

async function main() {
   try {
       const wallet = await Wallets.newFileSystemWallet('./wallet');
```

```
        let args = process.argv.slice(2);

        const registrarLabel = args[0];

        let registrarIdentity = await wallet.get(registrarLabel);
        if (!registrarIdentity) {
            console.log(`An identity for the registrar user ${registrarLabel}
does not exist in the wallet`);
            console.log('Run the enrollUser.js application before retrying');
            return;
        }

        // Logic for user registration would go here.

    } catch (error) {
        console.error(`Failed to register user: ${error}`);
        process.exit(1);
    }
}

main().then(() => {
    console.log('User registration completed successfully.');
}).catch((e) => {
    console.log('User registration exception.');
    console.log(e);
    console.log(e.stack);
    process.exit(-1);
});
```

Once we know the registrar is enrolled, we can start the registration process. First, create the `FabricCAServices` object to interact with the Fabric CA server.

```
        const orgName = registrarLabel.split('@')[1];
        const orgNameWithoutDomain = orgName.split('.')[0];

        let connectionProfile = JSON.parse(fs.readFileSync(
            path.join(testNetworkRoot,
                'organizations/peerOrganizations',
                orgName,
                `/connection-${orgNameWithoutDomain}.json`), 'utf8')
        );
        const ca = new
 FabricCAServices(connectionProfile['certificateAuthorities'][`ca.${orgName}`]
 .url);
```

Next, we should obtain the registrar identity in the form of the `User` object. Note that we cannot use the `Wallet.get` method because it returns the identity object, while the `User` object is required. The simplest way to obtain a properly constructed registrar `User` object is to initialize a provider for a certain identity type (e.g., `X.509`), and get a user context based on the identity.

```
        const provider =
wallet.getProviderRegistry().getProvider(registrarIdentity.type);
        const registrarUser = await provider.getUserContext(registrarIdentity,
registrarLabel);
```

Now, we have the registrar. The only thing left is to compose the `RegisterRequest` object and pass it to the `ca.register` method. `RegisterRequest` has only one required field—the enrollment ID of a user. It is also possible to set the enrollment secret, role, affiliation, and other optional parameters that will affect the enrollment process. If an enrollment secret is not set, then the Fabric CA server will generate one for a user and return it as a result of `ca.register`.

In `registerUser.js`, parse command-line arguments to extract the enrollment ID and a JSON object containing optional fields if provided.

```
        const enrollmentID = args[1];

        let optional = {};
        if (args.length > 2) {
            optional = JSON.parse(args[2]);
        }
```

To keep it simple, at this stage, we will only process the `secret` field in `optional`. Thus, a complete request can look as follows.

```
        let registerRequest = {
            enrollmentID: enrollmentID,
            enrollmentSecret: optional.secret || "",
            role: 'client',
        };
        const secret = await ca.register(registerRequest, registrarUser);
        console.log(`Successfully registered the user with the ${enrollmentID}
enrollment ID and ${secret} enrollment secret.`);
```

Let’s now register and enroll a new user with `CAAdmin@org1.example.com` as a registrar.

```
# node registerUser.js 'CAAdmin@org1.example.com' 'User@org1.example.com'
'{"secret": "userpw"}'
# node enrollUser.js 'User@org1.example.com' 'User@org1.example.com' userpw
```