Lab 4. Developing with HyperLedger Composer

In this lab, we're going to install the development environment and create queries for a sample network that we will call 'braunwidgets'.

Step 1. Install prerequisites.

The following are prerequisites for installing the required development tools:

Operating Systems: Ubuntu Linux 14.04 / 16.04 LTS (both 64-bit), or Mac OS 10.12

Docker Engine: Version 17.03 or higher

Docker-Compose: Version 1.8 or higher

Node: 8.9 or higher (note version 9 is not supported)

npm: v5.x

git: 2.9.x or higher

Python: 2.7.x.

Run the following commands

```
$ curl -O https://hyperledger.github.io/composer/prereqs-ubuntu.sh
$ chmod u+x prereqs-ubuntu.sh
$ ./prereqs-ubuntu.sh
```

Step 2. Install the composer tools.

Now we'll install the composer CLI tools and the playground, if it hasn't already been installed.

```
$ npm install -g composer-cli
$ npm install -g composer-rest-server
$ npm install -g generator-hyperledger-composer
```

```
$ npm install -g yo
$ npm install -g composer-playground
```

The above commands install the composer-cli tool, the Rest server so that we can generate HTTP requests to the queries, it install the generator and a tool called 'yeoman' that will automatically build our network for us. Finally we install the composer-playground web app if it isn't already installed.

In order to do the next series of commands, we'll need to install the unzip tool for Ubuntu.

```
$ sudo apt-get update
$ sudo apt-get install unzip
```

Step 3. Create the fabric tools directory and install the binaries.

Next we'll create the directory that the fabric-tools will be stored to.

```
$ mkdir ~/fabric-tools && cd ~/fabric-tools
$ curl -O https://raw.githubusercontent.com/hyperledger/composer-
tools/master/packages/fabric-dev-servers/fabric-dev-servers.zip
$ unzip fabric-dev-servers.zip
```

Now, make sure you're in the fabric-tools directory and run the downloadFabric.sh script.

```
$ cd ~/fabric-tools
$ ./downloadFabric.sh
```

Step 4. Start the network.

We will control the network runtime through a series of scripts. The first time you run the network you will need to run the following in the fabric-tools directory.

```
$ cd ~/fabric-tools
$ ./startFabric.sh
$ ./createPeerAdminCard.sh
```

It is possible to stop the runtime by simply typing in

```
$ ~/fabric-tools/stopFabric.sh
```

There is also a teardownFabric.sh script. Note that if you run this, you will need to create a new peeradmin card the next time you start the network again.

Step 5. Create the business network structure.

We'll start by using the yeoman tool that we downloaded earlier.

```
$ yo hyperledger-composer:businessnetwork
```

The tool will prompt you for a number of questions:

1. Select 'braunwidgets as the network name.
2. For description, enter 'Braun Widget Network'
3. For the author name enter 'Braun Brelin'
4. Enter 'bbrelin@gmail.com' for the author e-mail.
5. Select the Apache-2.0 software license.
6. Select com.braunwidgets.widgetnet as the namespace.

Note: You are certainly free to replace my name and information with your own.

Step 6. Define the Business Network

A business network is made up of assets, participants, transactions, access control rules, and optionally events and queries. In the skeleton business network created in the previous steps, there is a model (.cto) file which will contain the class definitions for all assets, participants, and transactions in the business network. The skeleton business network also contains an access control (permissions.acl) document with basic access control rules, a script (logic.js) file containing transaction processor functions, and a package.json file containing business network metadata.

We are now going to create or modify all of these files.

Make sure that you are in the 'braunwidgets' directory created in Step 5.

Open the 'model/com.braunwidgets.widgetnet.cto' file with whatever editor you choose.

Replace the existing file with the following:

```
/**
 * My commodity trading network
 */
namespace com.braunwidgets.widgetnet
asset Commodity identified by tradingSymbol {
    o String tradingSymbol
    o String description
    o String mainExchange
    o Double quantity
    --> Trader owner
}
participant Trader identified by tradeId {
    o String tradeId
    o String firstName
    o String lastName
}
transaction Trade {
    --> Commodity commodity
    --> Trader newOwner
}
transaction RemoveHighQuantityCommodities {
}

event RemoveNotification {
    --> Commodity commodity
}
```

Save the changes and exit.

Now, open the lib/logic.js script file.

Replace the contents with the following:

```
/**
 * Track the trade of a commodity from one trader to another
 * @param {com.braunwidgets.widgetnet.Trade} trade - the trade to be
processed
 * @transaction
 */
function tradeCommodity(trade) {
    trade.commodity.owner = trade.newOwner;
    return getAssetRegistry('com.braunwidgets.widgetnet.Commodity')
        .then(function (assetRegistry) {
            return assetRegistry.update(trade.commodity);
        });
}
/**
 * Remove all high volume commodities
 * @param {com.braunwidgets.widgetnet.RemoveHighQuantityCommodities}
remove - the remove to be processed
 * @transaction
 */
function removeHighQuantityCommodities(remove) {

    return getAssetRegistry('com.braunwidgets.widgetnet.Commodity')
        .then(function (assetRegistry) {
            return query('selectCommoditiesWithHighQuantity')
                    .then(function (results) {

                        var promises = [];

                        for (var n = 0; n < results.length; n++) {
                            var trade = results[n];

                            // emit a notification that a trade was
removed
                            var removeNotification =
getFactory().newEvent('com.braunwidgets.widgetnet',
'RemoveNotification');
                            removeNotification.commodity = trade;
                            emit(removeNotification);

                            // remove the commodity

promises.push(assetRegistry.remove(trade));
                        }

                        // we have to return all the promises
                        return Promise.all(promises);
                    });
        });
}
```

Save your changes and exit

Now, we'll create a permissions.acl file. This file decides the access rules for the widgetnet.

```
/**
 * Access control rules for tutorial-network
 */
rule Default {
    description: "Allow all participants access to all resources"
    participant: "ANY"
    operation: ALL
    resource: "com.braunwidgets.widgetnet.*"
    action: ALLOW
}

rule SystemACL {
  description:  "System ACL to permit all access"
  participant: "ANY"
  operation: ALL
  resource: "org.hyperledger.composer.system.**"
  action: ALLOW
}
```

Save this file and exit.

Step 7. Generate a business network archive

Making sure that you are in the braunwidgets directory run the following command:

```
$ composer archive create -t dir -n .
```

After this command has been run, a file called braunwidgets@0.0.1.bna will have been created. This is the archive file containing all of the files that you have either created or modified, along with the business network skeleton.

Step 8. Install and deploy the business network

Note that for this step, we've already created a Peer Admin card for the network administrator.

Run the following command to install our braunwidget widgetnet

```
$ composer runtime install --card PeerAdmin@hlfv1 --
businessNetworkName braunwidgets
```

Import the card

```
$ Composer card import --file networkadmin.card
```

Now, we'll deploy it:

```
$ composer network start --card PeerAdmin@hlfv1 --networkAdmin admin -
-networkAdminEnrollSecret adminpw --archiveFile braunwidgets@0.0.1.bna
--file networkadmin.card
```

Step 9. Deploy the REST API Server

Make sure that you are in the braunwidgets directory and run the following command:

```
$ composer-rest-server
```

When prompted, answer the following questions:

Enter admin@braunwidgets as the card name.

Select never use namespaces when asked whether to use namespaces in the generated API.

Select No when asked whether to secure the generated API.

Select Yes when asked whether to enable event publication.

Select No when asked whether to enable TLS security.

Once this is done, a REST server will be started at: https://<IP address of server>:3000/explorer

Step 10. Write a REST client that makes calls to the REST API.