V3.1

Hyperledger Fabric Masterclass for Administrators – Student Lab Guide

BLOCKCHAIN TRAINING ALLIANCE
KB

Table of Contents

Lab 1 – Installing Required Components

In this lab you will install all the required Fabric components.

Creating Project Folders

Open a terminal window, then create the required project folders and sub-folders by using the commands below.

```
cd ~/Desktop

mkdir fabric

cd fabric

mkdir sdk

mkdir network

mkdir cc
```

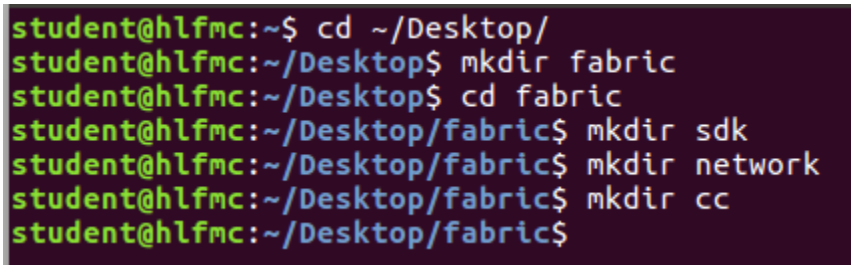

Figure 1 – Snippet 1.1

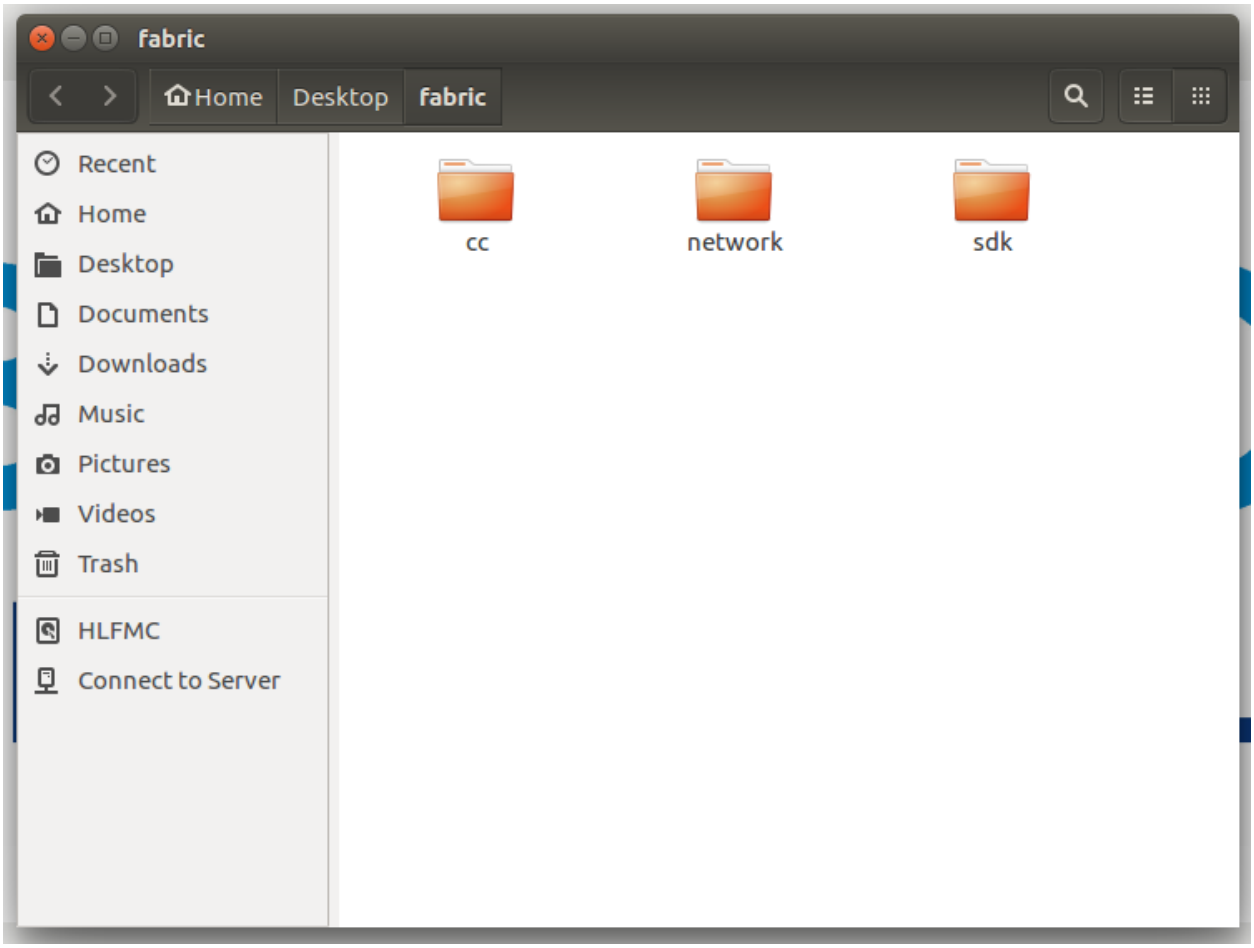

Figure 2 - The project folder structure viewed through the Files explorer

Installing Prerequisites

Update the system package repositories and install any available package upgrades using the commands below.

```
sudo apt update
```

```
sudo apt -y upgrade
```

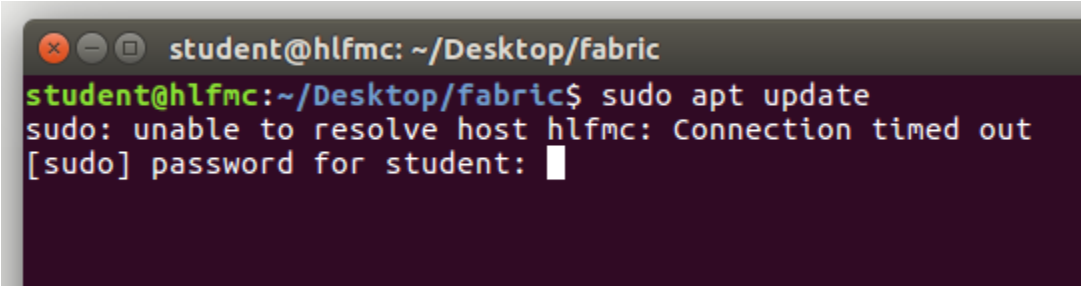

Figure 3 - Snippet 1.2

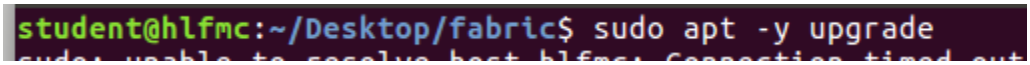

Figure 4 - Snippet 1.3

Install curl using the command below.

```
sudo apt install curl
```

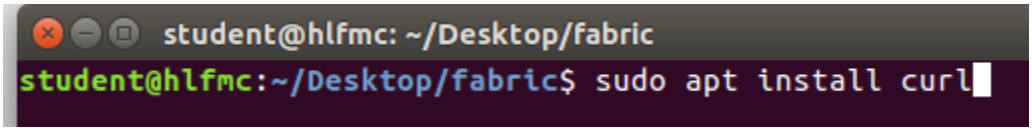

Figure 5 - Snippet 1.4

Install git using the command below.

```
sudo apt install git
```

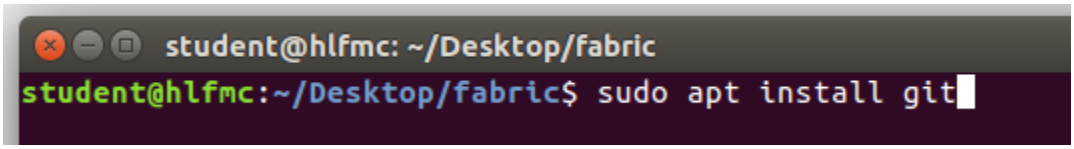

Figure 6 - Snippet 1.5

Installing Docker

Start by updating the system repositories using the command below.

```
sudo apt update
```

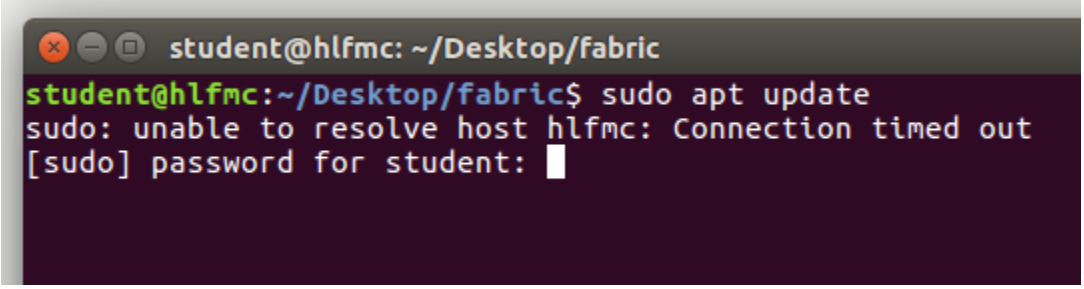

Figure 7 - Snippet 1.6

Install all the required Docker pre-requisites using the commands below.

```
sudo apt install apt-transport-https
sudo apt install ca-certificates
sudo apt install gnupg-agent
sudo apt install software-properties-common
```

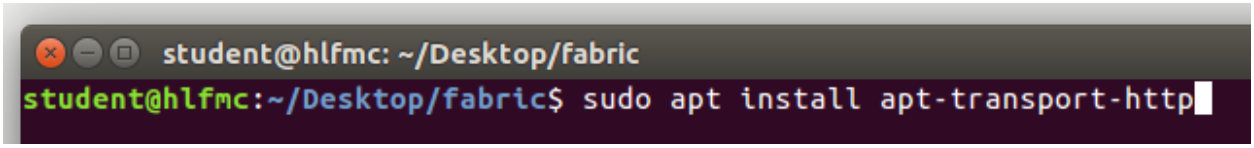

Figure 8 - Snippet 1.7

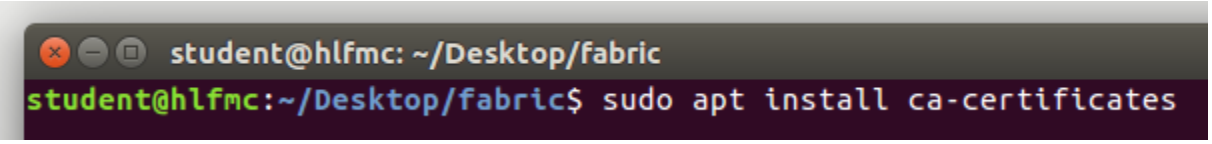

Figure 9 - Snippet 1.8

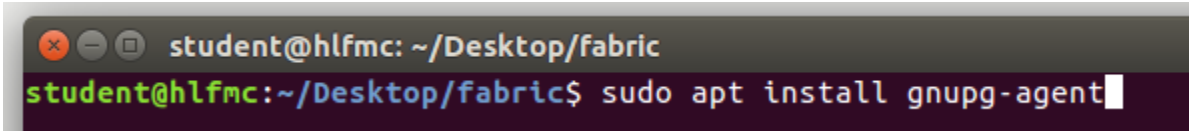

Figure 10 - Snippet 1.9

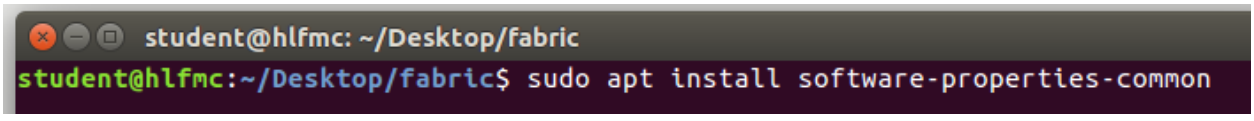

Figure 11 - Snippet 1.10

Curl Docker key from host site and add it to registry list using the command below.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

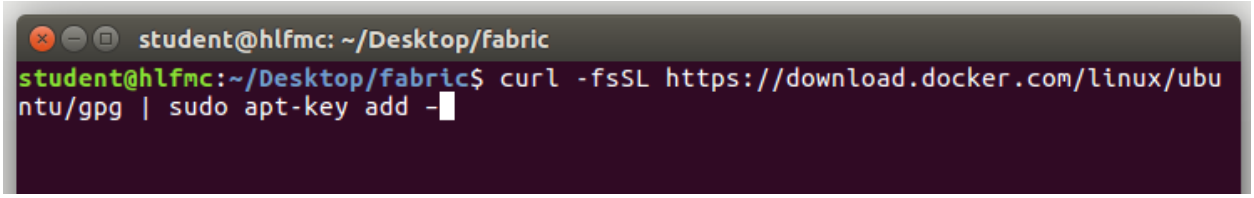

Figure 12 - Snippet 1.11

Use the command below to add Docker to system repository list.

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

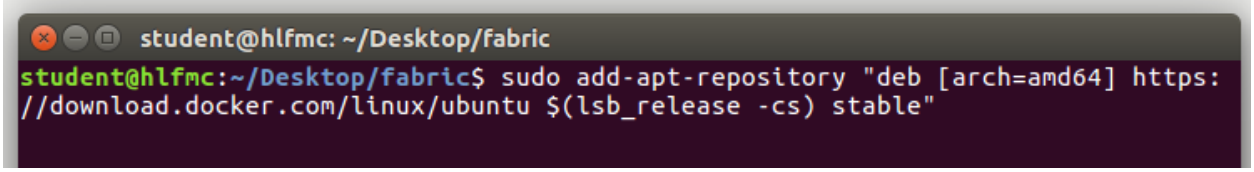

Figure 13 - Snippet 1.12

Now that Docker has been added to the system repository list, use the command below to update system repository.

```
sudo apt update
```

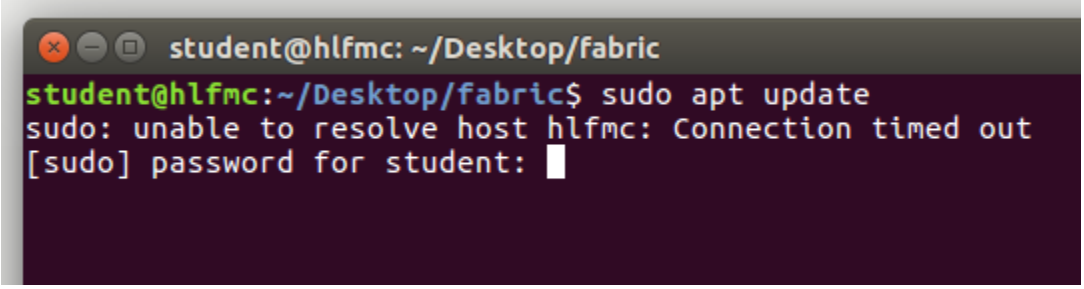

Figure 14 - Snippet 1.13

Install Docker Community Edition (CE) by using the command below.

```
sudo apt -y install docker-ce
```

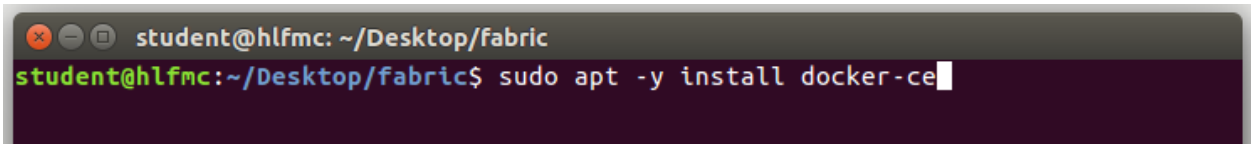

Figure 15 - Snippet 1.14

Next, add Docker to the Student user group with the following command.

```
sudo usermod -aG docker student
```

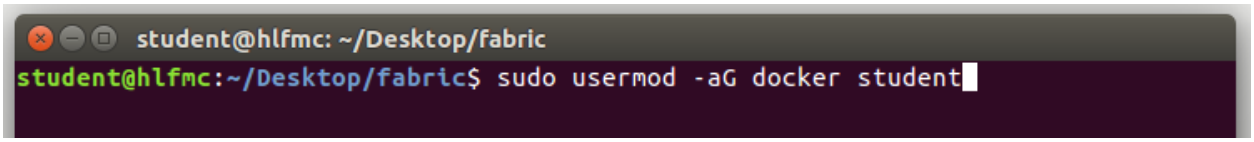

Figure 16 - Snippet 1.15

Use the command below to verify Docker is installed (NOTE: your version number may be different).

```
docker -v
```

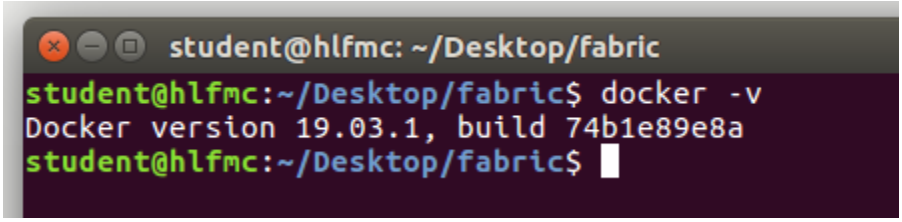

Figure 17 - Snippet 1.16

Installing Docker Compose

Start by using the *curl* command to get the Docker Composer file.

```
sudo curl -L
https://github.com/docker/compose/releases/download/1.18.0/docker-
compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

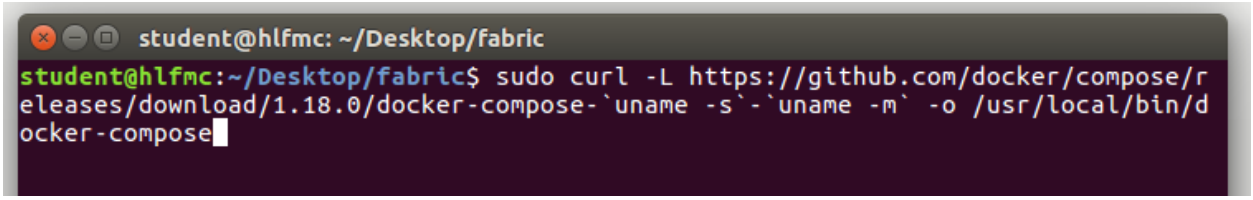

Figure 18 - Snippet 1.17

Modify permissions of Docker Compose using the command below.

```
sudo chmod +x /usr/local/bin/docker-compose
```

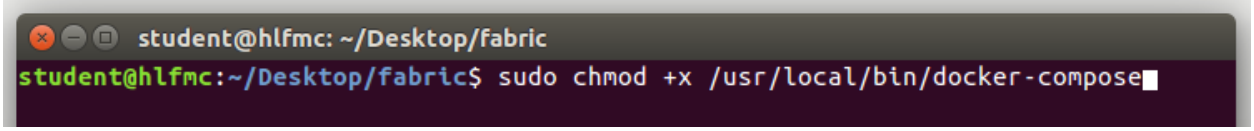

Figure 19 - Snippet 1.18

Reboot the system using the command below.

```
sudo reboot now
```

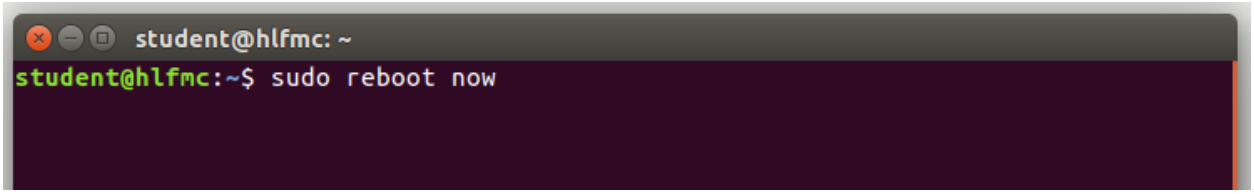

Figure 20 - Snippet 1.19

Once the system has rebooted, log in. Then open a terminal window and verify Docker Compose is installed by running the following command.

```
docker-compose -v
```

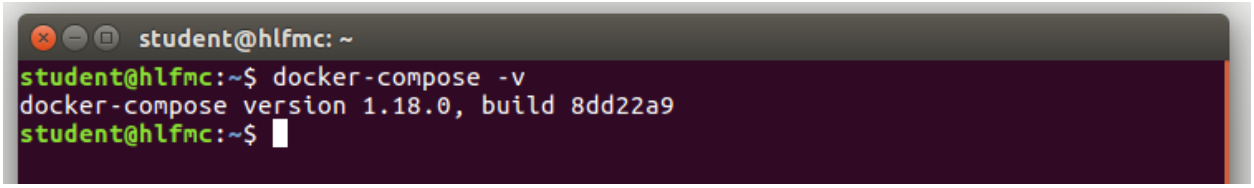

Figure 21 - Snippet 1.20

Installing Golang

Start the Golang installation by getting the installation files using the command below.

```
wget https://s3.us-east-2.amazonaws.com/fabric-masterclass/go.tar.gz
```

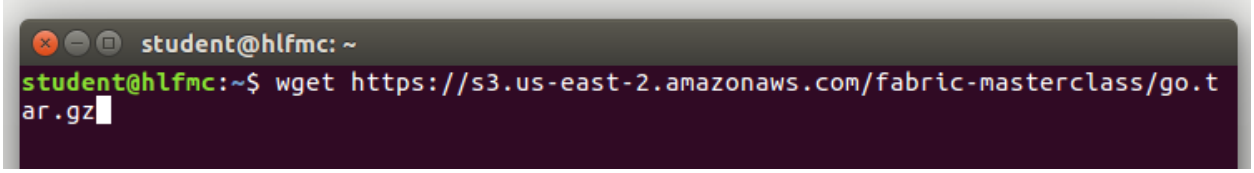

Figure 22 - Snippet 1.21

Next, use the command below to unzip the downloaded installation files.

```
tar -xvf go.tar.gz
```

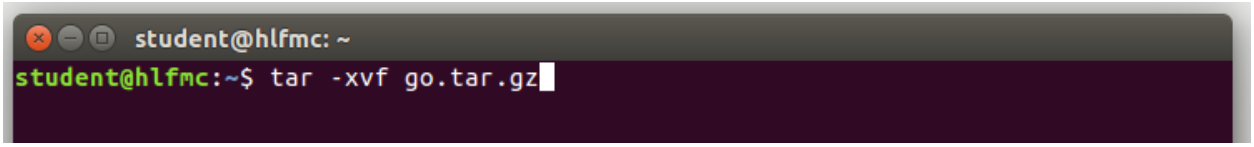

Figure 23 - Snippet 1.22

Move the go package to the */usr* directory using the command below.

```
sudo mv go /usr/local/go
```

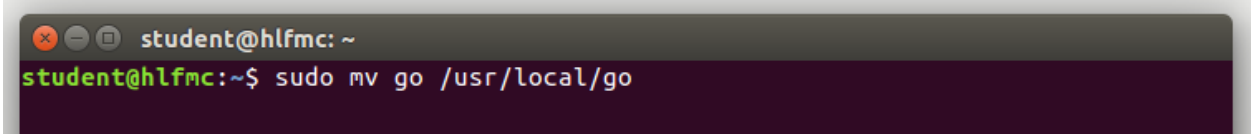

Figure 24 - Snippet 1.23

Enter the commands below to set the required Go system environment variables.

```
cd ~/Desktop/fabric/cc
```

```
export GOPATH=$PWD
```

```
echo $GOPATH
```

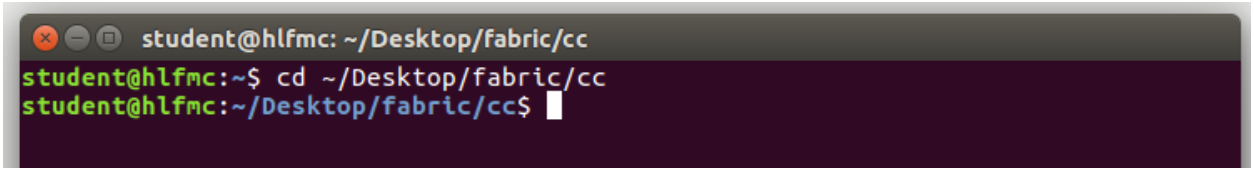

Figure 25 - Snippet 1.24

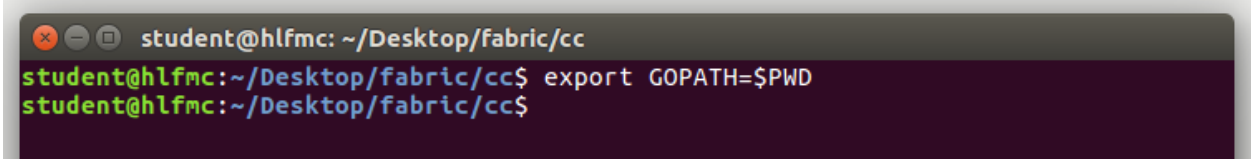

Figure 26 - Snippet 1.25

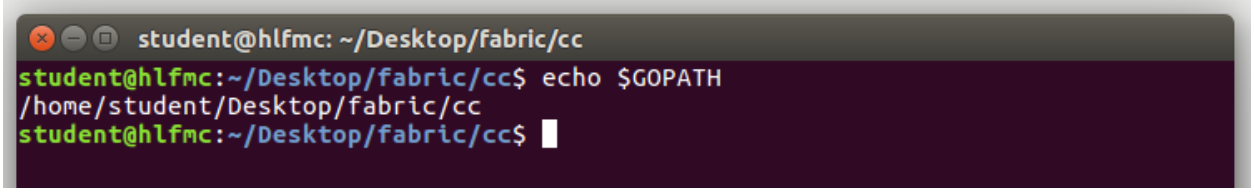

Figure 27 - Snippet 1.26

In this step, the *.profile* file will be edited to set required Go environment variables for every terminal session. Start by running the command below.

```
gedit ~/.profile
```

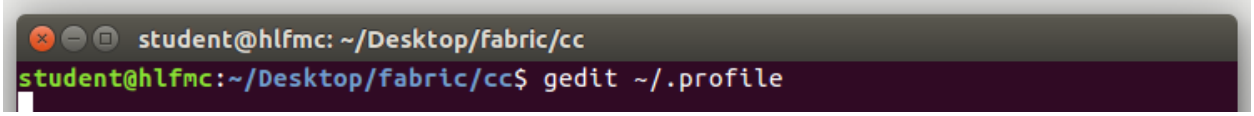

Figure 28 - Snippet 1.27

Next, add the following lines to the bottom of the file and save the changes.

```
export GOROOT=/usr/local/go

export GOPATH=/home/student

export PATH=$GOPATH/bin:$GOROOT/bin:$PATH
```

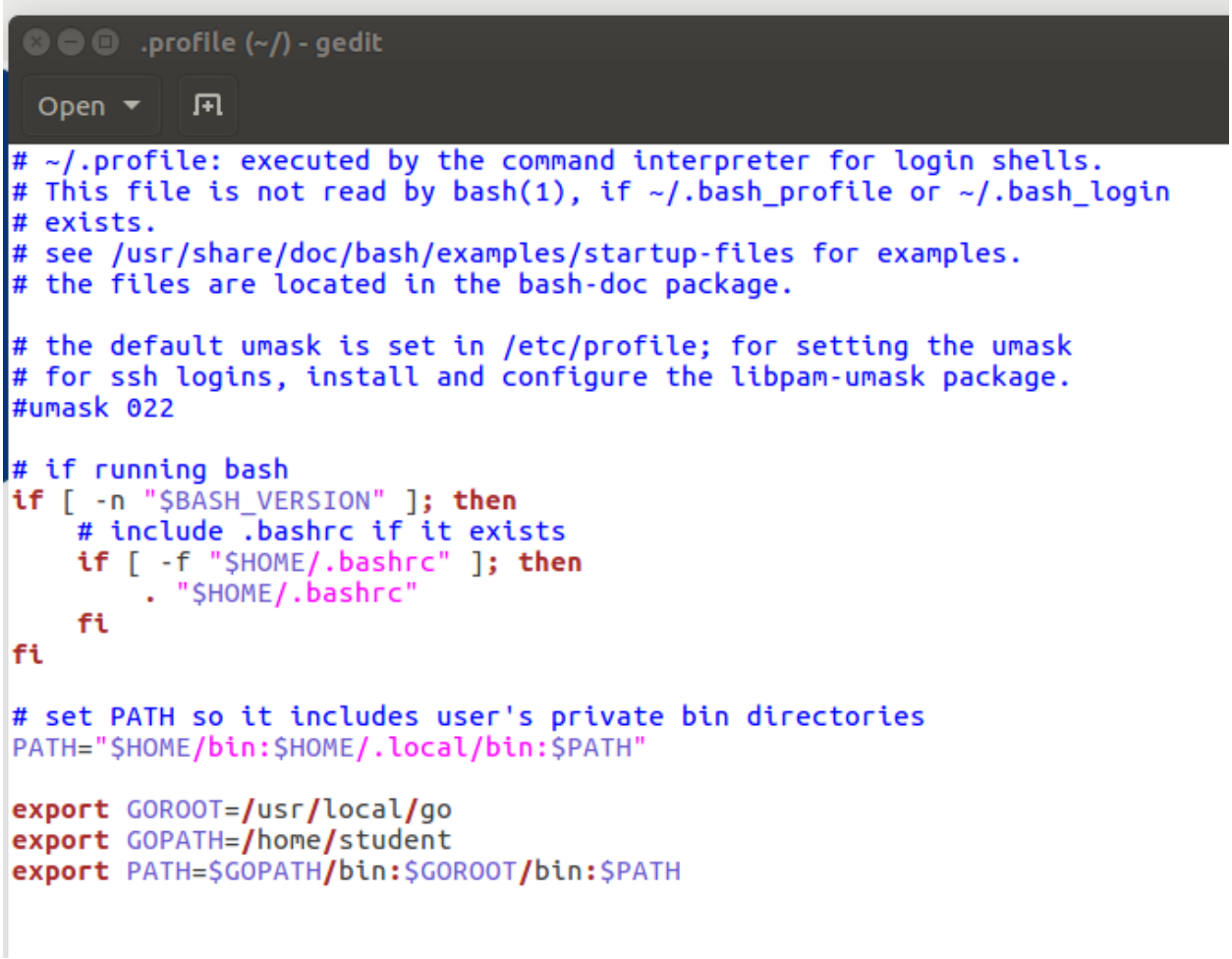

Figure 29 - Snippet 1.28

Next, execute the updated *.profile* file using the source command.

```
source ~/.profile
```

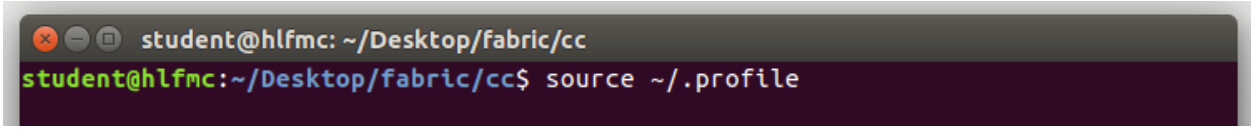

Figure 30 - Snippet 1.29

Check to make sure Go is installed correctly.

```
go version
```

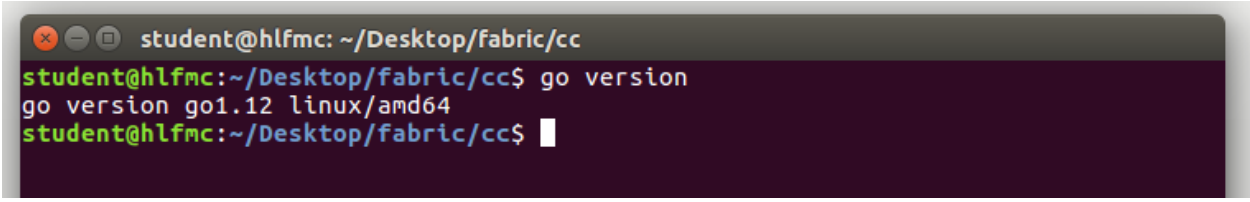

Figure 31 - Snippet 1.30

Installing Node.js

Begin the Node.js installation by issuing the curl command below.

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo bash -
```

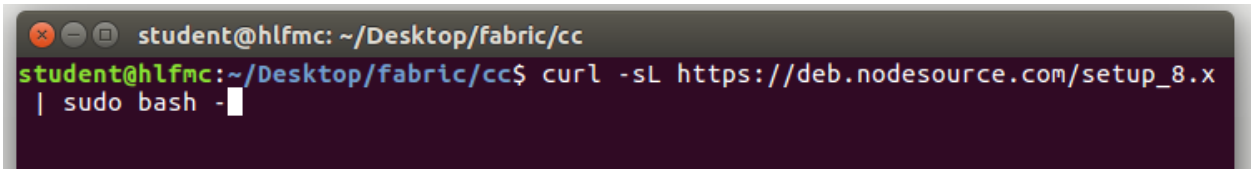

Figure 32 - Snippet 1.31

Install the Node JS package using the command below.

```
sudo apt install nodejs
```

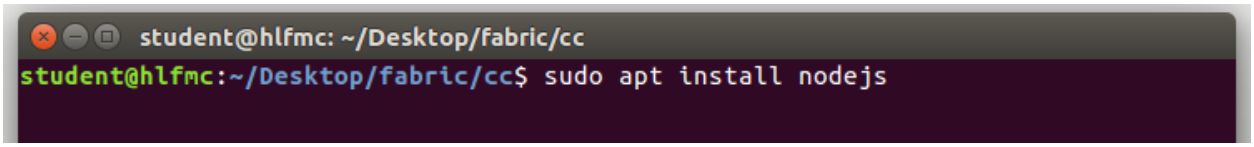

Figure 33 - Snippet 1.32

Verify Node and NPM are installed correctly using the following commands.

```
node -v
```

```
npm -v
```

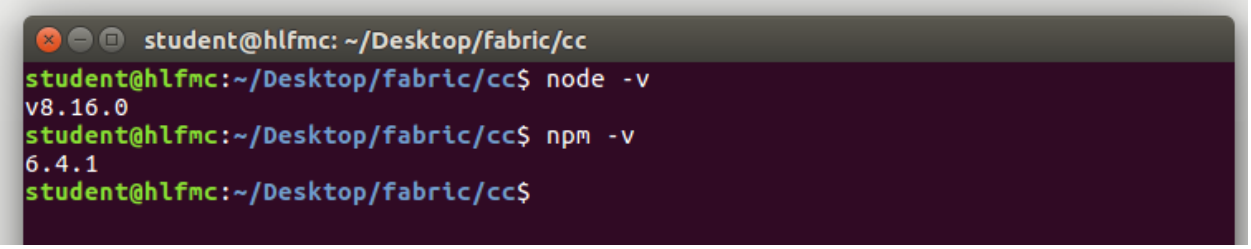

Figure 34 - Snippet 1.33

Lab 2 – Configuring the Network

In this lab you will learn about and work with the various YAML configuration files used by Fabric to configure a network.

Creating the Required Files and Folders

Begin by navigating to the *fabric/network* folder.

```
cd ~/Desktop/fabric/network
```

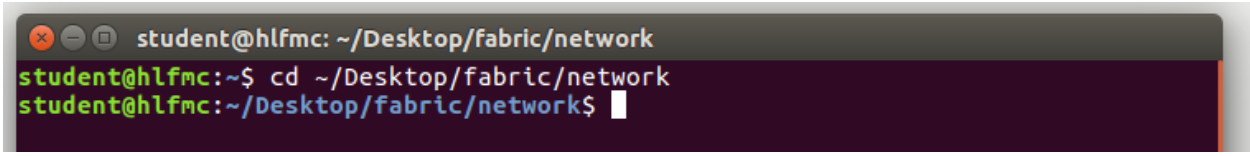

Figure 35 - Snippet 2.1

Use the touch command to create three new files (*configtx.yaml, crypto-config.yaml,* and *docker-compose.yml*).

```
touch configtx.yaml
```

```
touch crypto-config.yaml
```

```
touch docker-compose.yml
```

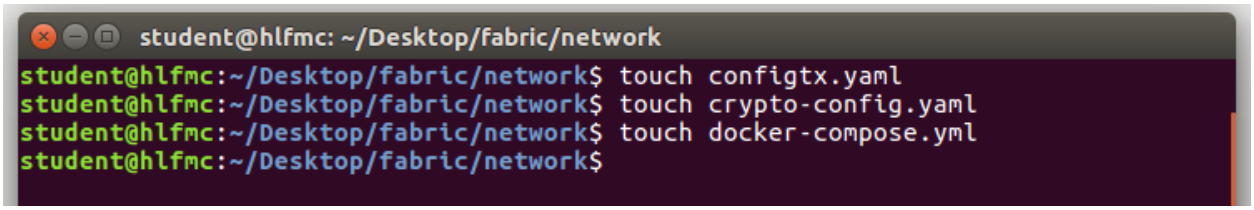

Figure 36 - Snippet 2.2

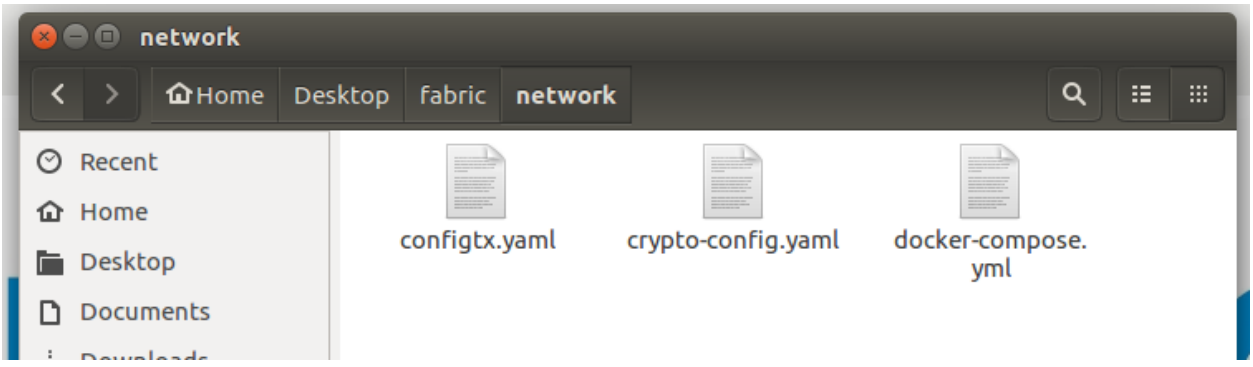

Figure 37 - The files created by running the touch commands

Two directories will be needed, one for configuration outputs and the second for managing default configuration values to be referenced upon network initialization. Create these directories using the two commands below.

```
mkdir config
```

```
mkdir defaults
```

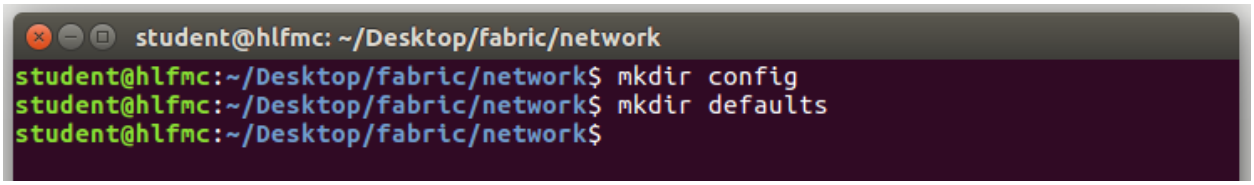

Figure 38 - Snippet 2.3

Figure 39 - The folders created using the mkdir commands

To save time, use the provided default configuration files for your Fabric network. These configuration files will be consumed during network initialization. The *orderer.yaml* file contains Ordering service configuration defaults, while *core.yaml* contains default configuration values for all peer nodes on the network.

```
cd defaults
```

```
wget https://s3.us-east-2.amazonaws.com/fabric-masterclass/orderer.yaml
```

```
wget https://s3.us-east-2.amazonaws.com/fabric-masterclass/core.yaml
```

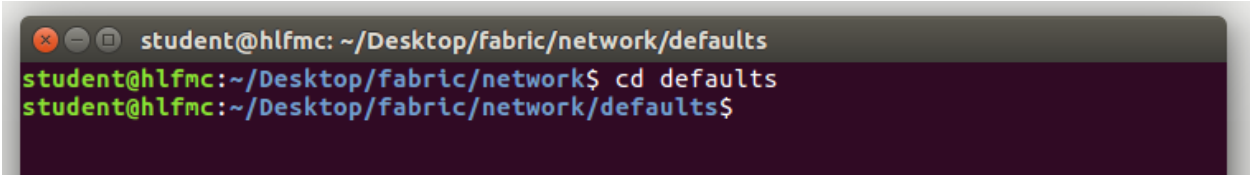

Figure 40 - Snippet 2.4

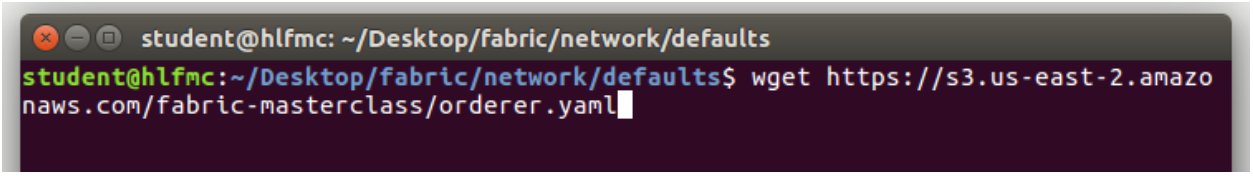

Figure 41 - Snippet 2.5

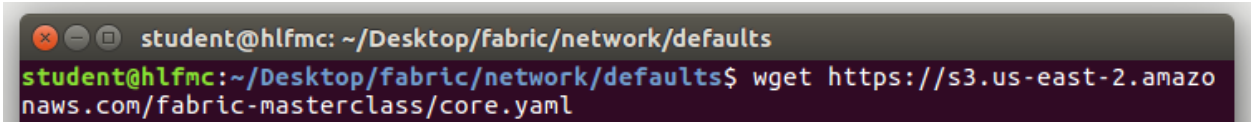

Figure 42 - Snippet 2.6

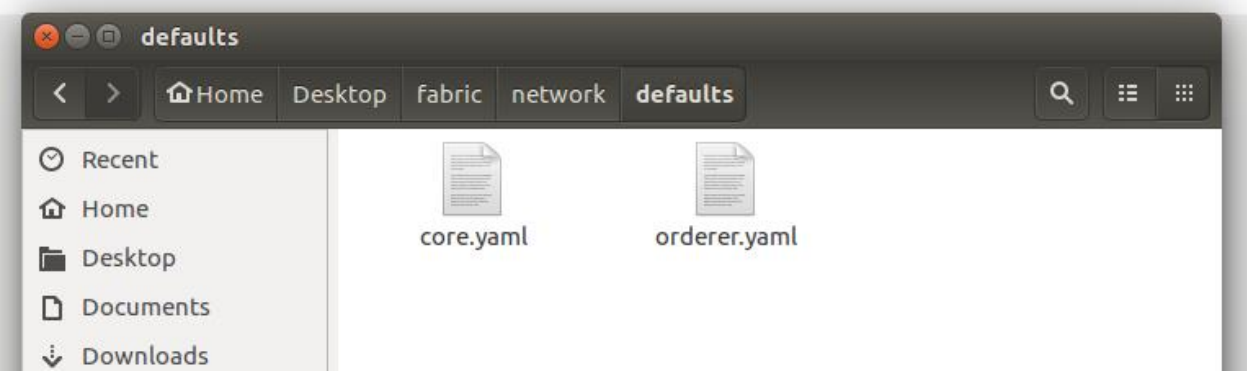

Figure 43 - The files downloaded using the wget commands

Run the following command to pull the Fabric version 1.4 binary tools.

```
cd ..
wget https://s3.us-east-2.amazonaws.com/fabric-masterclass/bin.tar.gz
```

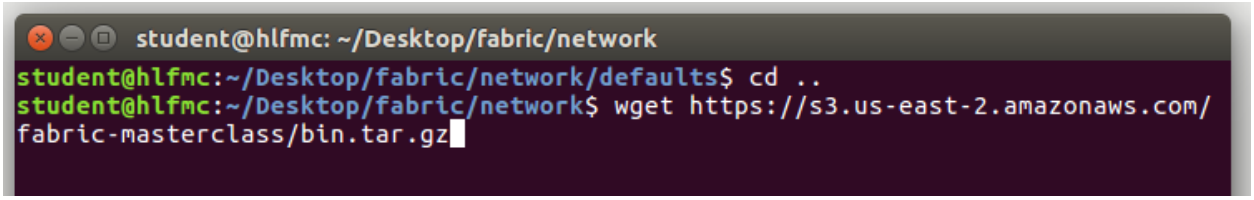

Figure 44 - Snippet 2.7

Unzip the downloaded files.

```
tar -xvf bin.tar.gz
```

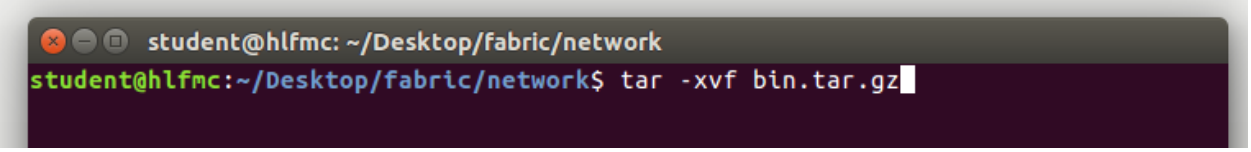

Figure 45 - Snippet 2.8

Permissions need to be updated on the extracted binaries so they can be executed. Use the following commands to update permissions.

```
cd bin
chmod +755 *
```

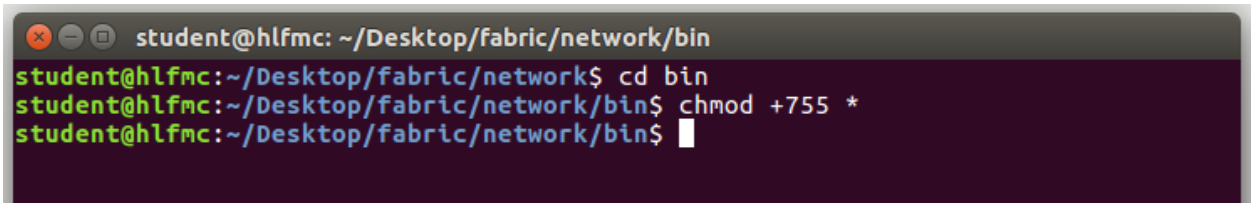

Figure 46 - Snippet 2.9

Now the extracted binaries will be moved under the */usr/local/bin* location so they can be executed from anywhere on the system. Use the following commands to move the binaries and delete the downloaded zip archive as well as the now empty bin folder under *Desktop/fabric/network*.

```
sudo mv * /usr/local/bin
```

```
cd ..
```

```
rm -rf bin bin.tar.gz
```

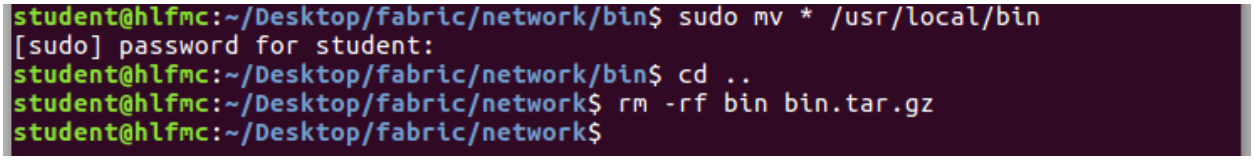

Figure 47 - Snippet 2.10

Editing Environment Variables

More default environment variables will need to be added to the .profiles file. Run the command below to edit the .profiles file.

```
gedit ~/.profile
```

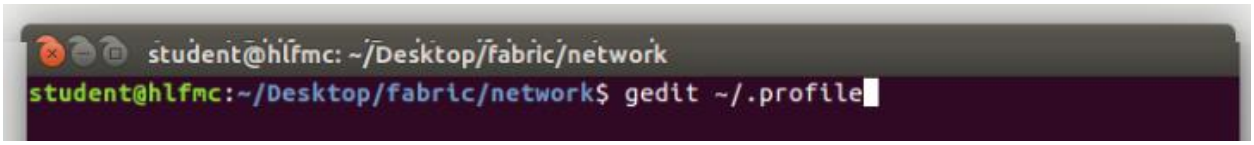

Figure 48 - Snippet 2.11

Add the following lines to the bottom of the file and save.

```
export FABRIC_CFG_PATH=/home/student/Desktop/fabric/network
export FABRIC_HOME=/home/student/Desktop/fabric/
```

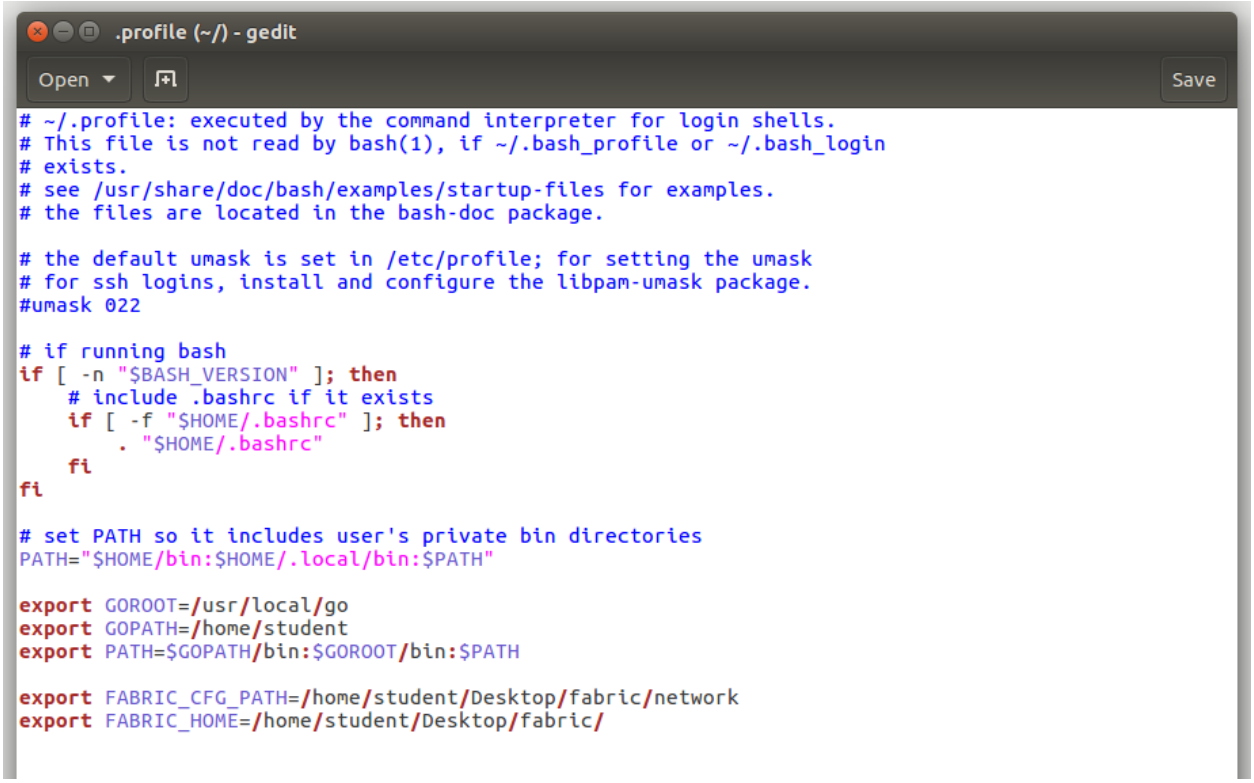

Figure 49 - Snippet 2.12

Execute the updated .profile file using the source command.

```
source ~/.profile
```

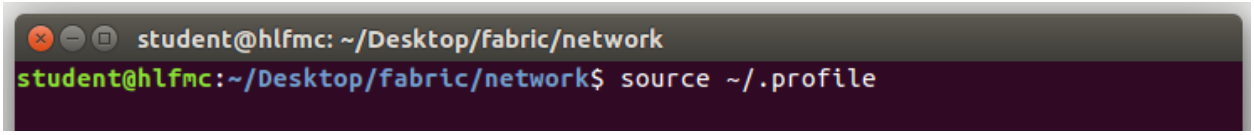

Figure 50 - Snippet 2.13

Ensure the latest settings to .profile are active by using the following command.

```
echo $FABRIC_HOME && echo $FABRIC_CFG_PATH
```

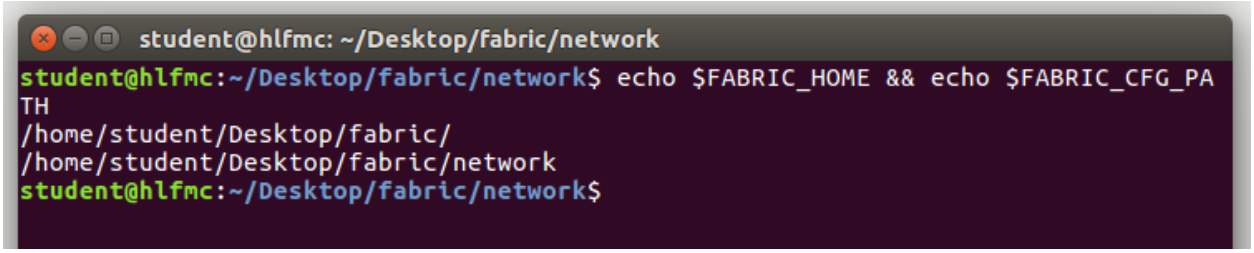

Figure 51 - Snippet 2.14

Building the *docker-compose.yml* file

This file contains information about container startup configuration and commands. Used by Docker Compose tool to stand up network. Allows for easy network startup, teardown, and restarts.

Open the *docker-composer.yml* file in Visual Studio code. Double-click on the *fabric* folder on the Desktop. From the *fabric* folder open the *network* folder. Then right-click on the *docker-compose.yml* file and select *Visual Studio Code* from the *Open With* dialogue.

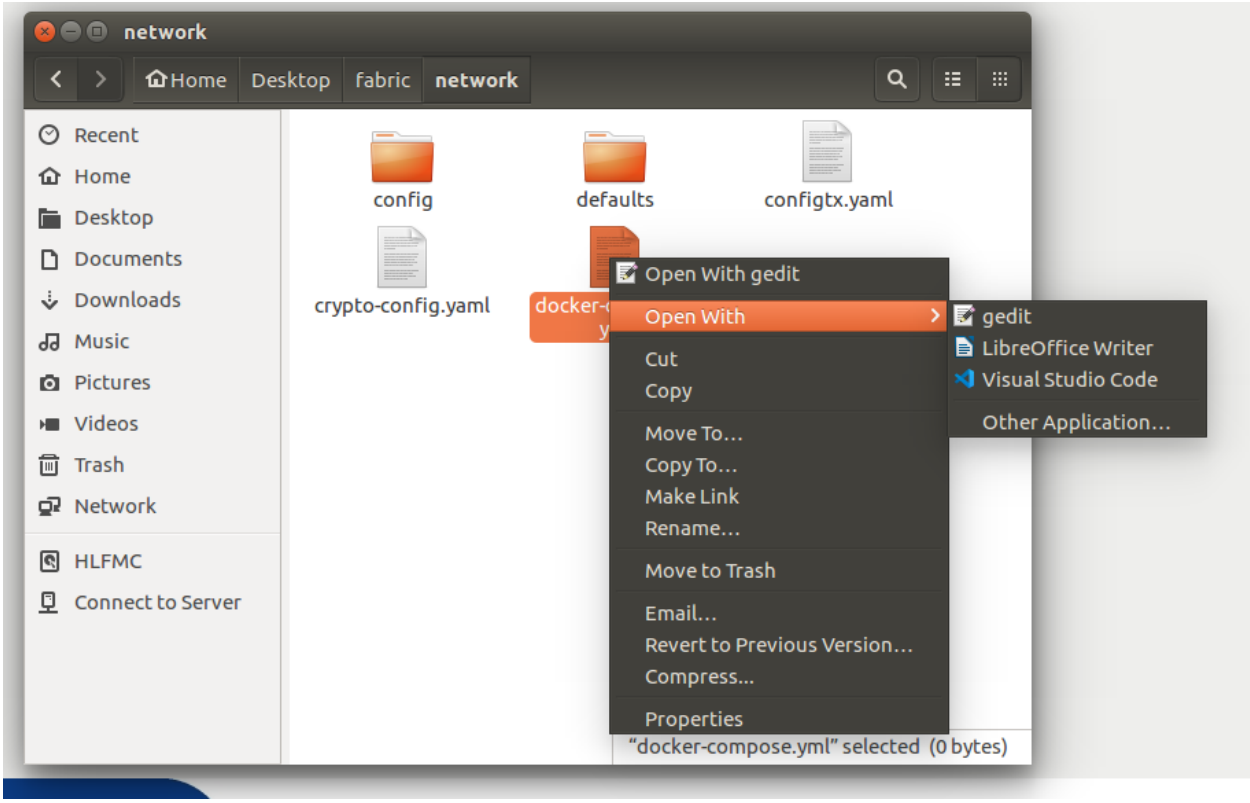

Figure 52 - Opening the docker-compose.yml file with Visual Studio Code

The *docker-compose.yml* file will be empty when first opened. Add the following code into the file.

```
version: '2'

networks:
    btacoin:

services:
```

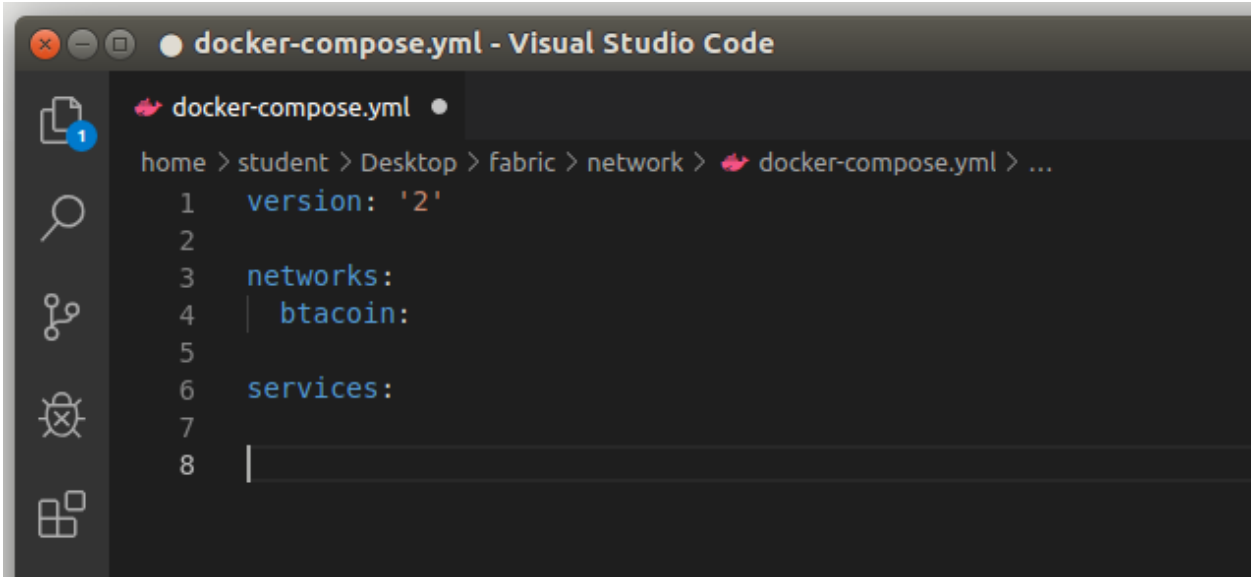

Figure 53 - Snippet 2.15

In any Fabric network, two containers are always required - the Orderer container and the Certificate Authority (CA) container. The Certificate Authority is used to manage membership while the Orderer is used to deliver transactions. Add the following code to your configuration file under the *services:* line to define the CA container.

```
GeneralCA.btacoin.com:
    container_name: GeneralCA.btacoin.com
    image: hyperledger/fabric-ca
    environment:
        - FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
    ports:
        - 7054:7054
```

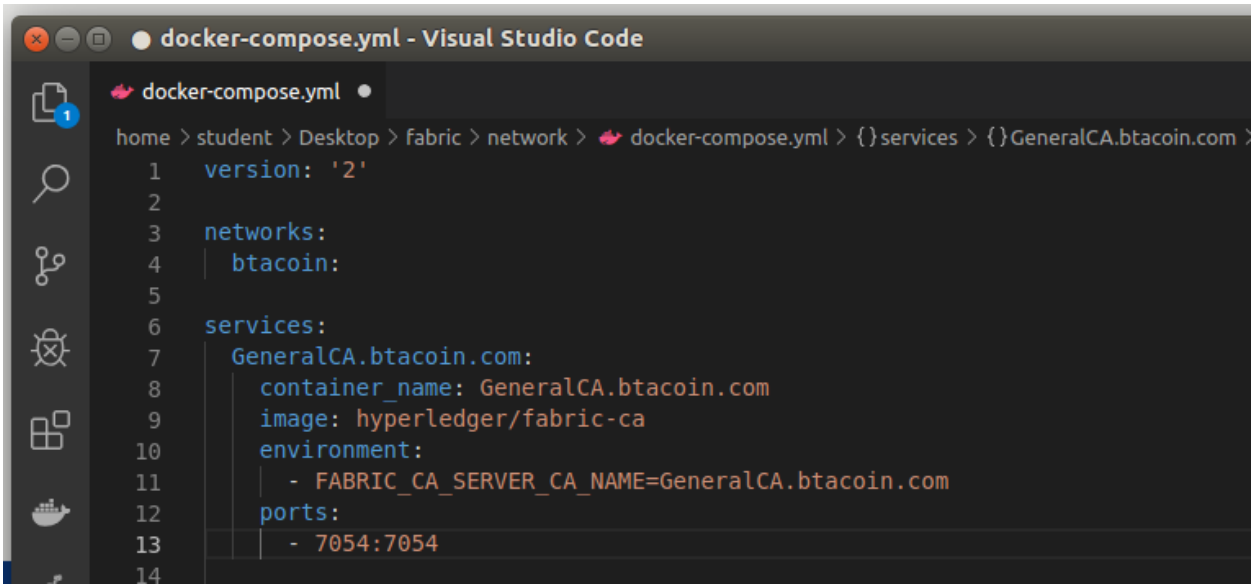

Figure 54 - Snippet 2.16

Now add the definition for the Orderer container. Add the code below under the definition for the CA container. Note that the *LISTENADDRESS* property has been set to an open address to allow the Orderer to listen from anywhere.

```
Devorderer.btacoin.com:
    container_name: Devorderer.btacoin.com
    image: hyperledger/fabric-orderer
    environment:
      - FABRIC_LOGGING_SPEC=info
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_GENESISMETHOD=file
    ports:
      - 7050:7050
```

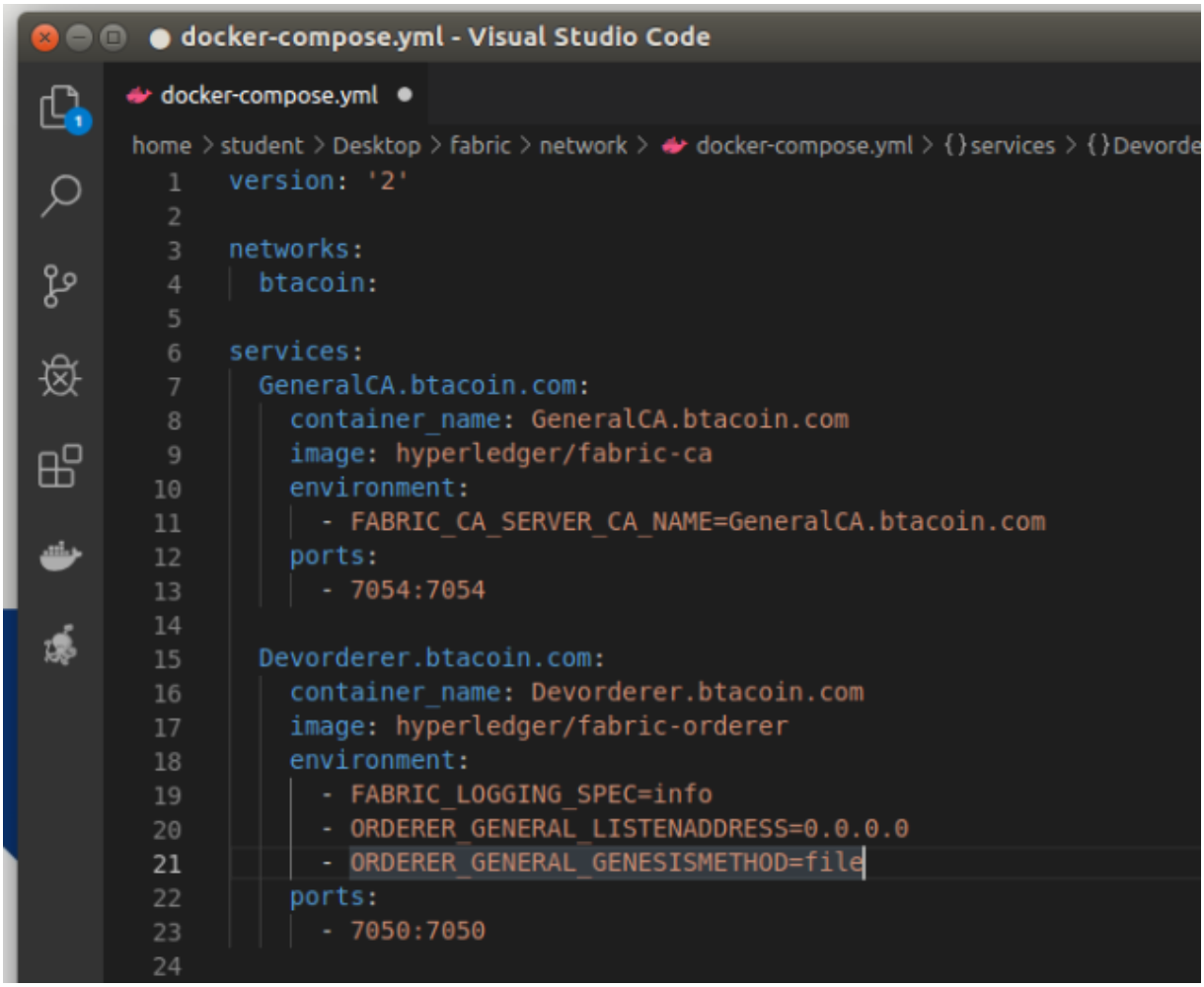

Figure 55 - Snippet 2.17

Now that the CA and Orderer are defined, it's time to move on to defining the peer nodes on the network. The first peer node to be defined will be the Andy.BTA.btacoin.com peer. The naming convention being used is peerName.OrgName.domain.com. Add the definition below to the bottom of your *docker-compose.yml* file to define the Andy peer node.

```
Andy.BTA.btacoin.com:
    container_name: Andy.BTA.btacoin.com
    image: hyperledger/fabric-peer
    environment:
        -
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
        - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    ports:
        - 7051:7051
        - 7053:7053
```

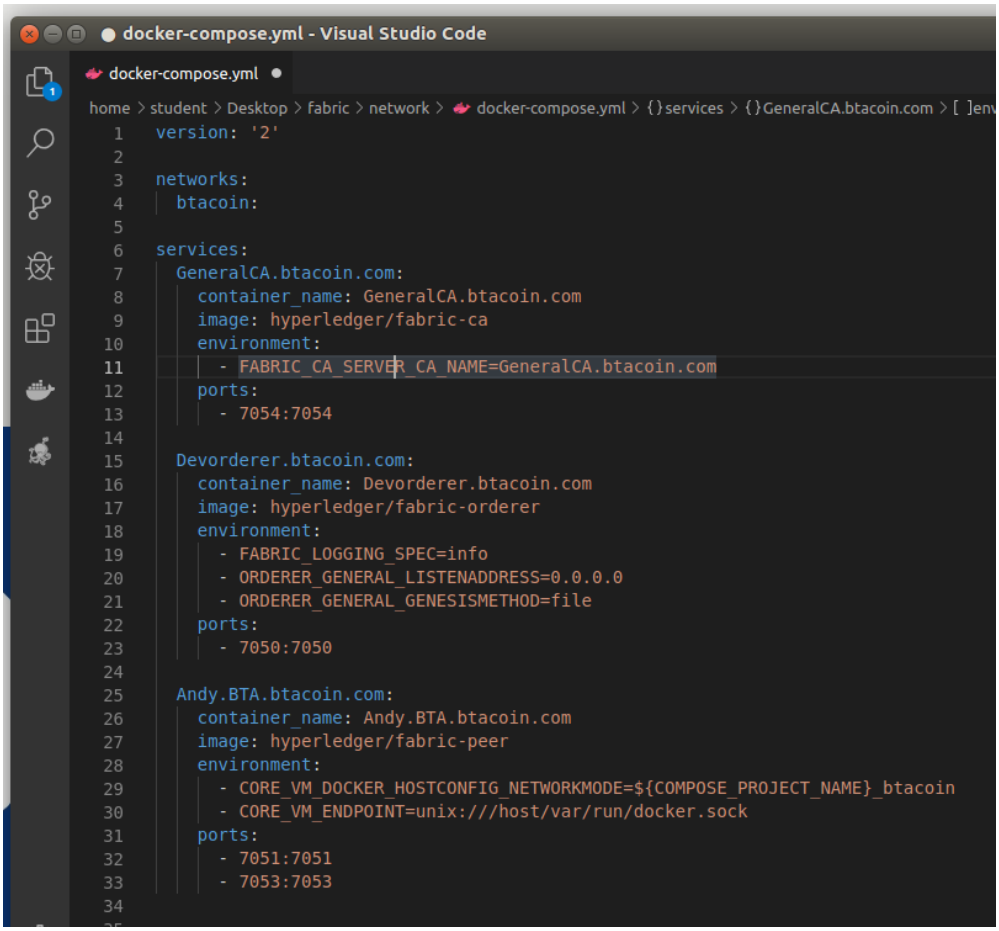

Figure 56 - Snippet 2.18

Now the *cli* (command line interface) container will be defined. This container houses all the Fabric command line interface APIs and libraries. Add the following definition to your *docker-compose.yml* file.

```
cli:
    container_name: cli
    image: hyperledger/fabric-tools
    tty: true
    environment:
        - GOPATH=/opt/gopath/src
        - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
```

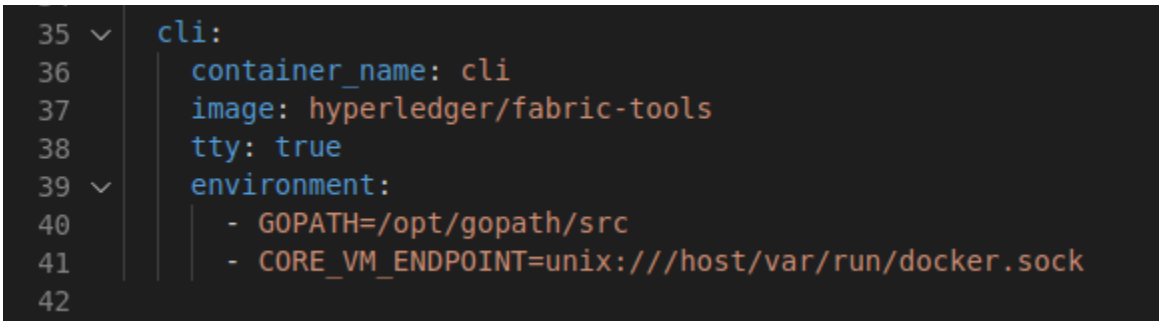

Figure 57 - Snippet 2.19

At this point, you should have a *docker-compose.yml* file which resembles the following: -- 056

```
version: '2'

networks:
  btacoin:

services:
  GeneralCA.btacoin.com:
    container_name: GeneralCA.btacoin.com
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
    ports:
      - 7054:7054

  Devorderer.btacoin.com:
    container_name: Devorderer.btacoin.com
    image: hyperledger/fabric-orderer
    environment:
      - FABRIC_LOGGING_SPEC=info
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_GENESISMETHOD=file
    ports:
      - 7050:7050

  Andy.BTA.btacoin.com:
    container_name: Andy.BTA.btacoin.com
    image: hyperledger/fabric-peer
    environment:
```

```
      -
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    ports:
      - 7051:7051
      - 7053:7053


  cli:
    container_name: cli
    image: hyperledger/fabric-tools
    tty: true
    environment:
      - GOPATH=/opt/gopath/src
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
```

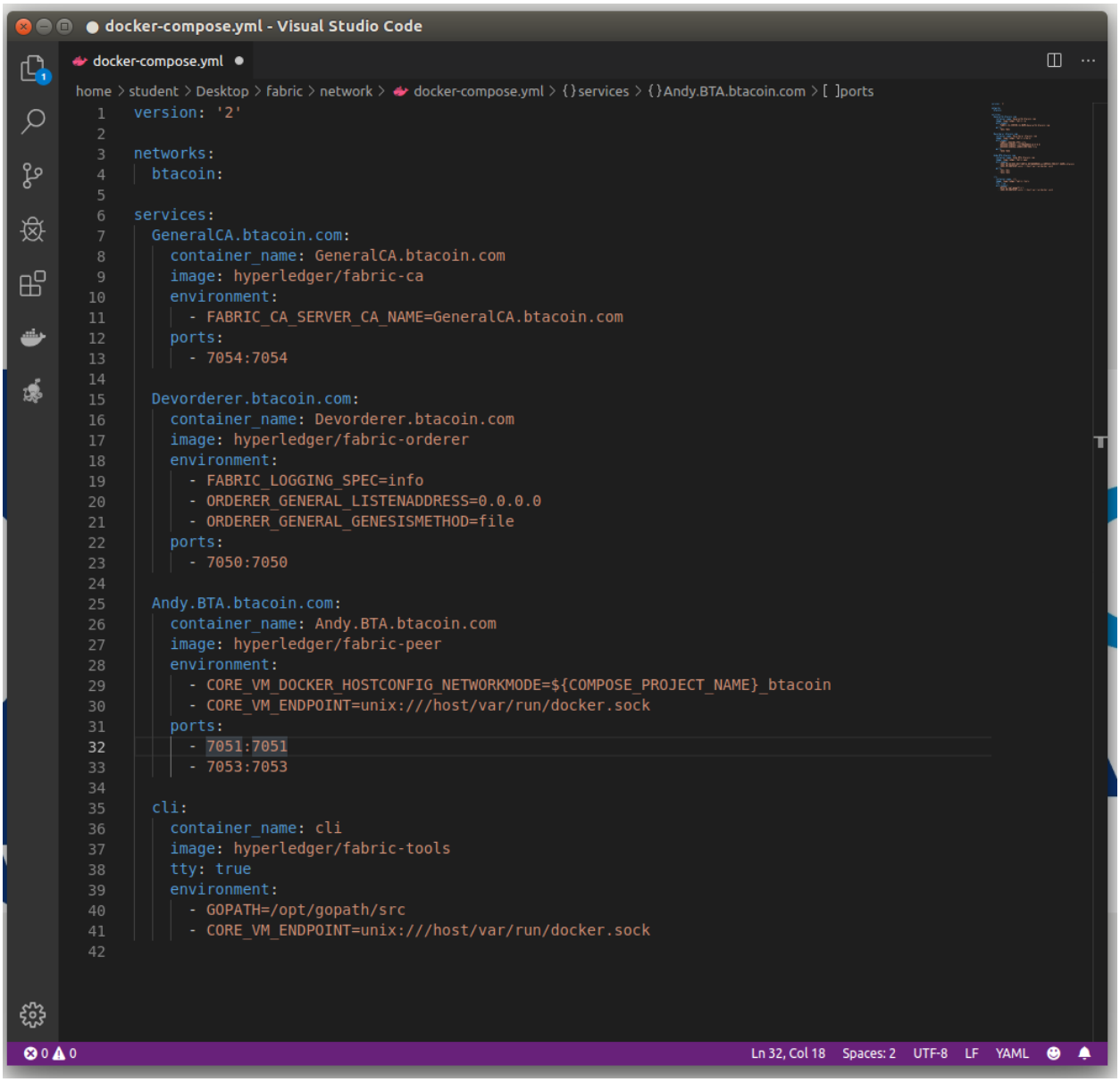

Figure 58 - Snippet 2.20

None of the containers defined so far have their logging output location specified. For the purposes of this lab logging output will be sent to DEBUG to allow for easier monitoring. Append the following line to under the *environment* section of each container definition (*GeneralCA, Devorderer, Andy,* and *cli*).

```
- FABRIC_LOGGING_SPEC=debug
```

```
6   services:
7     GeneralCA.btacoin.com:
8       container_name: GeneralCA.btacoin.com
9       image: hyperledger/fabric-ca
10      environment:
11        - FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
12        - FABRIC_LOGGING_SPEC=debug
13      ports:
14        - 7054:7054
```

Figure 59 - Snippet 2.21

Each container can be a member of one or more Fabric networks, however the current configuration does not specify network membership for any of the defined containers. For the purposes of this lab, each container should be a member of the *btacoin* network. Add the following *network* parameter at the bottom of each container definition.

```
networks:
- btacoin
```

```
6   services:
7     GeneralCA.btacoin.com:
8       container_name: GeneralCA.btacoin.com
9       image: hyperledger/fabric-ca
10      environment:
11        - FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
12        - FABRIC_LOGGING_SPEC=debug
13      ports:
14        - 7054:7054
15      networks:
16        - btacoin
```

Figure 60 - Snippet 2.22

In order to function properly, all peer and cli nodes must have an Ordering Service in place. This means that the container definitions for the Andy peer and the cli have a dependency on the *Devorderer* container. Add the following code to the container definitions for the *Andy* peer as well as the *cli* container to indicate that the *Devorderer* container must be started first.

```
depends_on:
- Devorderer.btacoin.com
```

```
31  Andy.BTA.btacoin.com:
32    container_name: Andy.BTA.btacoin.com
33    image: hyperledger/fabric-peer
34    environment:
35      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
36      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
37      - FABRIC_LOGGING_SPEC=debug
38    ports:
39      - 7051:7051
40      - 7053:7053
41    networks:
42      - btacoin
43    depends_on:
44      - Devorderer.btacoin.com
45
46  cli:
47    container_name: cli
48    image: hyperledger/fabric-tools
49    tty: true
50    environment:
51      - GOPATH=/opt/gopath/src
52      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
53      - FABRIC_LOGGING_SPEC=debug
54    networks:
55      - btacoin
56    depends_on:
57      - Devorderer.btacoin.com
```

Figure 61 - Snippet 2.23

At this point your *docker-compose.yml* file should match the following.

```
version: '2'

networks:
  btacoin:

services:
  GeneralCA.btacoin.com:
    container_name: GeneralCA.btacoin.com
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
      - FABRIC_LOGGING_SPEC=debug
    ports:
      - 7054:7054
    networks:
      - btacoin

  Devorderer.btacoin.com:
    container_name: Devorderer.btacoin.com
    image: hyperledger/fabric-orderer
    environment:
      - FABRIC_LOGGING_SPEC=info
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_GENESISMETHOD=file
      - FABRIC_LOGGING_SPEC=debug
    ports:
      - 7050:7050
    networks:
      - btacoin
```

```
  Andy.BTA.btacoin.com:
    container_name: Andy.BTA.btacoin.com
    image: hyperledger/fabric-peer
    environment:
      -
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - FABRIC_LOGGING_SPEC=debug
    ports:
      - 7051:7051
      - 7053:7053
    networks:
      - btacoin
    depends_on:
      - Devorderer.btacoin.com


  cli:
    container_name: cli
    image: hyperledger/fabric-tools
    tty: true
    environment:
      - GOPATH=/opt/gopath/src
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - FABRIC_LOGGING_SPEC=debug
    networks:
      - btacoin
    depends_on:
      - Devorderer.btacoin.com
```

Figure 62 - Snippet 2.24

Editing the *crypto-config.yaml* file

The *crypto-config.yaml* file defines cryptographic templates which will structure how network node certificates will be output as well as the number of certificates to output for each. Begin by opening the "crypto-config.yaml" located in your Desktop/fabric/network folder in Visual Studio Code.

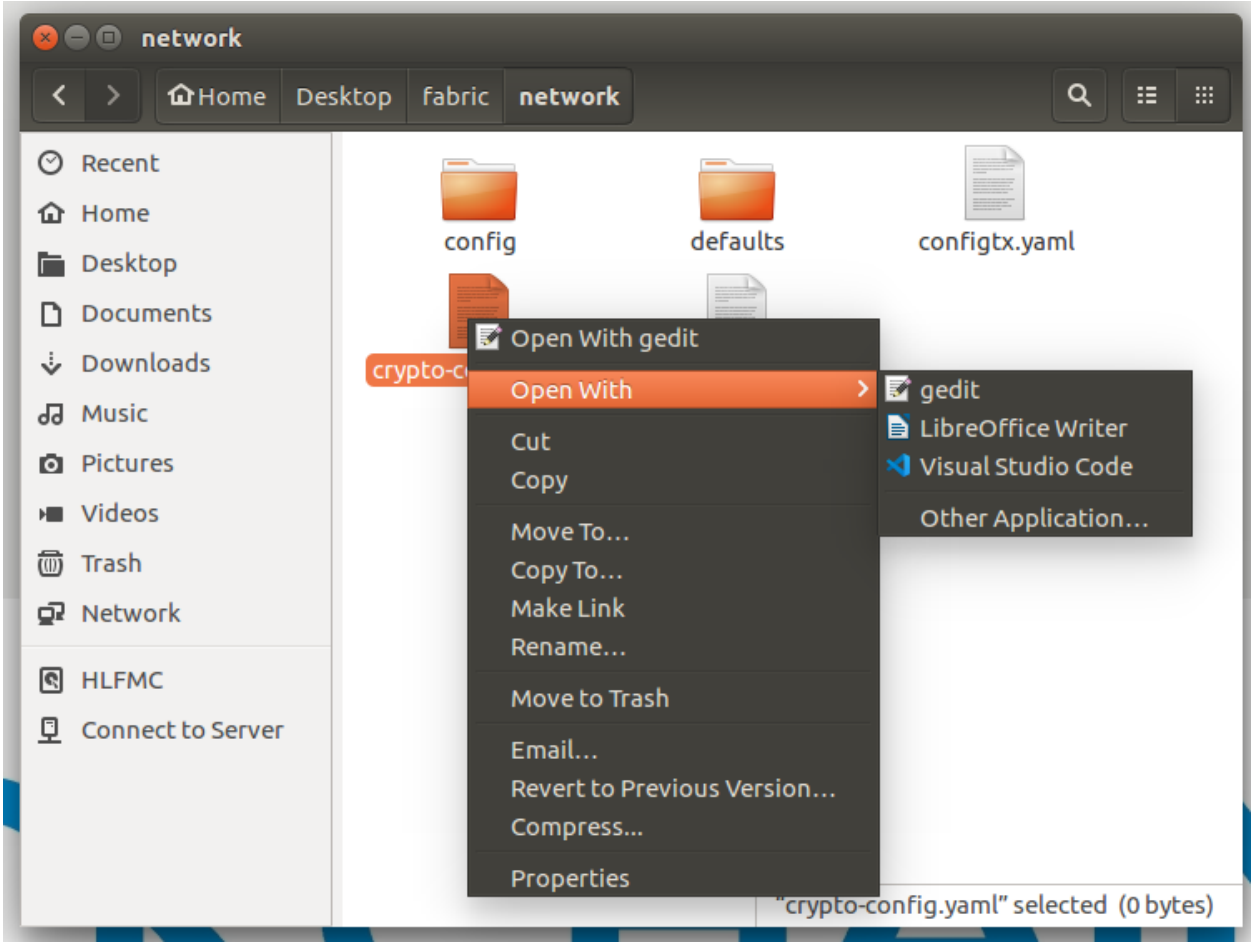

Figure 63 - Opening the crypto-config.yaml file for editing with Visual Studio Code

Begin by adding the following code.

```
OrdererOrgs:


PeerOrgs:
```

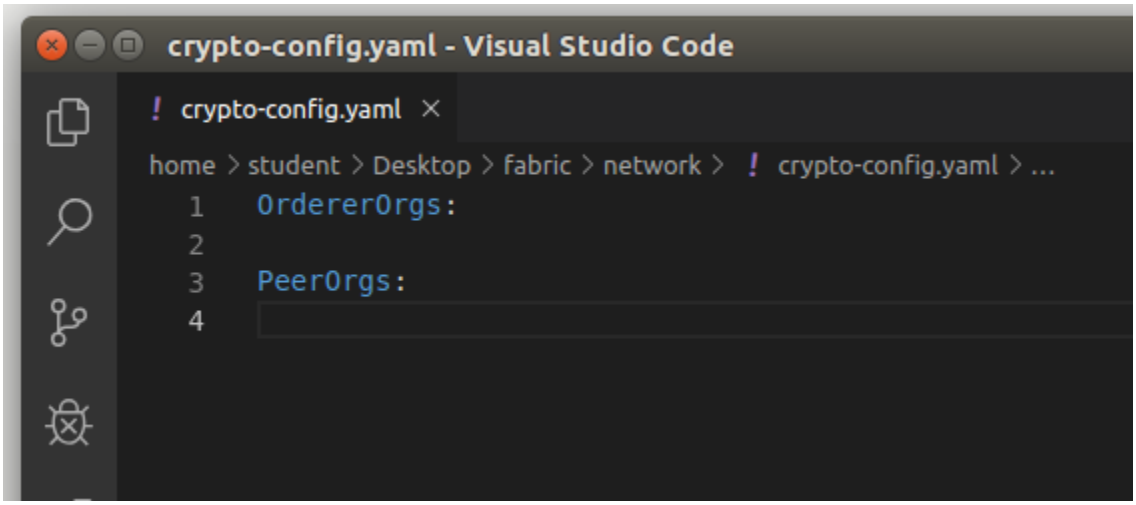

Figure 64 - Snippet 2.25

Under the section *OrdererOrgs* add the following.

```
  - Name: btacoinOrderers
    Domain: btacoin.com
    Specs:
      - Hostname: Devorderer
```

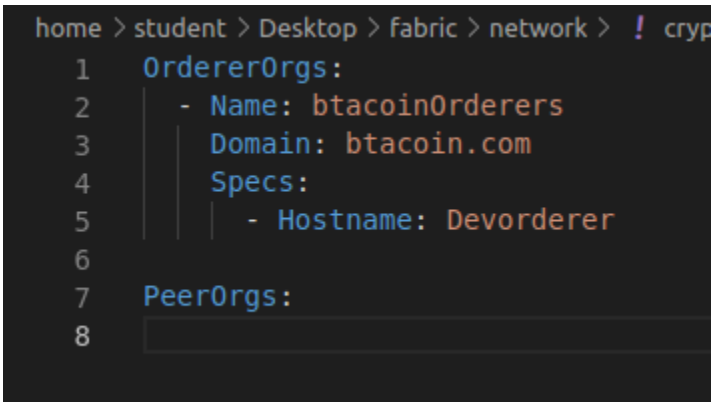

Figure 65 - Snippet 2.26

Under *PeerOrgs* add the following.

```
- Name: BTA

Domain: BTA.btacoin.com

Specs:

- Hostname: Andy
```

```
7   PeerOrgs:
8     - Name: BTA
9       Domain: BTA.btacoin.com
10      Specs:
11        - Hostname: Andy
```

Figure 66 - Snippet 2.27

At this point a template is needed to customize the output of the names on the crypto assets. The only peer defined on the network so far is the Andy peer. As such, we can specify those certificates and leave an administrator as well. The number of desired certificates can be specified under the *Template* key by setting the *Count* property. Since we explicitly defined the only peer's (*Andy*) certificate already, additional peer certs are not needed. However, two more sets of crypto assets can be generated for an Admin and a User account. Add the following code to the bottom of the *PeerOrgs* section.

```
    Template:

      Count: 1

    Users:

      Count: 1
```

```
7   PeerOrgs:
8     - Name: BTA
9       Domain: BTA.btacoin.com
10      Specs:
11        - Hostname: Andy
12      Template:
13        Count: 1
14      Users:
15        Count: 1
16
```

Figure 67 - Snippet 2.28

At this point your *crypto-config.yaml* file should resemble the following.

```
OrdererOrgs:
  - Name: btacoinOrderers
    Domain: btacoin.com
    Specs:
      - Hostname: Devorderer


PeerOrgs:
  - Name: BTA
    Domain: BTA.btacoin.com
    Specs:
      - Hostname: Andy
    Template:
      Count: 1
    Users:
      Count: 1
```

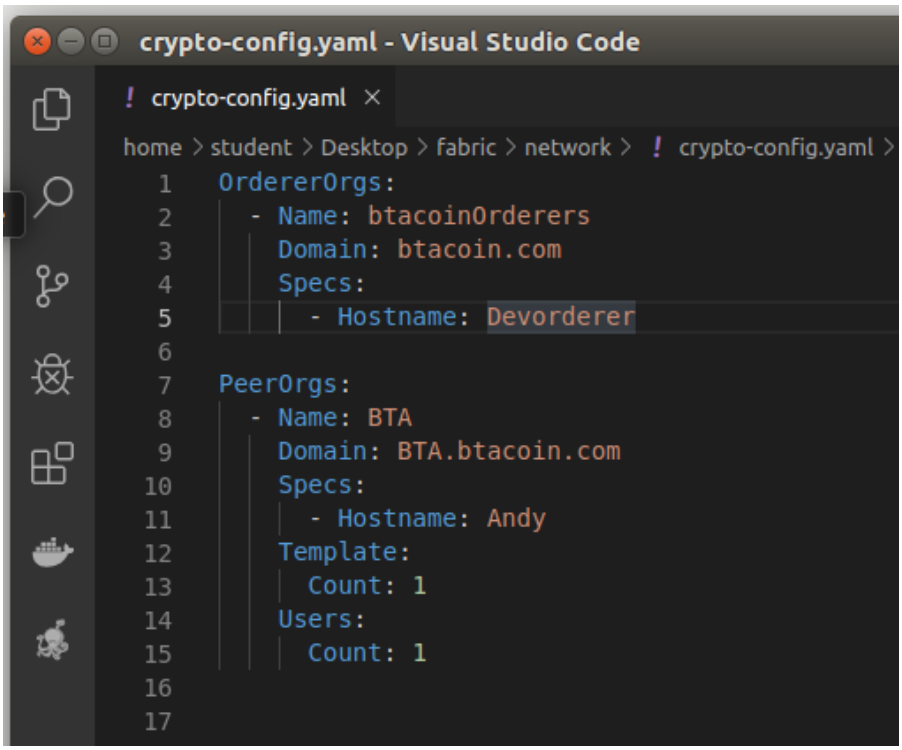

Figure 68 - Snippet 2.29

Editing the *configtx.yaml* file

The *configtx.yaml* file is the heart of network node configuration. From this file definitions for each organization, ordering service, consortium, channel, and client are provided. Begin by opening the empty *configtx.yaml* file in the *Desktop/fabric/network* folder in Visual Studio Code.

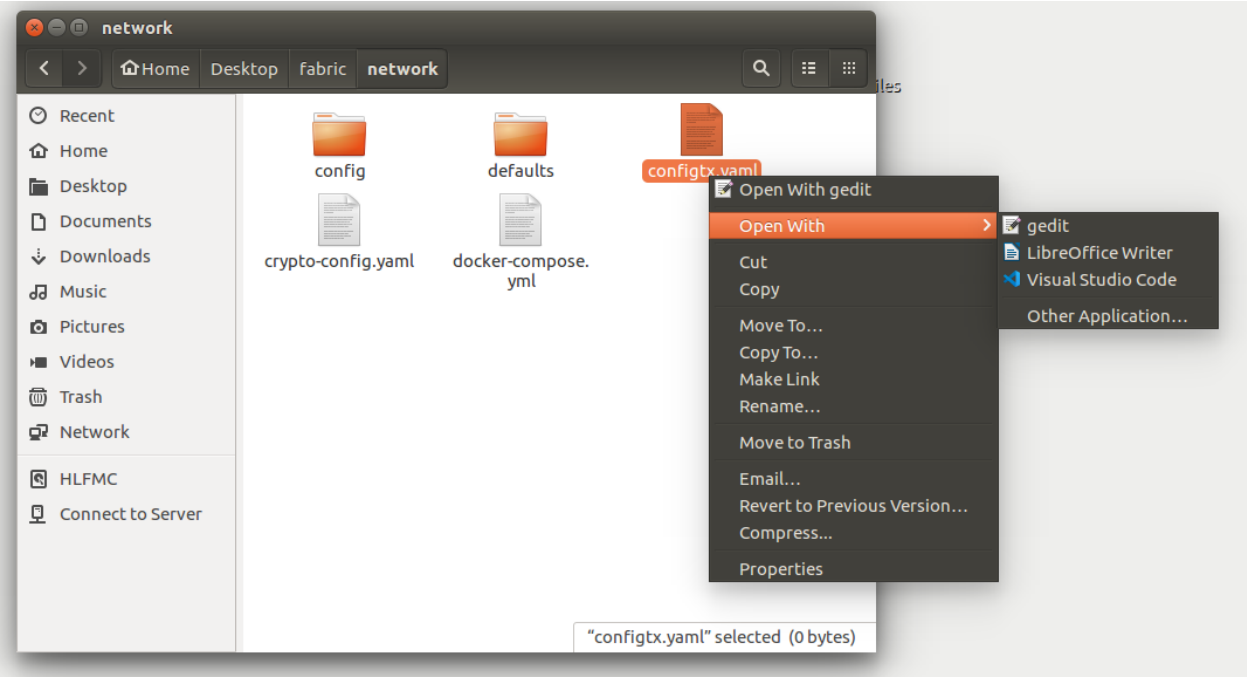

Figure 69 - Opening the configtx.yaml file for editing in Visual Studio Code

The *configtx.yaml* file is comprised of five sections - *Organizations, Application, Orderer, Channel*, and *Profiles*. Create each of those five sections using the code below.

```
Organizations:


Application:


Orderer:


Channel:


Profiles:
```

```
1   Organizations:
2
3   Application:
4
5   Orderer:
6
7   Channel:
8
9   Profiles:
10
11
```

Figure 70 - Snippet 2.30

The *Organizations* section will contain the initial definition of each member organization with corresponding network details such as Membership Service Provider (MSP) information and Anchor Peer details. For the purposes of this lab two organizations will be defined - the Orderer Organization and the BTA organization.

NOTE - This example assumes BTA will host the ordering service for the network, although shared hosting responsibilities between participants or ordering services provided by consortiums are common as well.

Use the code below to define the Orderer Organization (*btacoinOrderers*).

```
- &btacoinOrderers:

  Name: btacoinOrderersMSP
```

configtx.yaml

home > student > Desktop > fabric > network >

```
1   Organizations:
2     - &btacoinOrderers:
3       Name: btacoinOrderersMSP
4
5   Application:
6
7   Orderer:
8
9   Channel:
10
11  Profile:
12
13
```

Figure 71 - Snippet 2.31

To define the Membership Service Provider (MSP) for each organization, a unique ID and path to the MSP must be provided. Use the code below to set the ID and path (*MSPDir*) to the MSP for the *btacoinOrderers* organization.

```
ID: btacoinOrderersMSP

MSPDir: crypto-config/ordererOrganizations/btacoin.com/msp
```

configtx.yaml

home > student > Desktop > fabric > network > configtx.yaml > ...

```
1  Organizations:
2      - &btacoinOrderers:
3        Name: btacoinOrderersMSP
4        ID: btacoinOrderersMSP
5        MSPDir: crypto-config/ordererOrganizations/btacoin.com/msp
6
```

Figure 72 - Snippet 2.32

The same set of information will be provided for the second organization, *BTA*. Use the code below to define the *BTA* organization in the *configtx.yaml* file.

```
- &BTA

  Name: BTA

  ID: BTAMSP

  MSPDir: crypto-config/peerOrganizations/BTA.btacoin.com/msp
```

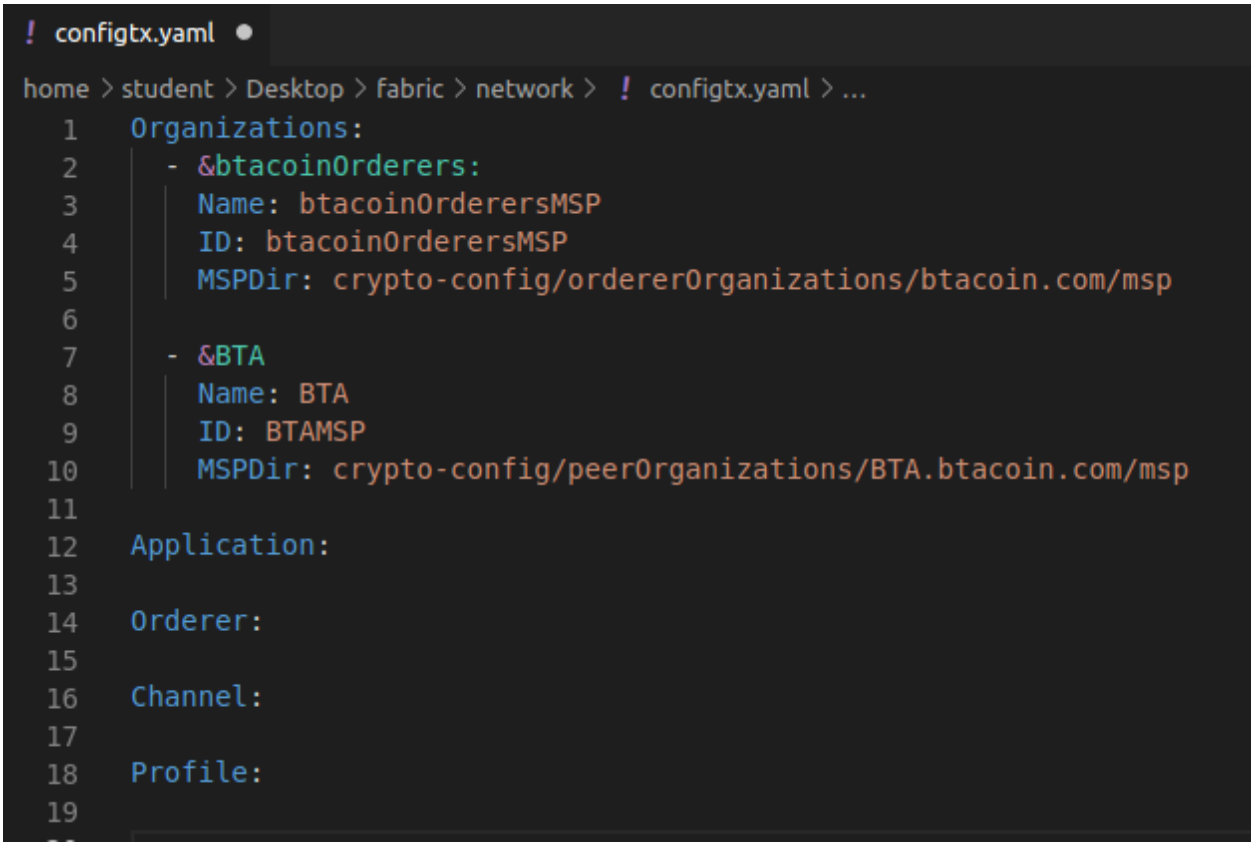

Figure 73 - Snippet 2.33

Note that when defining peer organizations (as opposed to orderer organizations) Anchor Peer information must be provided in the *AnchorPeers* section of the peer definition in *configtx.yaml*. Each Anchor Peer definition requires both the host / domain name (HOST) and port (PORT) where the Anchor Peer can be reached.

Currently there is only one peer defined (*Andy.BTA.btacoin.com*). Use the code below to make that peer an Anchor Peer for the BTA peer organization.

```
AnchorPeers:
  - Host: Andy.BTA.btacoin.com
    Port: 7051
```

```
Organizations:
    - &btacoinOrderers:
        Name: btacoinOrderersMSP
        ID: btacoinOrderersMSP
        MSPDir: crypto-config/ordererOrganizations/btacoin.com/msp

    - &BTA
        Name: BTA
        ID: BTAMSP
        MSPDir: crypto-config/peerOrganizations/BTA.btacoin.com/msp
        AnchorPeers:
            - Host: Andy.BTA.btacoin.com
              Port: 7051
```

Figure 74 - Snippet 2.34

We will skip the Application configuration for now and return to it in later labs. For the time being, please add the code below to the Application section.

```
- &ApplicationDefaults
```

```
15  Application:
16      - &ApplicationDefaults
17
18  Orderer:
19
20  Channel:
21
22  Profile:
23
```

Figure 75 - Snippet 2.35

The *Orderer* section contains the initial configuration required to define the Ordering Service. All Ordering Service definitions must have a uniquely identifiable name property. As this is a development exercise the Ordering Service will be name *DevModeOrdering*, although production Ordering Service implementations would likely differ in name. Use the code below to define the Ordering Service.

```
- &DevModeOrdering
```

```
18 Orderer:
19   - &DevModeOrdering
20
21 Channel:
22
23 Profile:
```

Figure 76 - Snippet 2.36

Each Ordering Service definition must contain a definition for one or more Orderer Nodes. As this is a development exercise you will use the *solo* Ordering Service type. Remember that the *solo* Orderer is intended for development environments only! Production deployments should use Kafka or RAFT with three or more Ordering Service nodes.

Use the code below to define the Ordering Service Type (*OrdererType*) as well as the address for the single Ordering Service node. Note the use of dashed list formatting for addresses, as implementations of Kafka or RAFT can (and should) make use of multiple endpoint addresses.

```
OrdererType: solo

Addresses:

  - Devorderer.btacoin.com:7050
```

```
17
18 Orderer:
19   - &DevModeOrdering
20     OrdererType: solo
21     Addresses:
22       - Devorderer.btacoin.com:7050
23
24 Channel:
```

Figure 77 - Snippet 2.37

The *BatchTimeout* property is used to define the period the Ordering Service will wait before creating a transaction batch. For the purposes of this lab a value of one second will be used, ensuring transactions will be committed as they are received. Production deployments should weigh the need to timely data against the load put against the Ordering Service when designing network implementations and batch timeout values. Use the code below to set a batch timeout value of one second.

```
BatchTimeout: 1s
```

```
18  Orderer:
19    - &DevModeOrdering
20      OrdererType: solo
21      Addresses:
22        - Devorderer.btacoin.com:7050
23      BatchTimeout: 1s
24
```

Figure 78 - Snippet 2.38

The maximum number of transactions in a block can be defined using the *MaxMessageCount* property. For the purposes of this lab a value of one will be used. This will cause the Ordering Service to create a batch for each single transaction, ensuring instant processing and results. This is an appropriate configuration for development scenarios, but architects of production Fabric deployments should consider the value that makes the most sense given the use case, overall solution architecture, and user expectations. Use the code below to set the *MaxMessageCount* to one - note the *MaxMessageSize* key is nested within the parent *BatchSize* key.

```
18  Orderer:
19    - &DevModeOrdering
20      OrdererType: solo
21      Addresses:
22        - Devorderer.btacoin.com:7050
23      BatchTimeout: 1s
24      BatchSize:
25        MaxMessageCount: 1
26
```

Figure 79 – The MaxMessageCount parameter

The *Profiles* section of the *configtx.yaml* file can be used to create one or more pre-defined configuration profiles for channels on a Fabric network. These pre-defined configurations can be used when creating new channels to reduce the amount of configuration and administrative work required. Each profile definition is comprised of a name, a consortium, an application, and an Orderer definition. Use the code below to create a profile for the Orderers Organization (*btacoinOrderers*). This will serve as the genesis block for the network.

```
  DefaultBlockOrderingService:
    Orderer:
      <<: *DevModeOrdering
      Organizations:
        - *btacoinOrderers
    Consortiums:
      NetworkConsortium:
        Organizations:
          - *BTA
```

```
28  Profiles:
29    DefaultBlockOrderingService:
30      Orderer:
31        <<: *DevModeOrdering
32        Organizations:
33          - *btacoinOrderers
34      Consortiums:
35        NetworkConsortium:
36          Organizations:
37            - *BTA
38
```

Figure 80 - Snippet 2.39

Next, define a profile for a channel which contains BTA members only using the code below.

```
btaMembersOnly:
  Consortium: NetworkConsortium
  Application:
    <<: *ApplicationDefaults
    Organizations:
      - *BTA
```

```
28  Profiles:
29    DefaultBlockOrderingService:
30      Orderer:
31        <<: *DevModeOrdering
32        Organizations:
33          - *btacoinOrderers
34        Consortiums:
35          NetworkConsortium:
36            Organizations:
37              - *BTA
```

Figure 81 - Snippet 2.40

Once complete, your *configtx.yaml* file should resemble the following: --

```
Organizations:
  - &btacoinOrderers
    Name: btacoinOrderersMSP
    ID: btacoinOrderersMSP
    MSPDir: crypto-config/ordererOrganizations/btacoin.com/msp

  - &BTA
    Name: BTAMSP
    ID: BTAMSP
    MSPDir: crypto-config/peerOrganizations/BTA.btacoin.com/msp
    AnchorPeers:
        - Host: Andy.BTA.btacoin.com
          Port: 7051

Application:
  - &ApplicationDefaults
    Organizations:

Orderer:
  - &DevModeOrdering
    OrdererType: solo
    Addresses:
      - Devorderer.btacoin.com:7050
    BatchTimeout: 1s
    BatchSize:
      MaxMessageCount: 1

Channel:
```

```
Profiles:
  DefaultBlockOrderingService:
    Orderer:
      <<: *DevModeOrdering
      Organizations:
        - *btacoinOrderers
    Consortiums:
      NetworkConsortium:
        Organizations:
          - *BTA

  btaMembersOnly:
    Consortium: NetworkConsortium
    Application:
      <<: *ApplicationDefaults
      Organizations:
        - *BTA
```

```yaml
Organizations:
  - &btacoinOrderers
    Name: btacoinOrderersMSP
    ID: btacoinOrderersMSP
    MSPDir: crypto-config/ordererOrganizations/btacoin.com/msp

  - &BTA
    Name: BTAMSP
    ID: BTAMSP
    MSPDir: crypto-config/peerOrganizations/BTA.btacoin.com/msp
    AnchorPeers:
        - Host: Andy.BTA.btacoin.com
          Port: 7051

Application:
  - &ApplicationDefaults
    Organizations:

Orderer:
  - &DevModeOrdering
    OrdererType: solo
    Addresses:
      - Devorderer.btacoin.com:7050
    BatchTimeout: 1s
    BatchSize:
      MaxMessageCount: 1

Channel:

Profiles:
  DefaultBlockOrderingService:
    Orderer:
      <<: *DevModeOrdering
      Organizations:
        - *btacoinOrderers
    Consortiums:
      NetworkConsortium:
        Organizations:
          - *BTA

  btaMembersOnly:
    Consortium: NetworkConsortium
    Application:
      <<: *ApplicationDefaults
      Organizations:
        - *BTA
```

Figure 82 - Snippet 2.41

Lab 3 – Starting the Network

In this lab you will generate all the required network artifacts using the configuration completed in the last lab, then start the network.

Building the Network Artifacts

Open a terminal window (if one is not already open) and navigate to the *Desktop/fabric/network* folder.

```
cd ~/Desktop/fabric/network
```

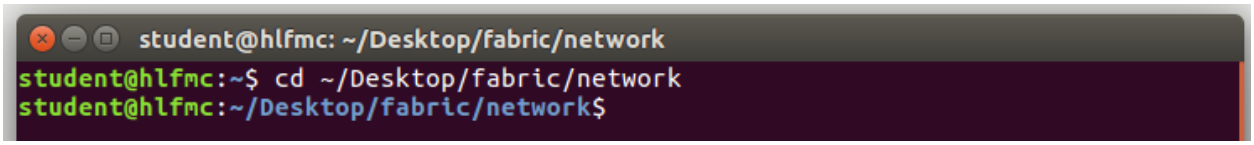

Figure 83 - Snippet 3.1

This lab will make use of the *cryptogen* binary tool included as part of the Fabric toolset. This tool is used to create crypto-certificates based on the template defined in the *crypto-config.yaml* file. This file is specified using the *config* flag. Note that the default behavior for *cryptogen* is to generate a folder and title it *crypto-config*. This behavior can be overridden if desired by using the `--input` flag and specifying a custom location. For the purposes of this lab the default location will be used. Run the following command to generate cryptographic certificates and assets for the network.

```
cryptogen generate --config=./crypto-config.yaml
```

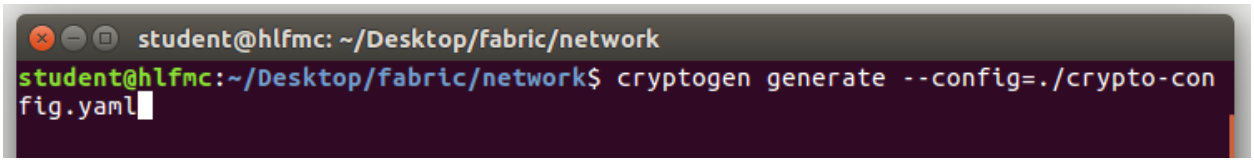

Figure 84 - Snippet 3.2

Now that certificates have been generated it is important to update the container definitions created in the *docker-compose.yml* file. The *GeneralCA* container will need access to the assets created by the *cryptogen* tool in the step above. To enable this access, a volume mapping will be created. This mapping will map the *ca* folder for the *BTA.btacoin.com* organization to the *fabric-ca-server-config* folder on the *GeneralCA* container. Add the code below to the definition of *GeneralCA* in the *docker-composer.yml* file.

```
    volumes:
      - ./crypto-config/peerOrganizations/BTA.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-config
```

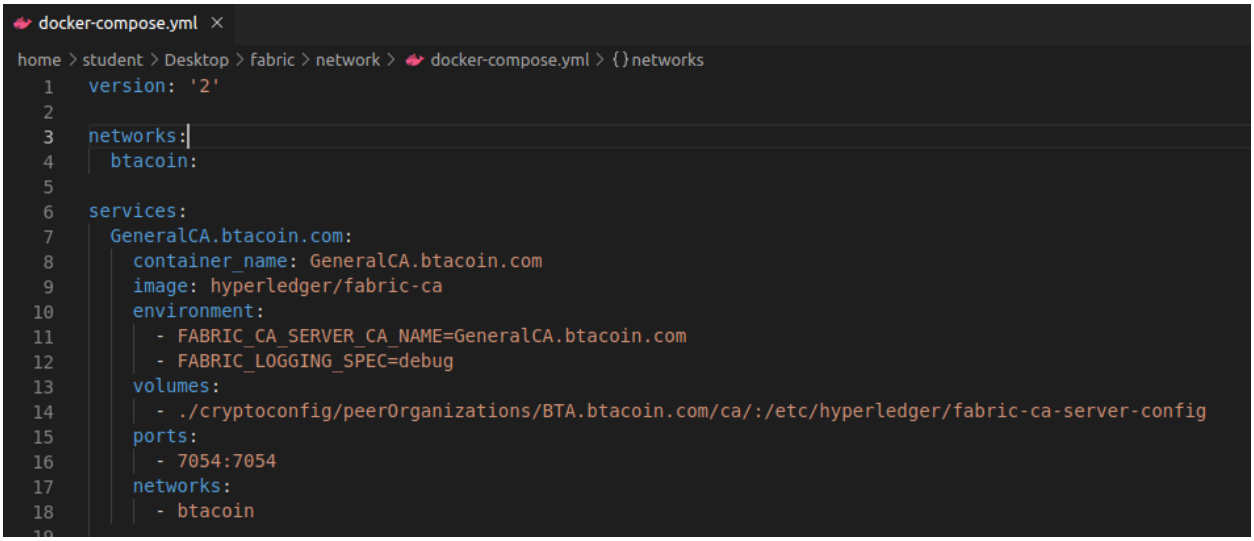

Figure 85 - Snippet 3.3

Each Fabric network will have its own unique *SK* file which gets generated by *cryptogen*. To determine the *SK* file name on your network, run the following command and note the returned filename. In the example pictured here the returned *SK* file name was *7d5e6ace900280ada6ba3a0b7000606270ce6dad497c5645bc583bf86054c164_sk*. **The value you will get will be different. Make note of it for use in the next step.**

```
ls ./crypto-config/peerOrganizations/BTA.btacoin.com/ca
```

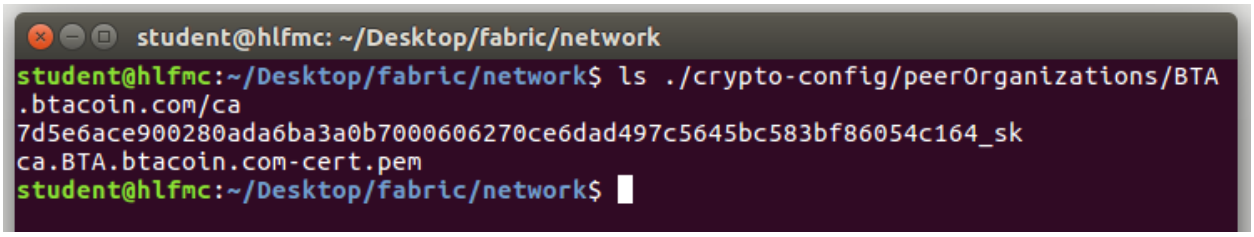

Figure 86 - Snippet 3.4

Editing the docker-compose.yml file

Append the following parameters to the *environment* section of the *GeneralCA* definition in *docker-compose.yml*. **Be sure to use the unique *SK* file name returned in the step above for the second parameter.**

```
      - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-server-config/ca.BTA.btacoin.com-cert.pem

      - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/YOUR_UNIQUE_SK_GOES_HERE
```

```
services:
  GeneralCA.btacoin.com:
    container_name: GeneralCA.btacoin.com
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
      - FABRIC_LOGGING_SPEC=debug
      - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-caserver-config/ca.BTA.btacoin.com-cert.pem
      - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-caserverconfig/7d5e6ace900280ada6ba3a0b7000606270ce6dad497c5645bc583bf86054c164_sk
    volumes:
      - ./cryptoconfig/peerOrganizations/BTA.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-config
    ports:
      - 7054:7054
    networks:
      - btacoin
```

Figure 87 - Snippet 3.5

Append the parameters below to the environment section of the *Devorderer* service definition.

```
      - ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp

      - ORDERER_GENERAL_LOCALMSPID=btacoinOrderersMSP
```

```
21
22    Devorderer.btacoin.com:
23      container_name: Devorderer.btacoin.com
24      image: hyperledger/fabric-orderer
25      environment:
26        - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
27        - ORDERER_GENERAL_GENESISMETHOD=file
28        - FABRIC_LOGGING_SPEC=debug
29        - ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp
30        - ORDERER_GENERAL_LOCALMSPID=btacoinOrderersMSP
31      ports:
32        - 7050:7050
33      networks:
34        - btacoin
```

Figure 88 - Snippet 3.6

Add the parameters below into the *environment* section of the definition of the *cli* container in the *docker-compose.yml* file. These parameters will add pathing to the admin identity (*Andy*) allowing users to perform admin level tasks.

```
- CORE_PEER_ID=cli
- CORE_PEER_ADDRESS=Andy.BTA.btacoin.com:7051
- CORE_PEER_LOCALMSPID=BTAMSP
- CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/BTA.btacoin.com/users/Admin@BTA.btacoin.com/msp
```

```
50
51  cli:
52    container_name: cli
53    image: hyperledger/fabric-tools
54    tty: true
55    environment:
56      - GOPATH=/opt/gopath/src
57      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
58      - FABRIC_LOGGING_SPEC=debug
59      - CORE_PEER_ID=cli
60      - CORE_PEER_ADDRESS=Andy.BTA.btacoin.com:7051
61      - CORE_PEER_LOCALMSPID=BTAMSP
62      - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizatio
63    networks:
64      - btacoin
65    depends_on:
66      - Devorderer.btacoin.com
67
```

Figure 89 - Snippet 3.7

Volumes will now be mapped for the *Devorderer* container definition. Append the following code below the *environment* section of the *Devorderer* definition in the *docker-compose.yml* file.

```
    volumes:
      - ./config/:/etc/hyperledger/configtx
      - ./crypto-config/ordererOrganizations/btacoin.com/orderers/Devorderer.btacoin.com/:/etc/hyperledger/msp/orderer
      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/:/etc/hyperledger/msp/BTA
```

```
22  Devorderer.btacoin.com:
23    container_name: Devorderer.btacoin.com
24    image: hyperledger/fabric-orderer
25    environment:
26      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
27      - ORDERER_GENERAL_GENESISMETHOD=file
28      - FABRIC_LOGGING_SPEC=debug
29      - ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp
30      - ORDERER_GENERAL_LOCALMSPID=btacoinOrderersMSP
31    volumes:
32      - ./config/:/etc/hyperledger/configtx
33      - ./crypto-config/ordererOrganizations/btacoin.com/orderers/Devorderer.btacoin.com/:/etc/hyperledger/msp/orderer
34      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/:/etc/hyperledger/msp/BTA
35    ports:
36      - 7050:7050
37    networks:
38      - btacoin
39
```

Figure 90 - Snippet 3.8

Use the following code to map volumes on the container definition for the *Andy* peer node. Place the code below beneath the *environment* section of the *Andy.BTA.btacoin.com* container definition.

```
volumes:
      - /var/run/:/host/var/run/
      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/msp:/etc/hyperledger/msp/peer
      - ./crypto-config/peerOrganizations/BTA.btacoin.com/users:/etc/hyperledger/msp/users
      - ./config:/etc/hyperledger/configtx
      - ./../cc:/etc/hyperledger/chaincode
```

```
39
40  Andy.BTA.btacoin.com:
41    container_name: Andy.BTA.btacoin.com
42    image: hyperledger/fabric-peer
43    environment:
44      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
45      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
46      - FABRIC_LOGGING_SPEC=debug
47    volumes:
48      - /var/run/:/host/var/run/
49      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/msp:/etc/hyperledger/msp/peer
50      - ./crypto-config/peerOrganizations/BTA.btacoin.com/users:/etc/hyperledger/msp/users
51      - ./config:/etc/hyperledger/configtx
52      - ./../cc:/etc/hyperledger/chaincode
53    ports:
54      - 7051:7051
55      - 7053:7053
56    networks:
57      - btacoin
58    depends_on:
59      - Devorderer.btacoin.com
```

Figure 91 - Snippet 3.9

All default peer container values will be taken from the *core.yaml* file located in the *Desktop/fabric/network/defaults* folder. Any configuration values specified in *docker-compose.yml* will override the defaults coming from *core.yaml*. Several of these default values will need to be overriden in order for the *Andy* peer container to function properly. Add the following code to the *environment* section of the *Andy* container definition in *docker-compose.yml*.

```
- CORE_PEER_ID=Andy.BTA.btacoin.com
- CORE_PEER_ADDRESS=Andy.BTA.btacoin.com:7051
- CORE_PEER_LOCALMSPID=BTAMSP
- CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
```

```
39
40  Andy.BTA.btacoin.com:
41    container_name: Andy.BTA.btacoin.com
42    image: hyperledger/fabric-peer
43    environment:
44      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
45      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
46      - FABRIC_LOGGING_SPEC=debug
47      - CORE_PEER_ID=Andy.BTA.btacoin.com
48      - CORE_PEER_ADDRESS=Andy.BTA.btacoin.com:7051
49      - CORE_PEER_LOCALMSPID=BTAMSP
50      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
51    volumes:
52      - /var/run/:/host/var/run/
53      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/msp:/etc/hyperledger/msp/peer
54      - ./crypto-config/peerOrganizations/Andy.BTA.btacoin.com/users:/etc/hyperledger/msp/users
55      - ./config:/etc/hyperledger/configtx
56      - ./chaincode:/etc/hyperledger/chaincode
57    ports:
58      - 7051:7051
59      - 7053:7053
60    networks:
61      - btacoin
62    depends_on:
63      - Devorderer.btacoin.com
64
```

Figure 92 - Snippet 3.10

At this point, all the necessary certification pathing as well as mapping to the local (container host) configuration folder is complete for the *Andy* peer. The same configuration steps will now be performed against the *cli* container definition. Begin by adding the code below underneath the *environment* section of the *cli* container definition.

```
    volumes:
      - /var/run/:/host/var/run/
      - ./../cc/:/opt/gopath/src/
      - ./crypto-
config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
      - ./config:/etc/hyperledger/configtx
```

```
64
65  cli:
66    container_name: cli
67    image: hyperledger/fabric-tools
68    tty: true
69    environment:
70      - GOPATH=/opt/gopath/src
71      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
72      - FABRIC_LOGGING_SPEC=debug
73      - CORE_PEER_ID=cli
74      - CORE_PEER_ADDRESS=Andy.BTA.btacoin.com:7051
75      - CORE_PEER_LOCALMSPID=BTAMSP
76      - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/BTA.btacoin.com/users/Adm
77    volumes:
78      - /var/run/:/host/var/run/
79      - ./../cc/:/opt/gopath/src/
80      - ./cryptoconfig:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
81      - ./config:/etc/hyperledger/configtx
82    networks:
83      - btacoin
84    depends_on:
85      - Devorderer.btacoin.com
86
```

Figure 93 - Snippet 3.11

Next, volumes will be mapped for the CA container. Add the following code below the *environment* section of the definition for the *GeneralCA* container in the *docker-compose.yml* file.

```
    volumes:
      - ./crypto-config/peerOrganizations/BTA.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-config
```

```
6  services:
7    GeneralCA.btacoin.com:
8      container_name: GeneralCA.btacoin.com
9      image: hyperledger/fabric-ca
10     environment:
11       - FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
12       - FABRIC_LOGGING_SPEC=debug
13       - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-caserver-config/ca.BTA.btacoin.com-cert.pem
14       - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/7d5e6ace900280ada6ba3a0b70006
15     volumes:
16       - ./crypto-config/peerOrganizations/BTA.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-config
17     ports:
18       - 7054:7054
19     networks:
20       - btacoin
```

Figure 94 - Snippet 3.12

Additionally, the CA container will need the *FABRIC_CA_HOME* environment variable set. Add the code below to the *environment* section of the *GeneralCA* container definition.

```
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
```

```
5
6  services:
7    GeneralCA.btacoin.com:
8      container_name: GeneralCA.btacoin.com
9      image: hyperledger/fabric-ca
10     environment:
11       - FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
12       - FABRIC_LOGGING_SPEC=debug
13       - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-caserver-config/ca.BTA.btacoin.com-cert.pem
14       - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/7d5e6ace900280ada6ba3a0b70006
15       - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
16     volumes:
17       - ./crypto-config/peerOrganizations/BTA.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-config
18     ports:
19       - 7054:7054
20     networks:
21       - btacoin
```

Figure 95 - Snippet 3.13

Using the *configtxgen* tool

In this section of the lab you will be using the Configuration Transaction Generator (*configtxgen*) tool to create configuration artifacts required for the initial bootstrap of the network. The *configtxgen* tool will create a Channel Configuration Transaction artifact (*channel.tx*) based on the *Profiles* section from *configtx.yaml*. Note that creating a Channel Transaction does not immediately implement a channel on the network, but is a required first step in channel creation. From a terminal window, call the *configtxgen* tool using the commands below.

```
cd ~/Desktop/fabric/network
```

```
configtxgen -profile DefaultBlockOrderingService -outputBlock
./config/genesis.block -configPath $PWD
```

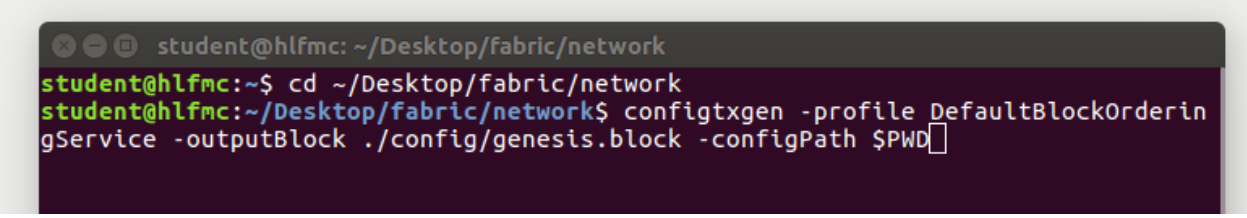

Figure 96 - Snippet 3.14

Next, initiate a *Channel Configuration Transaction* using the *configtxgen* command with the *outputCreateChannelTx* flag by using the command below.

```
configtxgen -profile btaMembersOnly -outputCreateChannelTx
./config/btamembersonly.tx -channelID btamembersonly
```

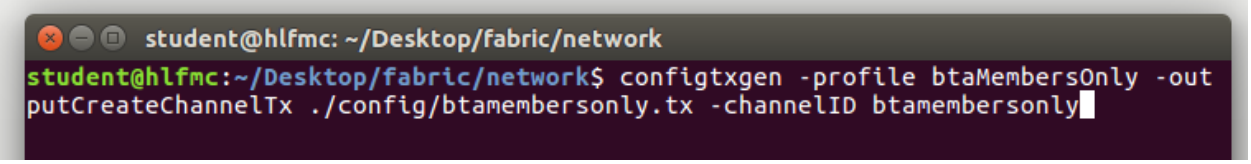

Figure 97 - Snippet 3.15

Now that the network and channel configuration artifacts have been created the next step will be to expose the artifact location to the Orderer and Peer containers on the network. In the *docker-compose.yml* file add the following to the container definition for *Devorderer* under the *environment* section.

```
      - ORDERER_GENERAL_GENESISFILE=/etc/hyperledger/configtx/genesis.block
```

```
22
23  Devorderer.btacoin.com:
24    container_name: Devorderer.btacoin.com
25    image: hyperledger/fabric-orderer
26    environment:
27      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
28      - ORDERER_GENERAL_GENESISMETHOD=file
29      - FABRIC_LOGGING_SPEC=debug
30      - ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp
31      - ORDERER_GENERAL_LOCALMSPID=btacoinOrderersMSP
32      - ORDERER_GENERAL_GENESISFILE=/etc/hyperledger/configtx/genesis.block
33    volumes:
34      - ./config/:/etc/hyperledger/configtx
35      - ./crypto-config/ordererOrganizations/btacoin.com/orderers/Devorderer.btacoin.com/:/etc/hyperledger/msp/orderer
36      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/:/etc/hyperledger/msp/BTA
37    ports:
38      - 7050:7050
39    networks:
40      - btacoin
41
```

Figure 98 - Snippet 3.16

Lab 4 – Working with Peers

In this lab you will install peers, create channels, and manage peer channel membership.

Pulling required Docker containers

For the purposes of this lab you will be working with the pre-built container images provided by the Hyperledger project. Use the *docker pull* commands below to pull down a copy of each container.

```
cd ~/Desktop/fabric/network

docker pull hyperledger/fabric-peer

docker pull hyperledger/fabric-orderer

docker pull hyperledger/fabric-ca

docker pull hyperledger/fabric-tools
```

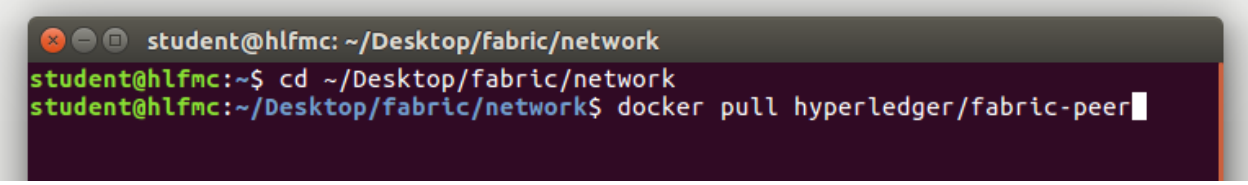

Figure 99 - Snippet 4.1

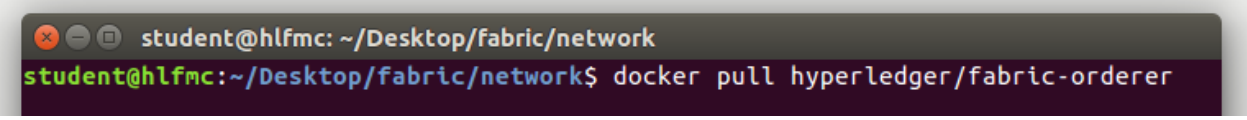

Figure 100 - Snippet 4.2

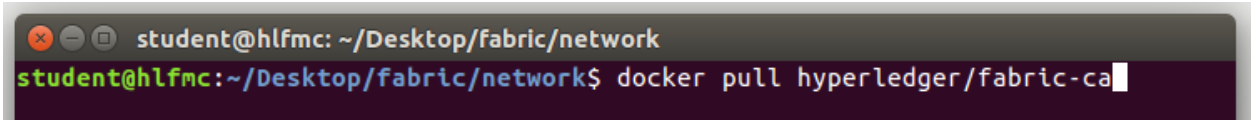

Figure 101 - Snippet 4.3

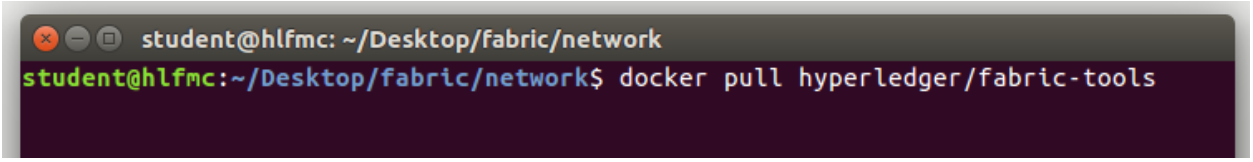

Figure 102 - Snippet 4.4

Use the *docker images* command to verify all images have been pulled down.

```
docker images
```

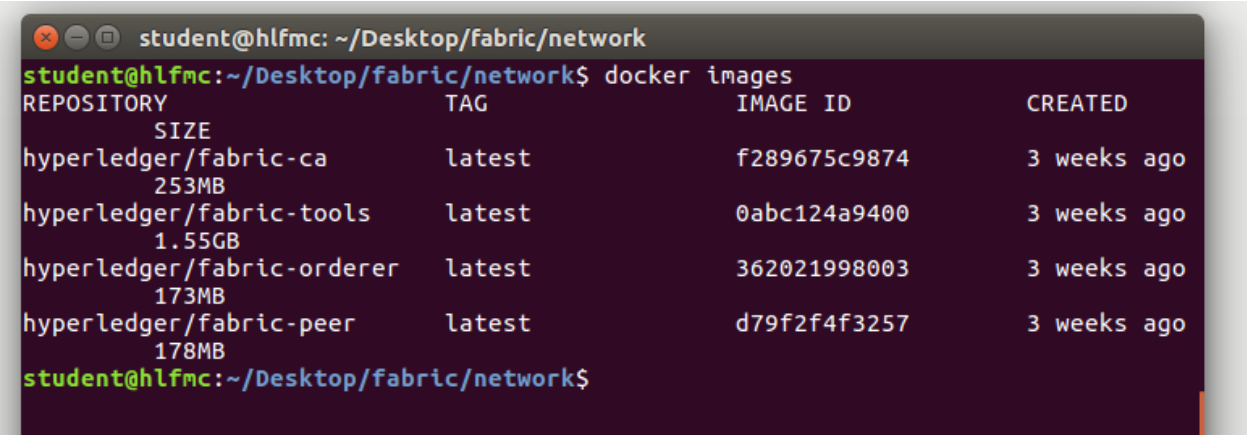

Figure 103 - Snippet 4.5

Editing docker-compose.yml

Each node type in Fabric requires a specific startup command. Startup commands for each container can be placed inside the *docker-compose.yml* file and will be executed upon container startup. If not already open, please open the *docker-compose.yml* file for editing. Inside the container definition for *Devorderer*, beneath the *environment* section append the command below.

```
command: orderer
```

```
23  Devorderer.btacoin.com:
24    container_name: Devorderer.btacoin.com
25    image: hyperledger/fabric-orderer
26    environment:
27      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
28      - ORDERER_GENERAL_GENESISMETHOD=file
29      - FABRIC_LOGGING_SPEC=debug
30      - ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp
31      - ORDERER_GENERAL_LOCALMSPID=btacoinOrderersMSP
32      - ORDERER_GENERAL_GENESISFILE=/etc/hyperledger/configtx/genesis.block
33    command: orderer
34    volumes:
35      - ./config/:/etc/hyperledger/configtx
36      - ./crypto-config/ordererOrganizations/btacoin.com/orderers/Devorderer.btacoin.com/:/etc/hyperledger/msp/orderer
37      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/:/etc/hyperledger/msp/BTA
38    ports:
39      - 7050:7050
40    networks:
41      - btacoin
```

Figure 104 - Snippet 4.6

Add the startup command below to the Andy peer node definition.

```
command: peer node start
```

```
42
43  Andy.BTA.btacoin.com:
44    container_name: Andy.BTA.btacoin.com
45    image: hyperledger/fabric-peer
46    environment:
47      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
48      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
49      - FABRIC_LOGGING_SPEC=debug
50      - CORE_PEER_ID=Andy.BTA.btacoin.com
51      - CORE_PEER_ADDRESS=Andy.BTA.btacoin.com:7051
52      - CORE_PEER_LOCALMSPID=BTAMSP
53      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
54    command: peer node start
55    volumes:
56      - /var/run/:/host/var/run/
57      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/msp:/etc/hyperledger/msp/peer
58      - ./crypto-config/peerOrganizations/Andy.BTA.btacoin.com/users:/etc/hyperledger/msp/users
59      - ./config:/etc/hyperledger/configtx
60      - ./chaincode:/etc/hyperledger/chaincode
61    ports:
62      - 7051:7051
63      - 7053:7053
64    networks:
65      - btacoin
66    depends_on:
67      - Devorderer.btacoin.com
68
```

Figure 105 - Snippet 4.7

Now add the startup command for the *cli* container.

```
command: /bin/bash
```

```
68
69  cli:
70    container_name: cli
71    image: hyperledger/fabric-tools
72    tty: true
73    environment:
74      - GOPATH=/opt/gopath/src
75      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
76      - FABRIC_LOGGING_SPEC=debug
77      - CORE_PEER_ID=cli
78      - CORE_PEER_ADDRESS=Andy.BTA.btacoin.com:7051
79      - CORE_PEER_LOCALMSPID=BTAMSP
80      - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peer
81    command: /bin/bash
82    volumes:
83      - /var/run/:/host/var/run/
84      - ./../cc/:/opt/gopath/src/
85      - ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
86      - ./config:/etc/hyperledger/configtx
87    networks:
88      - btacoin
89    depends_on:
90      - Devorderer.btacoin.com
91
92
```

Figure 106 - Snippet 4.8

Starting the Network

At this point the network can be started using the *docker-compose* command. Note the use of the `-d` flag to indicate containers should consume YAML file settings as well as the `-f` flag used to specify the YAML file which the *docker-compose* command should consume.

```
cd ~/Desktop/fabric/network
```

```
docker-compose -f docker-compose.yml up -d Devorderer.btacoin.com
Andy.BTA.btacoin.com GeneralCA.btacoin.com cli
```

```
student@hlfmc: ~/Desktop/fabric/network
student@hlfmc:~$ cd ~/Desktop/fabric/network
student@hlfmc:~/Desktop/fabric/network$ docker-compose -f docker-compose.yml up
-d Devorderer.btacoin.com Andy.BTA.btacoin.com GeneralCA.btacoin.com cli
```

Figure 107 - Snippet 4.9

```
WARNING: The COMPOSE_PROJECT_NAME variable is not set. Defaulting to a blank str
Creating Devorderer.btacoin.com ... done
Creating network "network_btacoin" with the default driver
Creating Andy.BTA.btacoin.com ... done
Creating Devorderer.btacoin.com ...
Creating cli ...
Creating Andy.BTA.btacoin.com ...
student@hlfmc:~/Desktop/fabric/network$
```

Figure 108 - The output of the docker-compose command

Channel Creation

To start the channel creation process on the network, connect to the Andy peer container using the *docker exec* command below. This command opens a tty connection to the peer container and starts a bash command shell session.

```
docker exec -it Andy.BTA.btacoin.com bash
```

```
root@524f03d38e5b: /
student@hlfmc:~/Desktop/fabric/network$ docker exec -it Andy.BTA.btacoin.com bas
h
root@524f03d38e5b:/#
```

Figure 109 - Snippet 4.10

Use the command below to navigate to the configuration folder.

```
cd /etc/hyperledger/configtx
```

```
root@524f03d38e5b:/# cd /etc/hyperledger/configtx
root@524f03d38e5b:/etc/hyperledger/configtx#
```

Figure 110 - Snippet 4.11

On a Fabric network, channels can only be created by administrators. Use the commands below to set environment variables on the Andy container which will ensure all requests coming from this peer container are signed with an administrative certificate.

```
export CORE_PEER_LOCALMSPID=BTAMSP
```

```
export
CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@BTA.btacoin.com/msp
```

```
root@524f03d38e5b:/etc/hyperledger/configtx# export CORE_PEER_LOCALMSPID=BTAMSP
root@524f03d38e5b:/etc/hyperledger/configtx# export CORE_PEER_MSPCONFIGPATH=/etc
/hyperledger/msp/users/Admin@BTA.btacoin.com/msp
root@524f03d38e5b:/etc/hyperledger/configtx#
```

Figure 111 - Snippet 4.12

Now a new channel can be created on the Fabric network. Use the *peer channel create* command below to create the new channel. Note the use of the `-o` flag to indicate the orderer, the `-f` flag used to tell the command where to find the channel configuration artifact, and the `-c` flag used to specify the channel name.

```
peer channel create -o Devorderer.btacoin.com:7050 -f
/etc/hyperledger/configtx/btamembersonly.tx -c btamembersonly
```

```
root@28b65b541262: /etc/hyperledger/configtx
root@28b65b541262:/etc/hyperledger/configtx# peer channel create -o Devorderer.b
tacoin.com:7050 -f /etc/hyperledger/configtx/btamembersonly.tx -c btamembersonly
```

Figure 112 - Snippet 4.13

Run the following command to list the files in the current directory. The output should contain *btamembersonly.block* as well as *btamembersonly.tx*.

```
ls -l
```

```
root@24713dbafad6: /etc/hyperledger/configtx
root@24713dbafad6:/etc/hyperledger/configtx# ls -l
total 24
-rw-r--r-- 1 root root 9352 Aug 13 23:54 btamembersonly.block
-rw-r--r-- 1 1000 1000  291 Aug 13 23:54 btamembersonly.tx
-rw-r--r-- 1 1000 1000 6469 Aug 13 23:42 genesis.block
root@24713dbafad6:/etc/hyperledger/configtx#
```

Figure 113 - Snippet 4.14

Channel Membership

Now that the channel has been created, peers can be joined to it. To join the *Andy* peer to the *btamembersonly* channel use the following command.

```
peer channel join -b btamembersonly.block
```

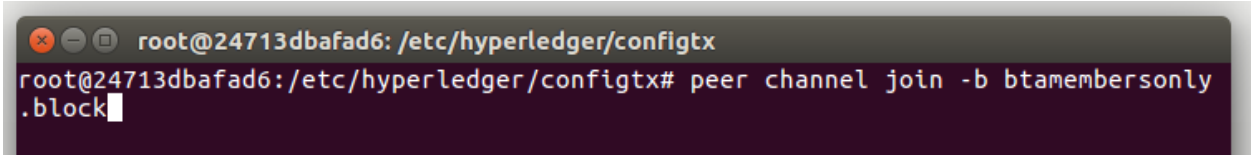

Figure 114 - Snippet 4.15

Verify the peer had joined the channel using the command below.

```
peer channel list
```

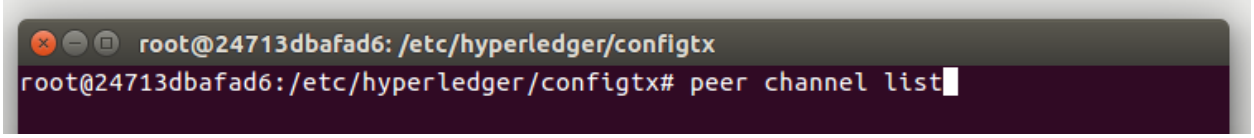

Figure 115 - Snippet 4.16

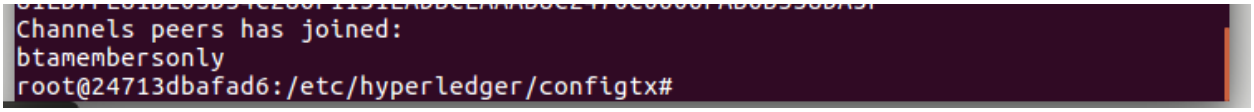

Figure 116 - The output of the peer channel list command

Lab 5 – Managing Organizations

In this lab a new organization will be added to the channel created in the previous lab *btamembersonly*. The organization being created and added to the channel will be called *courseParticipants*. The first course participant to be added will be *Ken*, a student enrolled in Blockchain Architecture.

Creating and Implementing New Organizations

Begin by opening the *docker-compose.yml* file located in the *Desktop/fabric/network* folder. Add the following container definition to the bottom of the file.

```
  Ken.courseParticipants.btacoin.com:
    container_name: Ken.courseParticipants.btacoin.com
    image: hyperledger/fabric-peer
    environment:
      - CORE_PEER_ID= Ken.courseParticipants.btacoin.com
      - CORE_PEER_ADDRESS=Ken.courseParticipants.btacoin.com:8051
      - FABRIC_LOGGING_SPEC=debug
      - CORE_CHAINCODE_LOGGING_LEVEL=debug
      - CORE_PEER_LOCALMSPID=courseParticipantsMSP
      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      -
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
    working_dir: /etc/hyperledger/configtx
    command: peer node start
    ports:
      - 8051:7051
      - 8053:7053
    volumes:
      - /var/run/:/host/var/run/
      - ./crypto-
config/peerOrganizations/courseParticipants.btacoin.com/peers/Ken.cour
seParticipants.btacoin.com/msp:/etc/hyperledger/msp/peers
```

```
      - ./crypto-
config/peerOrganizations/courseParticipants.btacoin.com/users:/etc/hyp
erledger/msp/users

      - ./config:/etc/hyperledger/configtx

    depends_on:

      - Devorderer.btacoin.com

    networks:

      - btacoin
```

```
91
92    Ken.courseParticipants.btacoin.com:
93      container_name: Ken.courseParticipants.btacoin.com
94      image: hyperledger/fabric-peer
95      environment:
96        - CORE_PEER_ID= Ken.courseParticipants.btacoin.com
97        - CORE_PEER_ADDRESS=Ken.courseParticipants.btacoin.com:8051
98        - FABRIC_LOGGING_SPEC=debug
99        - CORE_CHAINCODE_LOGGING_LEVEL=debug
100       - CORE_PEER_LOCALMSPID=courseParticipantsMSP
101       - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
102       - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
103       - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
104     working_dir: /etc/hyperledger/configtx
105     command: peer node start
106     ports:
107       - 8051:7051
108       - 8053:7053
109     volumes:
110       - /var/run/:/host/var/run/
111       - ./crypto-config/peerOrganizations/courseParticipants.btacoin.com/peers/Ken.courseParticipants.btacoin.com/msp:/etc/hyp
112       - ./crypto-config/peerOrganizations/courseParticipants.btacoin.com/users:/etc/hyperledger/msp/users
113       - ./config:/etc/hyperledger/configtx
114     depends_on:
115       - Devorderer.btacoin.com
116     networks:
117       - btacoin
```

Figure 117 - Snippet 5.1

Every organization on a Fabric network must have its own set of cryptographic certificates. As the *courseParticipants* organization is new to the network, the certificates must be generated using the *cryptogen* tool. Open the *crypto-config.yaml* file from the *Desktop/fabric/network* folder. Add the code below underneath the *BTA* organization definition in the *PeerOrgs* section of the file.

```
- Name: courseParticipants

  Domain: courseParticipants.btacoin.com

  Specs:

    - Hostname: Ken

  Template:

    Count: 1

  Users:

    Count: 1
```

```
6
7   PeerOrgs:
8     - Name: BTA
9       Domain: BTA.btacoin.com
10      Specs:
11        - Hostname: Andy
12      Template:
13        Count: 1
14      Users:
15        Count: 1
16    - Name: courseParticipants
17      Domain: courseParticipants.btacoin.com
18      Specs:
19        - Hostname: Ken
20      Template:
21        Count: 1
22      Users:
23        Count: 1
24
```

Figure 118 - Snippet 5.2

Generating artifacts for the new organization

Save all changes the *docker-compose.yml* and *crypto-config.yaml* files. Open a new terminal window (**leave the terminal window connected to the *Andy* peer running**). To open a new terminal window, right-click on the desktop and select *Open Terminal*. From the new terminal window run the following command.

```
cd Desktop/fabric/network
```

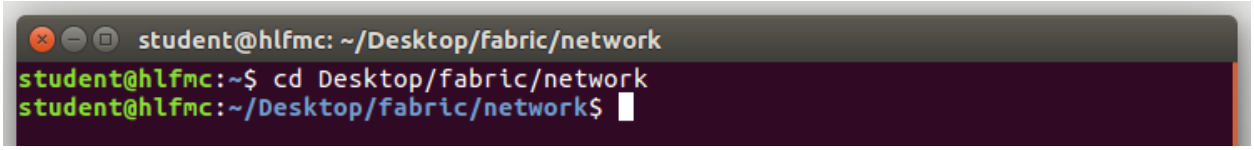

Figure 119 - Snippet 5.3

The *cryptogen* tool will be used to create the new certificates for the *courseParticipants* organization. As certificates for the existing organizations don't need to be changed or re-created, the *extend* flag will be used. Use the command below to create the new certificates.

```
cryptogen extend --config=./crypto-config.yaml
```

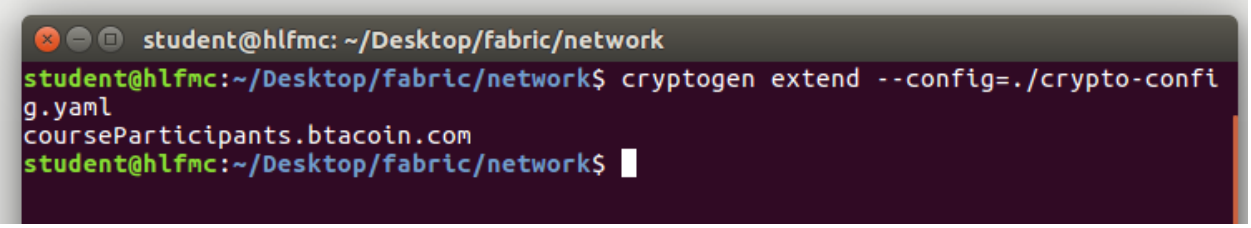

Figure 120 - Snippet 5.4

Confirm the certificates were created by running the following command.

```
ls crypto-config/peerOrganizations/courseParticipants.btacoin.com
```

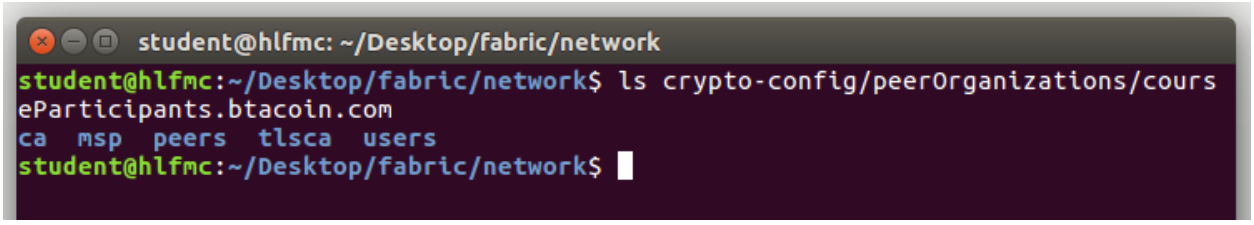

Figure 121 - Snippet 5.5

Editing configtx.yaml

A definition for the new organization, *courseParticipants*, must be added to the *configtx.yaml* file. This will allow information about the new organization to be consumed during channel creation and configuration. Open the *configtx.yaml* file located in the *Desktop/fabric/network* folder. Add the code below to the *Organizations* section underneath the *BTA* organization definition.

```
  - &courseParticipants

    Name: courseParticipantsMSP

    ID: courseParticipantsMSP

    MSPDir: crypto-
config/peerOrganizations/courseParticipants.btacoin.com/msp

    AnchorPeers:

      - Host: Ken.courseParticipants.btacoin.com

        Port: 8051
```

home > student > Desktop > fabric > network > ! configtx.yaml > Channel

```
1   Organizations:
2     - &btacoinOrderers
3       Name: btacoinOrderersMSP
4       ID: btacoinOrderersMSP
5       MSPDir: crypto-config/ordererOrganizations/btacoin.com/msp
6
7     - &BTA
8       Name: BTAMSP
9       ID: BTAMSP
10      MSPDir: crypto-config/peerOrganizations/BTA.btacoin.com/msp
11      AnchorPeers:
12          - Host: Andy.BTA.btacoin.com
13            Port: 7051
14
15    - &courseParticipants
16      Name: courseParticipantsMSP
17      ID: courseParticipantsMSP
18      MSPDir: crypto-config/peerOrganizations/courseParticipants.btacoin.com/msp
19      AnchorPeers:
20        - Host: Ken.courseParticipants.btacoin.com
21          Port: 8051
```

Figure 122 - Snippet 5.6

A new container must be defined on the network to act as the Certificate Authority for the new *courseParticipants* organization. In the *docker-compose.yml* file add the following code to begin the new container definition.

```
courseParticipantsCA.btacoin.com:
  container_name: courseParticipantsCA
  image: hyperledger/fabric-ca
  environment:
  - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
  - FABRIC_CA_SERVER_CA_NAME= courseParticipantsCA.btacoin.com
```

```
118
119   courseParticipantsCA.btacoin.com:
120     container_name: courseParticipantsCA
121     image: hyperledger/fabric-ca
122     environment:
123     - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
124     - FABRIC_CA_SERVER_CA_NAME= courseParticipantsCA.btacoin.com
```

Figure 123 - Snippet 5.7

Setting required environment variables

In order to properly set the next two required environment variables in the new *courseParticipantsCA* container definition two values will need to be retrieved from the *crypto-config* folder. Run the command below from a terminal window, and take note of the *_sk* and *.pem* filenames listed. These values will be needed for the next two environment variables.

```
ls crypto-config/peerOrganizations/courseParticipants.btacoin.com/ca
```

```
student@hlfmc: ~/Desktop/fabric/network
student@hlfmc:~/Desktop/fabric/network$ ls crypto-config/peerOrganizations/cours
eParticipants.btacoin.com/ca
5fcee588499b8ce7ed5242f535da1d927e4b4a7499ef64adc6792ea1fa149bb8_sk
ca.courseParticipants.btacoin.com-cert.pem
student@hlfmc:~/Desktop/fabric/network$
```

Figure 124 - Snippet 5.8

Add the rest of the container definition for *courseParticipantsCA* using the code below. Make sure to replace *PEM_FILE_NAME_GOES_HERE* and *SK_FILE_NAME_GOES_HERE* with the values you obtained in the last step.

```
      - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-
server-config/PEM_FILE_NAME_GOES_HERE

      - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-
config/SK_FILE_NAME_GOES_HERE

    ports:

      - "8054:7054"

    command: sh -c 'fabric-ca-server start -b
courseParticipantsCA:SimplePassword'

    volumes:

      - ./crypto-config/peerOrganizations/
courseParticipants.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-
config

    networks:

      - btacoin
```

```
118
119   courseParticipantsCA.btacoin.com:
120     container_name: courseParticipantsCA
121     image: hyperledger/fabric-ca
122     environment:
123       - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
124       - FABRIC_CA_SERVER_CA_NAME= courseParticipantsCA.btacoin.com
125       - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-caserver-config/PEM_FILE_NAME_GOES_HERE
126       - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-caserver-config/SK_FILE_NAME_GOES_HERE
127     ports:
128       - "8054:7054"
129     Command: sh -c 'fabric-ca-server start -b courseParticipantsCA:SimplePassword'
130     volumes:
131       - ./crypto-config/peerOrganizations/  courseParticipants.btacoin.com/ca/:/etc/hyperledger/fabric-caserver-config
132     networks:
133       - btacoin
```

Figure 125 - Snippet 5.9

The original network definition contained one CA container called *GeneralCA.btacoin.com*. At this point it would be more appropriate to rename it to something more specific to the *BTA* organization (the CA for the *courseParticipants* organization was created over the past two steps). Update the Service Definition and Container Name values using the code below.

`btaCA.btacoin.com:`

`container_name: btaCA.btacoin.com`

```
services:
  btaCA.btacoin.com:
    container_name: btaCA.btacoin.com
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_SERVER_CA_NAME=btaCA.btacoin.com
      - FABRIC_LOGGING_SPEC=debug
      - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-caserver-config/ca.BTA.btacoin.com-cert.pem
      - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/7d5e6ace900280ada6ba3a0b7000606270ce6dad49
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
    volumes:
      - ./crypto-config/peerOrganizations/BTA.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-config
    ports:
      - 7054:7054
    networks:
      - btacoin
```

Figure 126 - Snippet 5.10

Adjust the environment variable

```
- FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
```

to

```
- FABRIC_CA_SERVER_CA_NAME=btaCA.btacoin.com
```

```
5
6   services:
7     GeneralCA.btacoin.com:
8       container_name: GeneralCA.btacoin.com
9       image: hyperledger/fabric-ca
10      environment:
11        - FABRIC_CA_SERVER_CA_NAME=GeneralCA.btacoin.com
12        - FABRIC_LOGGING_SPEC=debug
13        - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-caserver-config/ca.BTA.btacoin.com-cert.pem
14        - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/7d5e6ace900280ada6ba3a0b7000606270ce6dad497c5645b
15        - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
16      volumes:
17        - ./crypto-config/peerOrganizations/BTA.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-config
18      ports:
19        - 7054:7054
20      networks:
21        - btacoin
```

Figure 127 - BEFORE making the change

```
6   services:
7     btaCA.btacoin.com:
8       container_name: btaCA.btacoin.com
9       image: hyperledger/fabric-ca
10      environment:
11        - FABRIC_CA_SERVER_CA_NAME=btaCA.btacoin.com
12        - FABRIC_LOGGING_SPEC=debug
13        - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-caserver-config/ca.BTA.btacoin.com-cert.pem
14        - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/7d5e6ace900280ada6ba3a0b7000606270ce6dad4
15        - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
16      volumes:
17        - ./crypto-config/peerOrganizations/BTA.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-config
18      ports:
19        - 7054:7054
20      networks:
21        - btacoin
```

Figure 128 - Snippet 5.11

Apply the changes

Save all updated YAML configuration files. From the command prompt, run the following command to remove the *GeneralCA.btacoin.com* container.

```
docker rm -f GeneralCA.btacoin.com
```

```
student@hlfmc: ~/Desktop/fabric/network
student@hlfmc:~/Desktop/fabric/network$ docker rm -f GeneralCA.btacoin.com
GeneralCA.btacoin.com
student@hlfmc:~/Desktop/fabric/network$
```

Figure 129 - Snippet 5.12

Run the following command to bring up the two new containers defined during this lab, *courseParticipantsCA.btacoin.com* and *btaCA.btacoin.com*.

```
docker-compose -f docker-compose.yml up -d
courseParticipantsCA.btacoin.com btaCA.btacoin.com
```

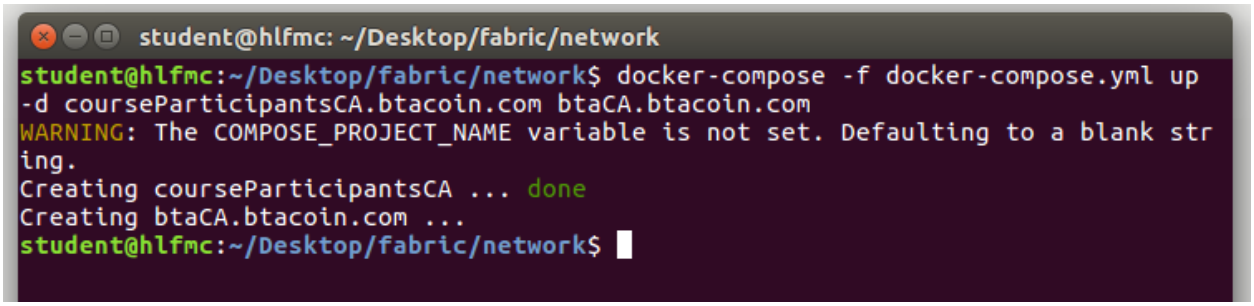

Figure 130 - Snippet 5.13

Lab 6 – Updating Network Configurations

In this lab you will complete the following tasks:

- Update the configuration of the Ordering Service to use Kafka rather than Solo.
- Update the World State Database configuration to use CouchDB rather than levelDB.
- Add additional Peers to the Network.

Updating the Ordering Service Configuration

Updating the Ordering Service configuration will require making edits in both the *docker-compose.yml* file as well as the *configtx.yaml* file. Open both of these files for editing in Visual Studio Code.

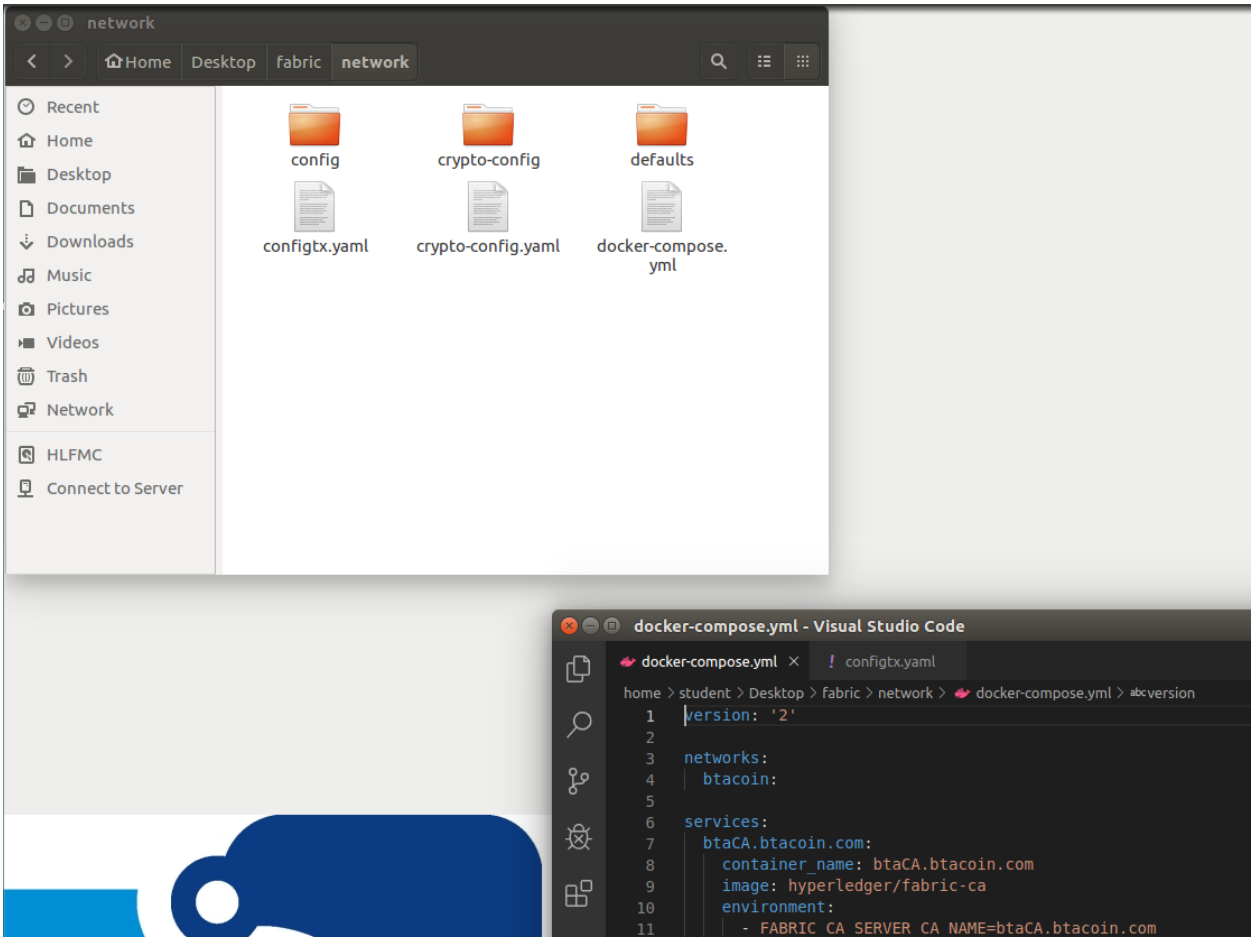

Figure 131 - Open configtx.yml for editing in Visual Studio Code

In the *docker-compose.yml* file, additional container definitions will be provided. These containers will be used to host the Kafka nodes. Two different nodes on ports 8092 and 9092 will be created using the container names *kafkaServer1* and *kafkaServer2*. Both containers will use the *fabric-kafka* container image, which will be automatically pulled down upon container startup. Add the following container definitions for *kafkaServer1* and *kafkaServer2*.

```
  kafkaServer1.btacoin.com:
    container_name: kafkaServer1.btacoin.com
    image: hyperledger/fabric-kafka
    restart: always
    environment:
      - KAFKA_MESSAGE_MAX_BYTES=103809024
      - KAFKA_REPLICA_FETCH_MAX_BYTES=103809024
      - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
      - KAFKA_MIN_INSYNC_REPLICAS=2
      - KAFKA_DEFAULT_REPLICATION_FACTOR=2
      -
KAFKA_ZOOKEEPER_CONNECT=zkServer1.btacoin.com:2181,zkServer2.btacoin.com:2181,zkServer3.btacoin.com:2181
      - KAFKA_BROKER_ID=0
    ports:
      - 10092:9092
      - 10093:9093
    depends_on:
      - zkServer1.btacoin.com
      - zkServer2.btacoin.com
      - zkServer3.btacoin.com
    networks:
      - btacoin


  kafkaServer2.btacoin.com:
    container_name: kafkaServer2.btacoin.com
    image: hyperledger/fabric-kafka
```

```
    restart: always
    environment:
      - KAFKA_MESSAGE_MAX_BYTES=103809024
      - KAFKA_REPLICA_FETCH_MAX_BYTES=103809024
      - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
      - KAFKA_MIN_INSYNC_REPLICAS=2
      - KAFKA_DEFAULT_REPLICATION_FACTOR=2
      -
KAFKA_ZOOKEEPER_CONNECT=zkServer1.btacoin.com:2181,zkServer2.btacoin.c
om:2181,zkServer3.btacoin.com:2181
      - KAFKA_BROKER_ID=1
    ports:
      - 9092:9092
      - 9093:9093
    depends_on:
      - zkServer1.btacoin.com
      - zkServer2.btacoin.com
      - zkServer3.btacoin.com
    networks:
      - btacoin
```

```
134
135   kafkaServer1.btacoin.com:
136     container_name: kafkaServer1.btacoin.com
137     image: hyperledger/fabric-kafka
138     restart: always
139     environment:
140       - KAFKA_MESSAGE_MAX_BYTES=103809024
141       - KAFKA_REPLICA_FETCH_MAX_BYTES=103809024
142       - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
143       - KAFKA_MIN_INSYNC_REPLICAS=2
144       - KAFKA_DEFAULT_REPLICATION_FACTOR=2
145       - KAFKA_ZOOKEEPER_CONNECT=zkServer1.btacoin.com:2181,zkServer2.btacoin.com:2181,zkServer3.btacoin.com:2181
146       - KAFKA_BROKER_ID=0
147     ports:
148       - 10092:9092
149       - 10093:9093
150     depends_on:
151       - zkServer1.btacoin.com
152       - zkServer2.btacoin.com
153       - zkServer3.btacoin.com
154     networks:
155       - btacoin
156
```

Figure 132 - Snippet 6.1

```
156
157   kafkaServer2.btacoin.com:
158     container_name: kafkaServer2.btacoin.com
159     image: hyperledger/fabric-kafka
160     restart: always
161     environment:
162       - KAFKA_MESSAGE_MAX_BYTES=103809024
163       - KAFKA_REPLICA_FETCH_MAX_BYTES=103809024
164       - KAFKA_UNCLEAN_LEADER_ELECTION_ENABLE=false
165       - KAFKA_MIN_INSYNC_REPLICAS=2
166       - KAFKA_DEFAULT_REPLICATION_FACTOR=2
167       - KAFKA_ZOOKEEPER_CONNECT=zkServer1.btacoin.com:2181,zkServer2.btacoin.com:2181,zkServer3.btacoin.com:2181
168       - KAFKA_BROKER_ID=1
169     ports:
170       - 9092:9092
171       - 9093:9093
172     depends_on:
173       - zkServer1.btacoin.com
174       - zkServer2.btacoin.com
175       - zkServer3.btacoin.com
176     networks:
177       - btacoin
```

Figure 133 - Snippet 6.1

The *Zookeeper* service manages the Kafka distributed service. As such, containers for *Zookeeper* will need to be defined as well. In this step, container definitions for three *Zookeeper* containers (*zkServer1*, *zkServer2*, and *zkServer3*) will be added to *docker-compose.yml*. Note that best practice recommends an odd number of *Zookeeper* broker nodes in production scenarios. Add the following container definitions to the *docker-compose.yml* file.

```
  zkServer1.btacoin.com:

    container_name: zkServer1.btacoin.com

    image: hyperledger/fabric-zookeeper

    environment:

      - ZOO_MY_ID=1

      - ZOO_SERVERS=server.1=zkServer1.btacoin.com:2888:3888
server.2=zkServer2.btacoin.com:2888:3888
server.3=zookeeper3.btacoin.com:2888:3888

    ports:

      - 12181:2181

      - 12888:2888

      - 13888:3888

    networks:

      - btacoin


  zkServer2.btacoin.com:

    container_name: zkServer2.btacoin.com

    image: hyperledger/fabric-zookeeper

    environment:

      - ZOO_MY_ID=2

      - ZOO_SERVERS=server.1=zkServer1.btacoin.com:2888:3888
server.2=zkServer2.btacoin.com:2888:3888
server.3=zkServer3.btacoin.com:2888:3888

    ports:

      - 2181:2181

      - 2888:2888

      - 3888:3888

    networks:
```

```
      - btacoin


  zkServer3.btacoin.com:
    container_name: zkServer3.btacoin.com
    image: hyperledger/fabric-zookeeper
    environment:
      - ZOO_MY_ID=3
      - ZOO_SERVERS=server.1=zkServer1.btacoin.com:2888:3888
server.2=zkServer2.btacoin.com:2888:3888
server.3=zkServer3.btacoin.com:2888:3888
    ports:
      - 22181:2181
      - 22888:2888
      - 23888:3888
    networks:
      - btacoin
```

```
178
179   zkServer1.btacoin.com:
180     container_name: zkServer1.btacoin.com
181     image: hyperledger/fabric-zookeeper
182     environment:
183       - ZOO_MY_ID=1
184       - ZOO_SERVERS=server.1=zkServer1.btacoin.com:2888:3888 server.2=zkServer2.btacoin.com:2888:3888 server.3=zookeeper3.btac
185     ports:
186       - 12181:2181
187       - 12888:2888
188       - 13888:3888
189     networks:
190       - btacoin
191
```

Figure 134 - Snippet 6.2

```
191
192   zkServer2.btacoin.com:
193     container_name: zkServer2.btacoin.com
194     image: hyperledger/fabric-zookeeper
195     environment:
196       - ZOO_MY_ID=2
197       - ZOO_SERVERS=server.1=zkServer1.btacoin.com:2888:3888 server.2=zkServer2.btacoin.com:2888:3888 server.3=zkServer3.btaco
198     ports:
199       - 2181:2181
200       - 2888:2888
201       - 3888:3888
202     networks:
203       - btacoin
```

Figure 135 - Snippet 6.2

```
204
205   zkServer3.btacoin.com:
206     container_name: zkServer3.btacoin.com
207     image: hyperledger/fabric-zookeeper
208     environment:
209       - ZOO_MY_ID=3
210       - ZOO_SERVERS=server.1=zkServer1.btacoin.com:2888:3888 server.2=zkServer2.btacoin.com:2888:3888 server.3=zkServer3.btac
211     ports:
212       - 22181:2181
213       - 22888:2888
214       - 23888:3888
215     networks:
216       - btacoin
217
```

Figure 136 - Snippet 6.2

To complete the configuration change from Solo to Kafka, edits must be made the *configtx.yaml* file as well. Begin by updating the *OrdererType* parameter in the `Orderer: &DevModeOrdering` section.

Change the *OrdererType* parameter from:

```
OrdererType: solo
```

to:

```
OrdererType: kafka
```

```
25
26  Orderer: &DevModeOrdering
27      OrdererType: kafka
28      Addresses:
29        - Devorderer.btacoin.com:7050
30      BatchTimeout: 1s
31      BatchSize:
32        MaxMessageCount: 1
33
```

Figure 137 - Snippet 6.3

Next, address endpoints must be specified for Kafka nodes. Append the following to the end of the `Orderer: &DevModeOrdering` section.

```
    Kafka:
      Brokers:
        - kafkaServer1.btacoin.com:9092
        - kafkaServer2.btacoin.com:9092
```

```
25
26  Orderer: &DevModeOrdering
27      OrdererType: kafka
28      Addresses:
29        - Devorderer.btacoin.com:7050
30      BatchTimeout: 1s
31      BatchSize:
32        MaxMessageCount: 1
33      Kafka:
34        Brokers:
35          - kafkaServer1.btacoin.com:9092
36          - kafkaServer2.btacoin.com:9092
37
```

Figure 138 - Snippet 6.4

In the *Profiles* section under the *DefaultBlockOrderingService* definition, append the *courseParticipants* organization under the *Consortiums* / *NetworkConsortium* / *Organizations* branch.

```
            - *courseParticipants
```

```
39
40  Profiles:
41    DefaultBlockOrderingService:
42      Orderer:
43        <<: *DevModeOrdering
44        Organizations:
45          - *btacoinOrderers
46      Consortiums:
47        NetworkConsortium:
48          Organizations:
49            - *BTA
50            - *courseParticipants
51
```

Figure 139 - Snippet 6.5

Generating new configuration artifacts

At this point new configuration artifacts must be re-generated to reflect the changes that have been made. The existing configuration artifacts must first be deleted. Begin by opening a terminal window and running the following commands.

```
cd ~/Desktop/fabric/network/config
```

```
rm btamembersonly.block btamembersonly.tx genesis.block
```

```
student@hlfmc: ~/Desktop/fabric/network/config
student@hlfmc:~$ cd ~/Desktop/fabric/network/config
student@hlfmc:~/Desktop/fabric/network/config$ rm btamembersonly.block btamember
sonly.tx genesis.block
rm: remove write-protected regular file 'btamembersonly.block'? y
student@hlfmc:~/Desktop/fabric/network/config$
```

Figure 140 - Snippet 6.6

Next, the *configtxgen* tool will be used to create the genesis block and *btamembersonly* configuration artifacts. Run the following commands.

```
configtxgen -profile DefaultBlockOrderingService -outputBlock
./genesis.block
```

```
configtxgen -profile btaMembersOnly -outputCreateChannelTx
./btamembersonly.tx -channelID btamembersonly
```

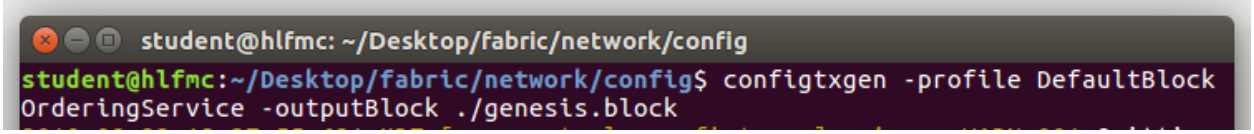

Figure 141 - Snippet 6.7

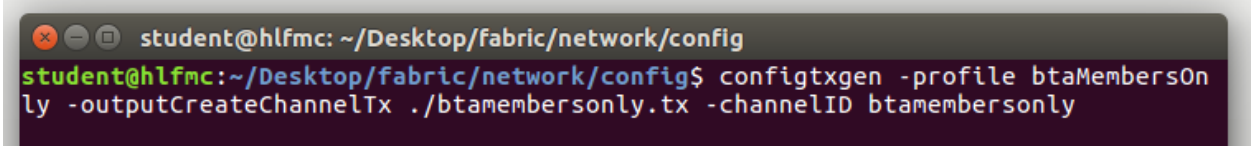

Figure 142 - Snippet 6.7

Restarting the network

To complete the transition from Solo to Kafka, all nodes on the network will be restarted. Begin by issuing the following command.

```
docker rm -f Andy.BTA.btacoin.com Devorderer.btacoin.com cli
courseParticipantsCA.btacoin.com btaCA.btacoin.com
```

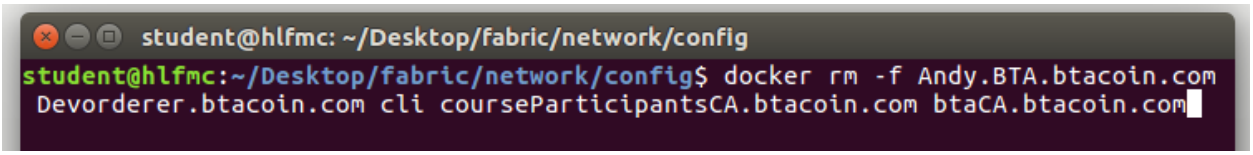

Figure 143 - Snippet 6.8

Use the commands below to bring up all containers on the network.

```
cd ..

docker-compose -f docker-compose.yml up -d Andy.BTA.btacoin.com
Devorderer.btacoin.com cli courseParticipantsCA.btacoin.com
btaCA.btacoin.com zkServer1.btacoin.com zkServer2.btacoin.com
zkServer3.btacoin.com kafkaServer1.btacoin.com
kafkaServer2.btacoin.com
```

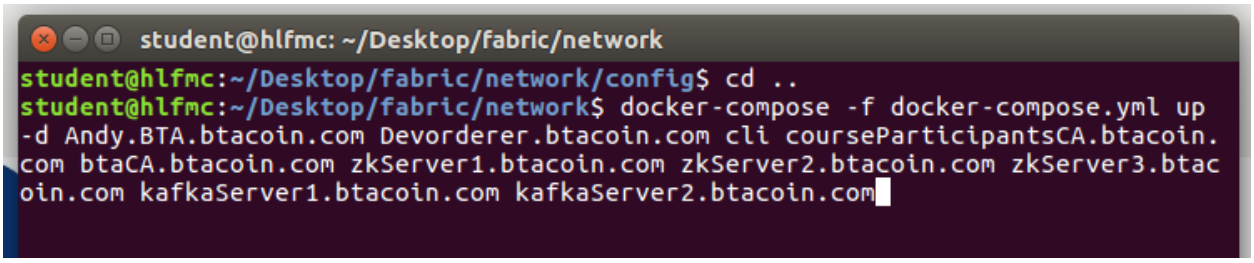

Figure 144 - Snippet 6.9

Updating the World State Database Configuration

As currently configured, the Fabric network is using *levelDB* as the database implementation of the World State database. Most production configurations opt to go with *CouchDB* instead. One difference between the two options is that *CouchDB* will run in its own container (one per peer) while *levelDB* runs inside the peer instances themselves. This can be verified by using the *docker ps* command. Note the lack of any containers dedicated to *levelDB* or the World State Database.

```
student@hlfmc:~/Desktop/fabric/network$ docker ps
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS              PORTS                                                                                  NAMES
452ff6566ee9        hyperledger/fabric-tools   "/bin/bash"              About a minute ago  Up About a minute                                                                                  cli
85fc8001c714        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                                     Andy.BTA.btacoin.com
2bfc73602b68        hyperledger/fabric-orderer "orderer"                About a minute ago  Up About a minute   0.0.0.0:7050->7050/tcp                                                             Devorderer.btacoin.co
m
85b517a7db32        hyperledger/fabric-ca      "/bin/sh -c 'fabric-…"   About a minute ago  Up About a minute   0.0.0.0:7054->7054/tcp                                                             btaCA.btacoin.com
a0d7e06069c1        hyperledger/fabric-kafka   "/docker-entrypoint.…"   4 minutes ago       Up 4 minutes        0.0.0.0:9092-9093->9092-9093/tcp                                                   kafkaServer2.btacoin.
com
71b63fd32b26        hyperledger/fabric-kafka   "/docker-entrypoint.…"   4 minutes ago       Up 4 minutes        0.0.0.0:10092->9092/tcp, 0.0.0.0:10093->9093/tcp                                   kafkaServer1.btacoin.
com
9b131e0eb7a7        hyperledger/fabric-zookeeper "/docker-entrypoint.…"  4 minutes ago       Up 4 minutes        0.0.0.0:12181->2181/tcp, 0.0.0.0:12888->2888/tcp, 0.0.0.0:13888->3888/tcp          zkServer1.btacoin.com
db7c1dc059Se        hyperledger/fabric-zookeeper "/docker-entrypoint.…"  4 minutes ago       Up 4 minutes        0.0.0.0:2181->2181/tcp, 0.0.0.0:2888->2888/tcp, 0.0.0.0:3888->3888/tcp             zkServer2.btacoin.com
2b4ecd0aff3b        hyperledger/fabric-zookeeper "/docker-entrypoint.…"  4 minutes ago       Up 4 minutes        0.0.0.0:22181->2181/tcp, 0.0.0.0:22888->2888/tcp, 0.0.0.0:23888->3888/tcp          zkServer3.btacoin.com
5ffda81c3e1c        hyperledger/fabric-ca      "sh -c 'fabric-ca-se…"   8 days ago          Up 4 minutes        0.0.0.0:8054->7054/tcp                                                             courseParticipantsCA
student@hlfmc:~/Desktop/fabric/network$
```

Figure 145 - The output of the docker ps command

As mentioned above, each peer will require a CouchDB container as they will no longer be running the World State database internally. Add the following container definition to the *docker-compose.yml* file to create the CouchDB container for the Andy peer. Note the use of the *hyperledger/fabric-couchdb* image type for CouchDB containers.

```
  WorldStateCouchAndy:
    container_name: WorldStateCouchAndy
    image: hyperledger/fabric-couchdb
    environment:
      - COUCHDB_USER=Andy
      - COUCHDB_PASSWORD=SimplePassword
    ports:
      - 5984:5984
    networks:
      - btacoin
```

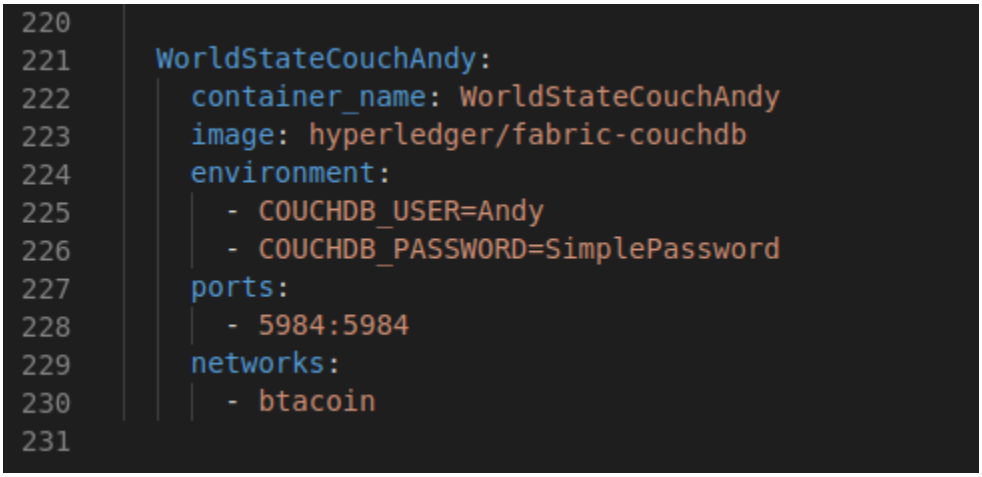

Figure 146 - Snippet 6.10

Now that the World State database will no longer be hosted inside network peer containers, an external dependency has been created. In order for the network to operate properly, it is important to ensure World State database containers are online and available before bringing up peer containers. This can be handled using the *depends_on* section of a container definition. In the *depends_on* section of the container definition for *Andy.BTA.btacoin.com* add the following.

```
- WorldStateCouchAndy
```

```
42
43  Andy.BTA.btacoin.com:
44    container_name: Andy.BTA.btacoin.com
45    image: hyperledger/fabric-peer
46    environment:
47      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
48      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
49      - FABRIC_LOGGING_SPEC=debug
50      - CORE_PEER_ID=Andy.BTA.btacoin.com
51      - CORE_PEER_ADDRESS=Andy.BTA.btacoin.com:7051
52      - CORE_PEER_LOCALMSPID=BTAMSP
53      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
54    command: peer node start
55    volumes:
56      - /var/run/:/host/var/run/
57      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/msp:/etc/hyperledger/msp/peer
58      - ./crypto-config/peerOrganizations/BTA.btacoin.com/users:/etc/hyperledger/msp/users
59      - ./config:/etc/hyperledger/configtx
60      - ./../cc:/etc/hyperledger/chaincode
61    ports:
62      - 7051:7051
63      - 7053:7053
64    networks:
65      - btacoin
66    depends_on:
67      - Devorderer.btacoin.com
68      - WorldStateCouchAndy
69
```

Figure 147 - Snippet 6.11

Several new environment variables need to be added to the *Andy.BTA.btacoin.com* container definition. Append the following to the *environment* section of the Andy container definition.

```
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=WorldStateCouchAndy:5984
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=Andy
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=SimplePassword
```

```
42
43  Andy.BTA.btacoin.com:
44    container_name: Andy.BTA.btacoin.com
45    image: hyperledger/fabric-peer
46    environment:
47      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
48      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
49      - FABRIC_LOGGING_SPEC=debug
50      - CORE_PEER_ID=Andy.BTA.btacoin.com
51      - CORE_PEER_ADDRESS=Andy.BTA.btacoin.com:7051
52      - CORE_PEER_LOCALMSPID=BTAMSP
53      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
54      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
55      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=WorldStateCouchAndy:5984
56      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=Andy
57      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=SimplePassword
58    command: peer node start
59    volumes:
60      - /var/run/:/host/var/run/
61      - ./crypto-config/peerOrganizations/BTA.btacoin.com/peers/Andy.BTA.btacoin.com/ms
62      - ./crypto-config/peerOrganizations/BTA.btacoin.com/users:/etc/hyperledger/msp/us
```

Figure 148 - Snippet 6.12

Applying the changes

At this point, the Andy peer container must be restarted for configuration changes to take effect. Open a terminal windows and run the following commands.

```
cd ~/Desktop/fabric/network

docker-compose -f docker-compose.yml restart Andy.BTA.btacoin.com

docker-compose -f docker-compose.yml up WorldStateCouchAndy
```

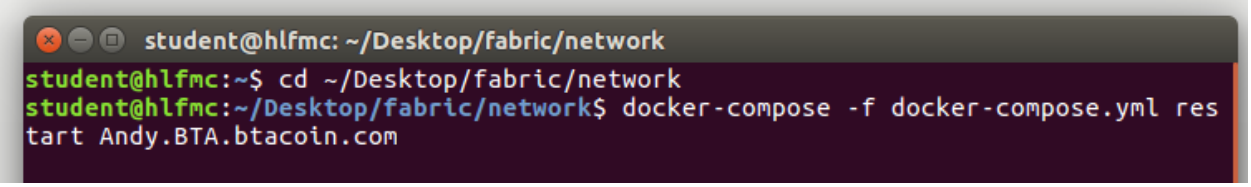

Figure 149 - Snippet 6.13

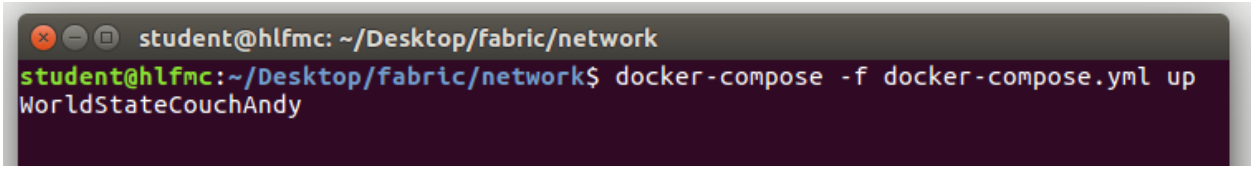

Figure 150 - Snippet 6.14

Run the *docker ps* command to verify the CouchDB container is up and running.

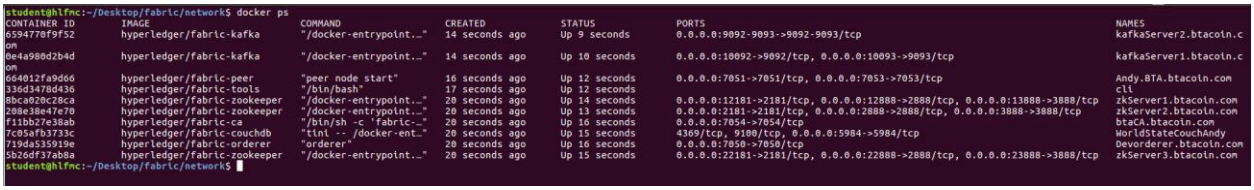

Figure 151 - Snippet 6.15

Adding a second *couchDB* instance

Now add a second couchDB instance for Ken, our course participant. Add the following container definitions to the *docker-compose-yml* file for *WorldStateCouchKen* and ensure the existing definition for *Ken.courseParticipants.btacoin.com* resembles the one below.

```
  Ken.courseParticipants.btacoin.com:
    container_name: Ken.courseParticipants.btacoin.com
    image: hyperledger/fabric-peer
    environment:
      - CORE_PEER_ID=Ken.courseParticipants.btacoin.com
      - CORE_PEER_ADDRESS=Ken.courseParticipants.btacoin.com:8051
      - FABRIC_LOGGING_SPEC=debug
      - CORE_CHAINCODE_LOGGING_LEVEL=debug
      - CORE_PEER_LOCALMSPID=courseParticipantsMSP
      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      -
CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      -
CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=WorldStateCouchKen:5984
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=Ken
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=SimplePassword
    working_dir: /etc/hyperledger/configtx
    command: peer node start
    ports:
      - 8051:7051
      - 8053:7053
    volumes:
      - /var/run/:/host/var/run/
      - ./crypto-
config/peerOrganizations/courseParticipants.btacoin.com/peers/Ken.cour
seParticipants.btacoin.com/msp:/etc/hyperledger/msp/peer
```

```
      - ./crypto-
config/peerOrganizations/courseParticipants.btacoin.com/users:/etc/hyp
erledger/msp/users
      - ./config:/etc/hyperledger/configtx
    depends_on:
      - Devorderer.btacoin.com
      - WorldStateCouchKen
    networks:
      - btacoin



  WorldStateCouchKen:
    container_name: WorldStateCouchKen
    image: hyperledger/fabric-couchdb
    environment:
      - COUCHDB_USER=Ken
      - COUCHDB_PASSWORD=SimplePassword
    ports:
      - 6984:5984
    networks:
      - btacoin
```

```
96
97  Ken.courseParticipants.btacoin.com:
98    container_name: Ken.courseParticipants.btacoin.com
99    image: hyperledger/fabric-peer
100   environment:
101     - CORE_PEER_ID=Ken.courseParticipants.btacoin.com
102     - CORE_PEER_ADDRESS=Ken.courseParticipants.btacoin.com:8051
103     - FABRIC_LOGGING_SPEC=debug
104     - CORE_CHAINCODE_LOGGING_LEVEL=debug
105     - CORE_PEER_LOCALMSPID=courseParticipantsMSP
106     - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/peer/
107     - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
108     - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_btacoin
109     - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
110     - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=WorldStateCouchKen:5984
111     - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=Ken
112     - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=SimplePassword
113   working_dir: /etc/hyperledger/configtx
114   command: peer node start
115   ports:
116     - 8051:7051
117     - 8053:7053
118   volumes:
119     - /var/run/:/host/var/run/
120     - ./crypto-config/peerOrganizations/courseParticipants.btacoin.com/peers/Ken.courseParticipants.btacoin.com/msp:/etc/hy
121     - ./crypto-config/peerOrganizations/courseParticipants.btacoin.com/users:/etc/hyperledger/msp/users
122     - ./config:/etc/hyperledger/configtx
123   depends_on:
124     - Devorderer.btacoin.com
125     - WorldStateCouchKen
126   networks:
127     - btacoin
```

Figure 152 - Snippet 6.16

```
265
266  WorldStateCouchKen:
267    container_name: WorldStateCouchKen
268    image: hyperledger/fabric-couchdb
269    environment:
270      - COUCHDB_USER=Ken
271      - COUCHDB_PASSWORD=SimplePassword
272    ports:
273      - 6984:5984
274    networks:
275      - btacoin
276
```

Figure 153 - Snippet 6.16

Lab 7 – Adding Channels to a Network

At this point, the network consists of two Organizations with multiple peers. In this lab, a channel will be created for use between the *BTA* and *courseParticipants* Organizations. The original channel used just for BTA members (*btaMembersOnly*) will remain unaffected.

Defining the new channel

Begin by opening the *configtx.yaml* file for editing. Create a new profile using the code below. Insert the code at the bottom of the *configtx.yaml* file, in the *Profiles* section beneath the definition for *btaMembersOnly.*

```
  CoinExchangeChannel:
    Consortium: NetworkConsortium
    Application:
      <<: *ApplicationDefaults
      Organizations:
        - *BTA
        - *courseParticipants
```

```
58
59    CoinExchangeChannel:
60      Consortium: NetworkConsortium
61      Application:
62        <<: *ApplicationDefaults
63        Organizations:
64          - *BTA
65          - *courseParticipants
```

Figure 154 - Snippet 7.1

Using *configtxgen* to apply the changes

Now that the appropriate edits have been made the *configtxgen* tool can be used to create the channel artifacts. Open a terminal window and issue the commands below.

```
cd ~/Desktop/fabric/network
```

```
configtxgen -profile CoinExchangeChannel -outputCreateChannelTx
./config/coinexchangechannel.tx -channelID coinexchangechannel
```

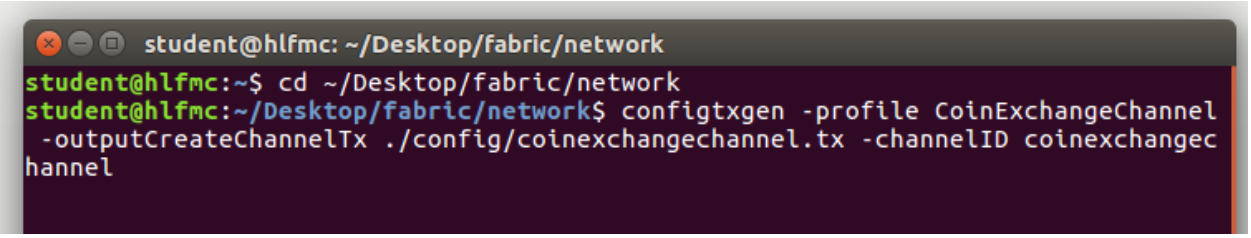

Figure 155 - Snippet 7.2

Now the Andy peer container will be used to initiate the create channel transaction. Connection to the Andy peer will be made using administrative credentials to avoid any permissioning errors. Use the command below to connect to the Andy peer.

```
docker exec -e
"CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@BTA.btacoin.
com/msp" -e "CORE_PEER_LOCALMSPID=BTAMSP" -it Andy.BTA.btacoin.com
bash
```

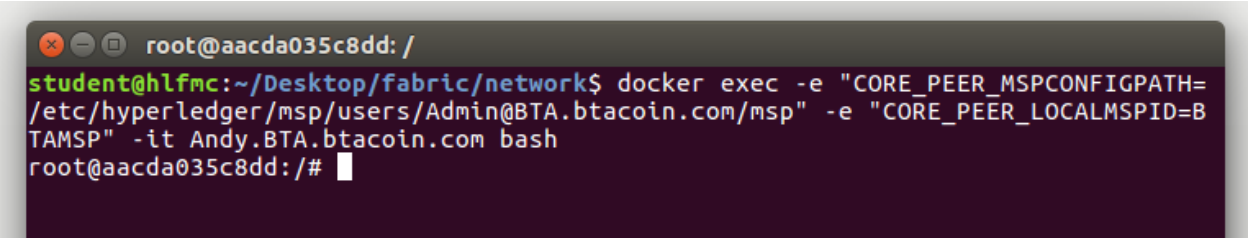

Figure 156 - Snippet 7.3

Creating and joining the new channel

Now that a container connection has been established the *btaMembersOnly* channel can be created from the channel artifacts. Use the following commands to create the *btaMembersOnly* channel.

```
cd /etc/hyperledger/configtx
```

```
peer channel create -o Devorderer.btacoin.com:7050 -f
/etc/hyperledger/configtx/btamembersonly.tx -c btamembersonly
```

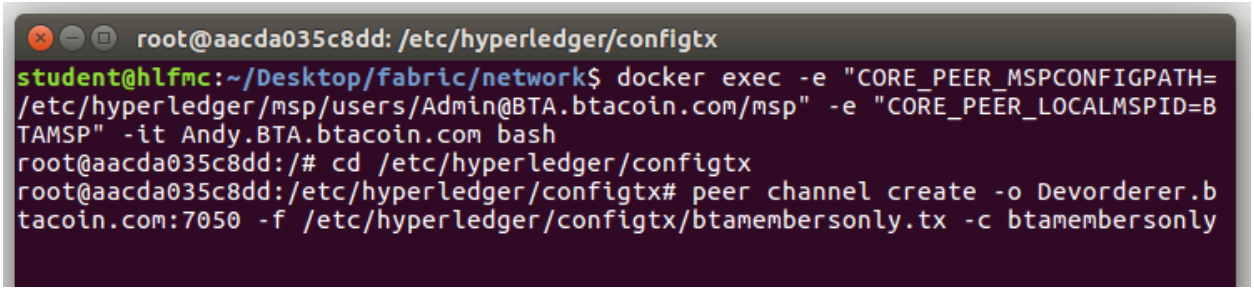

Figure 157- Snippet 7.4

Now join the peer to the *btaMembersOnly* channel.

```
peer channel join -b btamembersonly.block
```

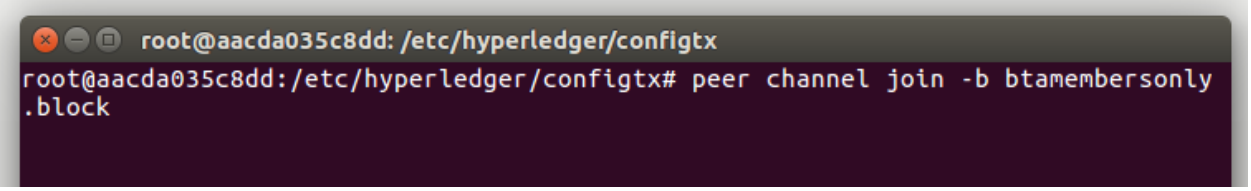

Figure 158 - Snippet 7.5

The process will be repeated for the new *CoinExchangeChannel*. First, the new channel will be created using the command below.

```
peer channel create -o Devorderer.btacoin.com:7050 -f
/etc/hyperledger/configtx/coinexchangechannel.tx -c
coinexchangechannel
```

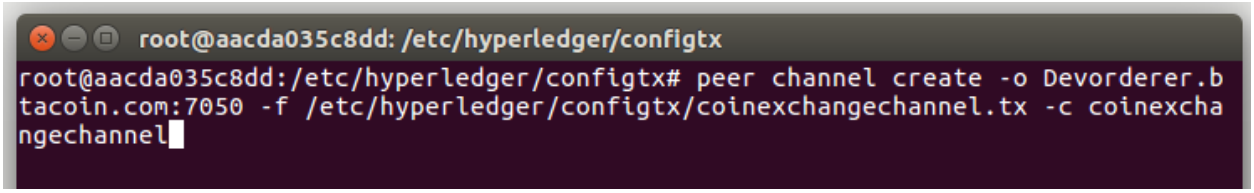

Figure 159 - Snippet 7.6

Finally, the peer will be joined to the new channel using the code below.

```
peer channel join -b coinexchangechannel.block
```

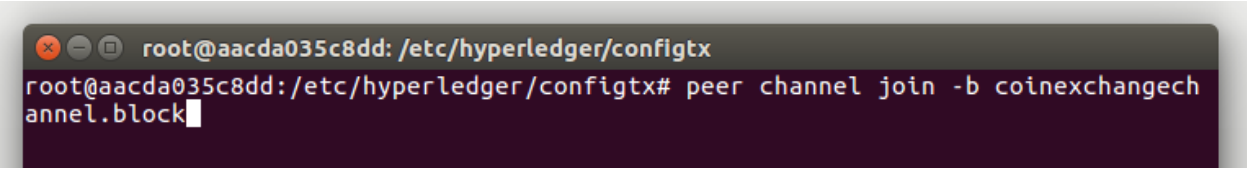

Figure 160 - Snippet 7.7

Confirm channel membership using the command below.

```
peer channel list
```

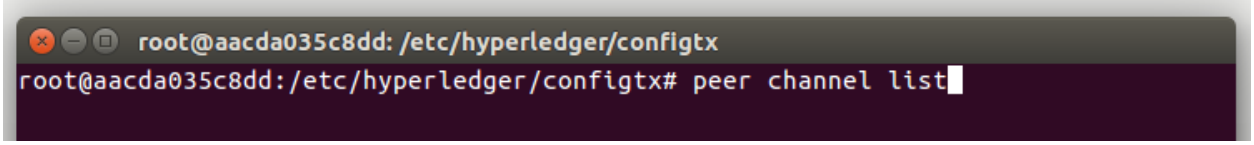

Figure 161 - Snippet 7.8

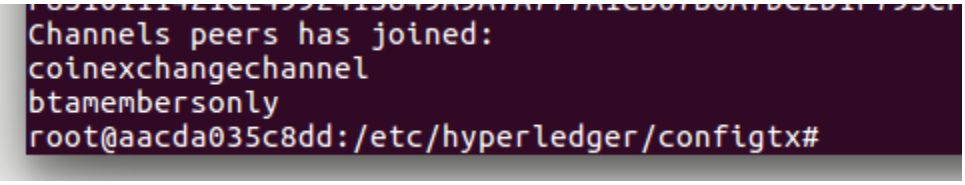

Figure 162 - The output of the peer channel list command

Exit the container using the *exit* command.

```
exit
```

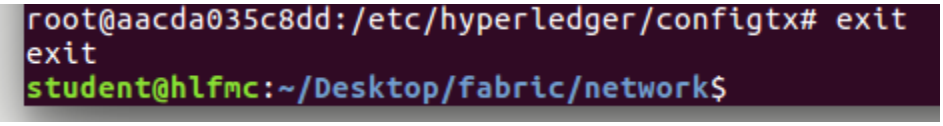

Figure 163 - Snippet 7.9

Joining the second peer to the new channel

The second peer, *Ken.courseParticipants.btacoin.com*, needs to be joined to the newly created channel *CoinExchangeChannel* but not to the first channel *btaMembersOnly*. Use the code below to connect to the Ken peer with administrative access.

```
docker exec -e
"CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@courseParticipants.btacoin.com/msp" -e
"CORE_PEER_LOCALMSPID=courseParticipantsMSP" -e
"CORE_PEER_ADDRESS=Ken.courseParticipants.btacoin.com:7051" -it
Ken.courseParticipants.btacoin.com bash
```

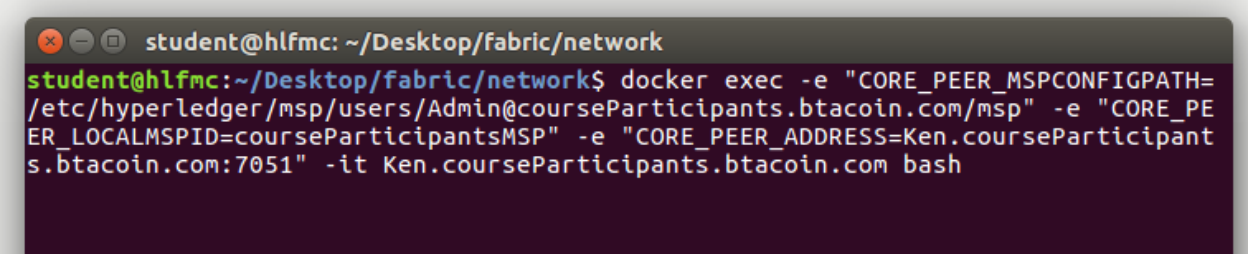

Figure 164 - Snippet 7.10

Now join the *Ken* peer to the *CoinExchangeChannel* using the commands below.

```
cd /etc/hyperledger/configtx
```

```
peer channel join -b coinexchangechannel.block
```

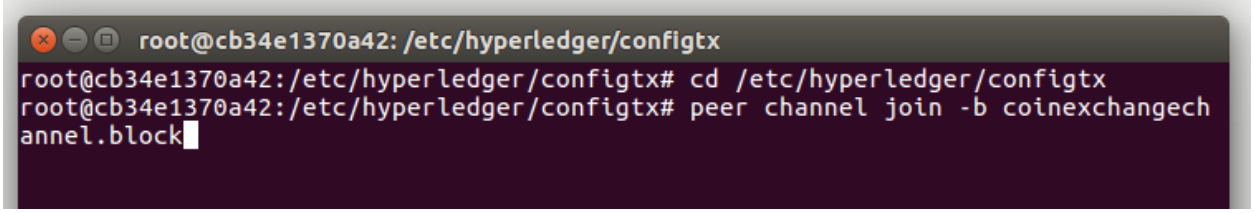

Figure 165 - Snippet 7.11

Verify channel membership using the *peer channel list* command.

```
peer channel list
```

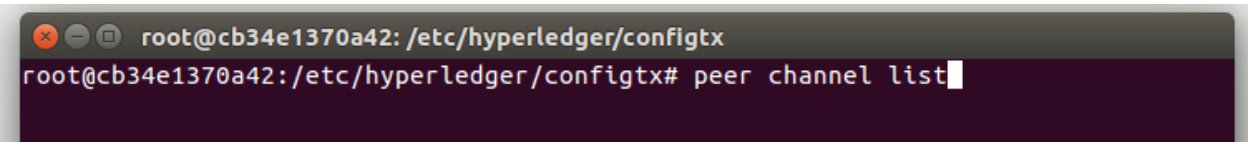

Figure 166 - Snippet 7.12

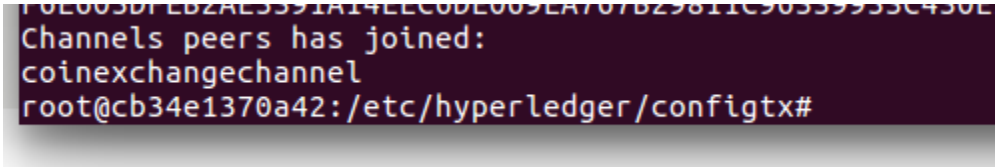

Figure 167 - The output of the peer channel list command

Exit the container using the *exit* command.

```
exit
```

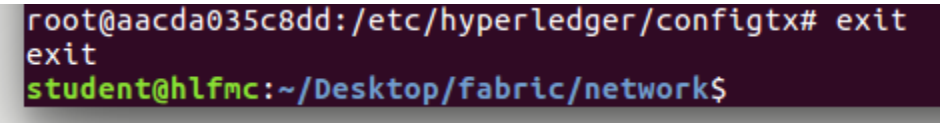

Figure 168 - Snippet 7.13

Lab 8 – Managing Identities on the Network

In this lab, you will perform further configuration on the Certificate Authority component as well as enroll several new identities onto the network.

Updating the network configuration

Begin by opening a terminal window and navigating to the *Desktop/fabric/network* folder location.

```
cd ~/Desktop/fabric/network/
```

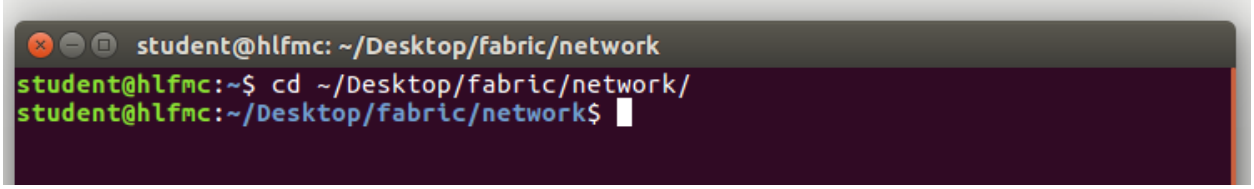

Figure 169 - Snippet 8.1

Remove the current *courseParticipantsCA.btacoin.com* container.

```
docker rm -f courseParticipantsCA.btacoin.com
```

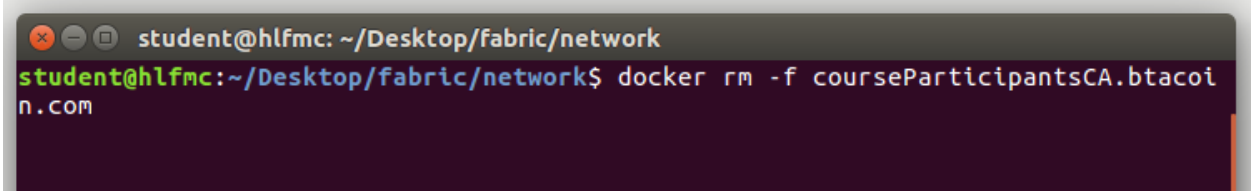

Figure 170 - Snippet 8.2

Open the *docker-compose.yml* file for editing. In the container definition for *courseParticipantsCA* replace the line:

```
    command: sh -c 'fabric-ca-server start -b courseParticipantsCA:SimplePassword'
```

with:

```
    command: sh -c 'sleep 100000'
```

```
courseParticipantsCA.btacoin.com:
  container_name: courseParticipantsCA
  image: hyperledger/fabric-ca
  environment:
    - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
    - FABRIC_CA_SERVER_CA_NAME= courseParticipantsCA.btacoin.com
    - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-caserver-config/ca.courseParticipants.btacoin.com-cert.pem
    - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-caserver-config/5fcee588499b8ce7ed5242f535da1d927e4b4a7499ef64
  ports:
    - "8054:7054"
  command: sh -c 'fabric-ca-server start -b courseParticipantsCA:SimplePassword'
  volumes:
    - ./crypto-config/peerOrganizations/  courseParticipants.btacoin.com/ca/:/etc/hyperledger/fabric-caserver-config
  networks:
    - btacoin
```

Figure 171 - Snippet 8.3 NOTE – This screenshot denotes the content to be changed, NOT the final state!

Add this line to the *btaCA.btacoin.com* container definition as well.

```
    command: sh -c 'sleep 100000'
```

```
services:
  btaCA.btacoin.com:
    container_name: btaCA.btacoin.com
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_SERVER_CA_NAME=btaCA.btacoin.com
      - FABRIC_LOGGING_SPEC=debug
      - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-caserver-config/ca.BTA.btacoin.com-cert.pem
      - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-config/7d5e6ace900280ada6ba3a0b7000
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
    volumes:
      - ./crypto-config/peerOrganizations/BTA.btacoin.com/ca/:/etc/hyperledger/fabric-ca-server-config
    ports:
      - 7054:7054
    command: sh -c 'sleep 100000'
    networks:
      - btacoin
```

Figure 172 - Snippet 8.4

Apply the changes

Save the changes to the YAML file and run the following command from the terminal window to bring up the CA containers according to their new configuration.

```
docker-compose -f docker-compose.yml up -d
courseParticipantsCA.btacoin.com btaCA.btacoin.com
```

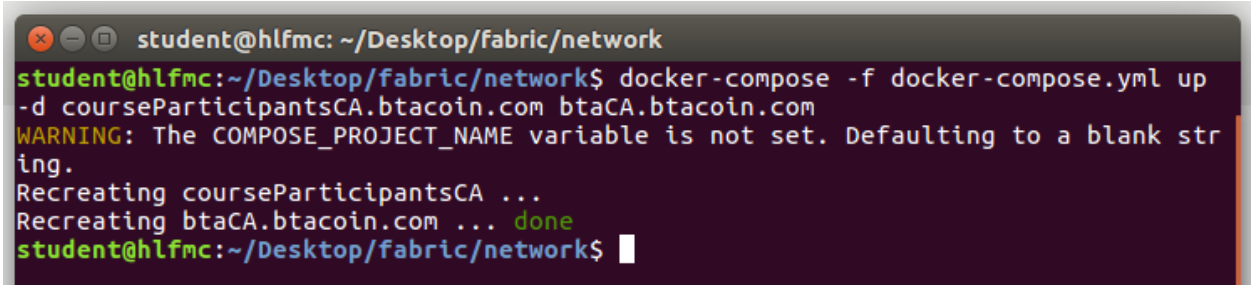

Figure 173 - Snippet 8.5

Connect to the running *btaCA* container using the command below.

```
docker exec -it btaCA.btacoin.com bash
```

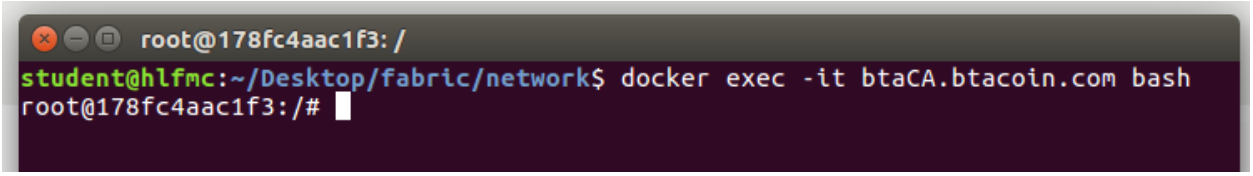

Figure 174 - Snippet 8.6

Reinitializing the CA

Use the commands below to navigate to the container's configuration directory and remove the initial setup information. This container will be re-initialized to provide a better understanding of how the CA is setup behind the scenes on startup.

```
cd /etc/hyperledger/fabric-ca-server
```

```
rm ca-cert.pem ca-key.pem fabric-ca-server-config.yaml
```

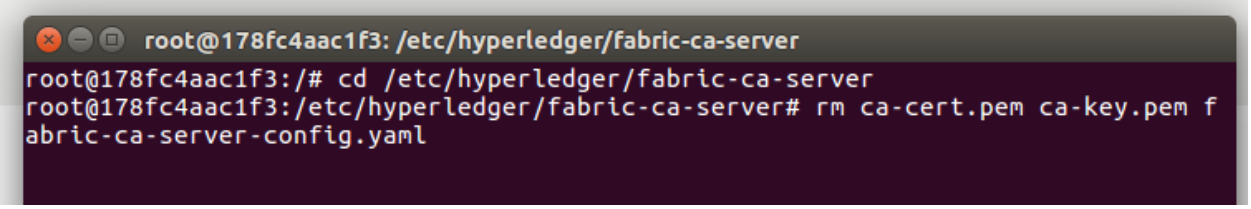

Figure 175 - Snippet 8.7

The CA server will need to be re-initialized before starting. Use the command below to perform initialization.

```
fabric-ca-server init -b btaCA:SimplePassword
```

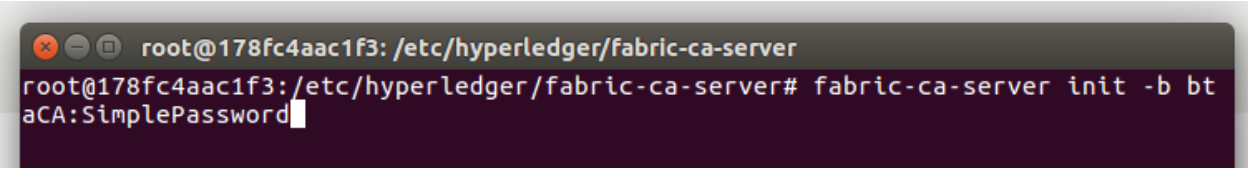

Figure 176 - Snippet 8.8

Now the CA server can be started. Use the command below to start the CA server.

```
fabric-ca-server start -b btaCA:SimplePassword -p 8000
```

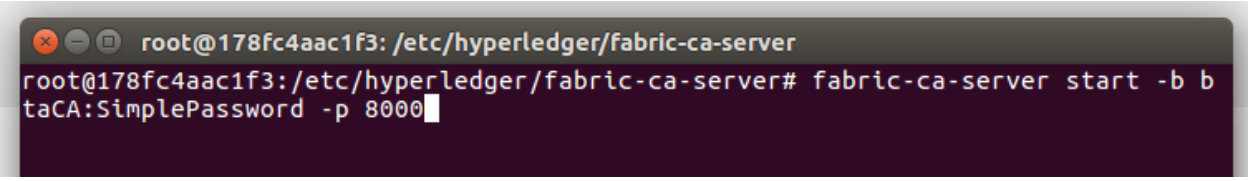

Figure 177 - Snippet 8.9

Please leave the current terminal window open and running! Begin by opening a new terminal window (**please leave the previous terminal window open and running**). Navigate into the *Desktop/fabric/network* folder.

```
cd ~/Desktop/fabric/network
```

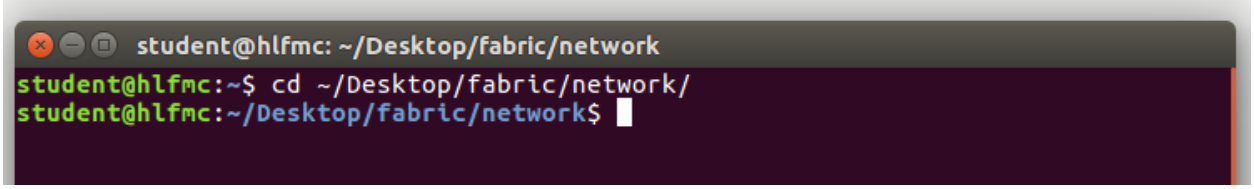

Figure 178 - Snippet 8.10

Connect to the *btaCA* container using the command below.

```
docker exec -it btaCA.btacoin.com bash
```

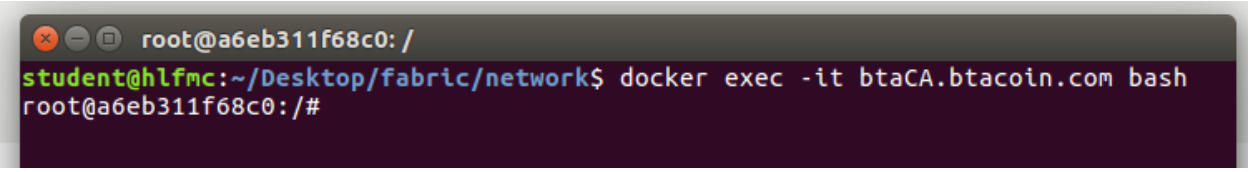

Figure 179 - Snippet 8.11

Registering new identities

The identity creation process for peers begins with registration. After registration, new identities can be enrolled. However, neither registration nor enrollment can occur before the CA has enrolled itself as an administrative identity. This one-time step is required to register and enroll subsequent identities. Use the commands below to enroll the CA.

```
fabric-ca-client enroll -u http://btaCA:SimplePassword@localhost:8000
-M FABRIC_CA_HOME/msp/
```

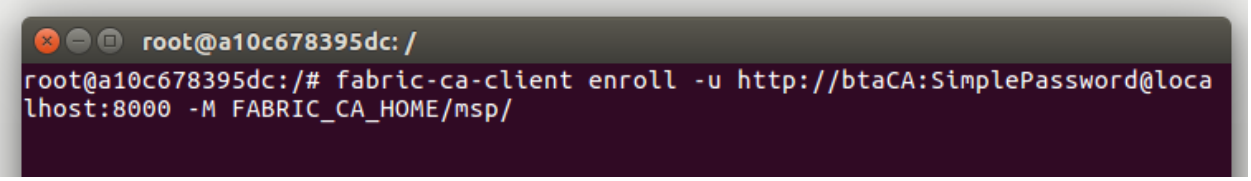

Figure 180 - Snippet 8.12

Continue by registering another administrator for the *CourseParticipants* organization using the command below.

```
fabric-ca-client register --id.name btaAdministrator --id.affiliation
org1 --id.type admin --id.attrs '"hf.Registrar.Roles=peer",
hf.GenCRL=true, admin=true:ecert' --id.secret BTARocks -u
http://localhost:8000
```

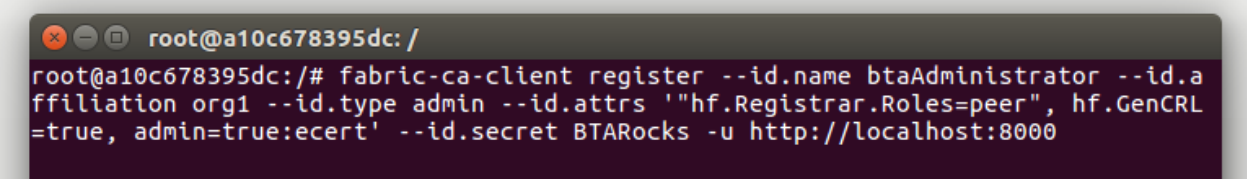

Figure 181 - Snippet 8.13

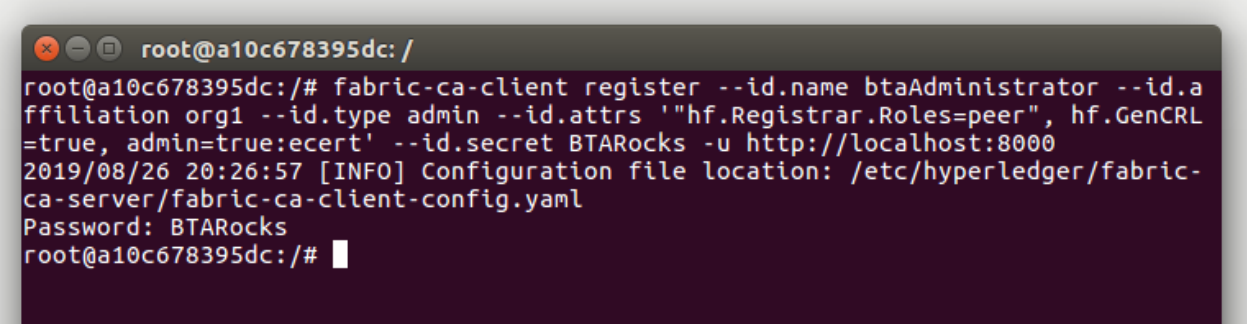

Figure 182 - The output of the fabric-ca-client register command

Enrolling new identities

After successfully registering, the new identity can be enrolled using the command below.

```
fabric-ca-client enroll -u
http://btaAdministrator:BTARocks@localhost:8000 -M
./msp/Org1Administrator
```

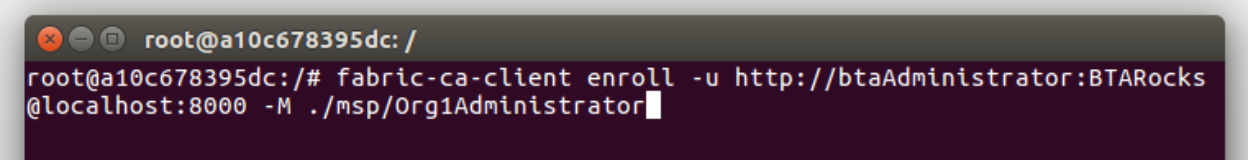

Figure 183 - Snippet 8.14

Now the registration / enrollment process will be repeated for a second new identity. Begin by registering a new identity, *JohnC*, using the command below.

```
fabric-ca-client register --id.name JohnC --id.affiliation org1 --
id.type peer --id.secret peerpw -u
http://btaAdministrator:BTARocks@localhost:8000
```

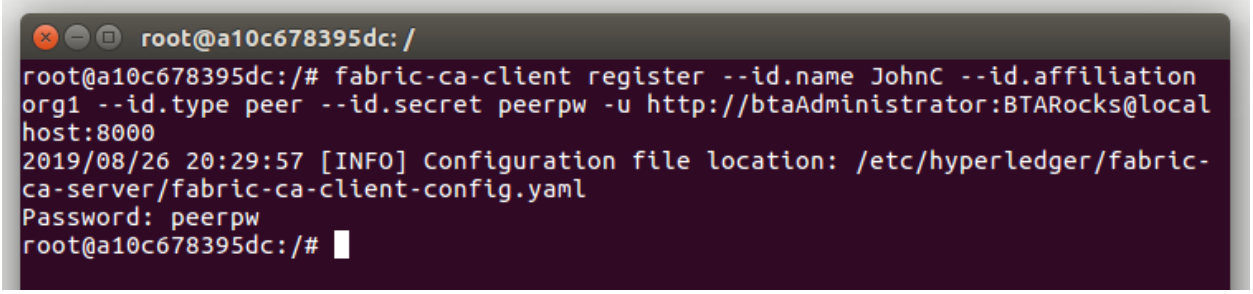

Figure 184 - Snippet 8.15

Complete the process by enrolling the new *JohnC* identity using the command below.

```
fabric-ca-client enroll -u http://JohnC:peerpw@localhost:8000 -M
./msp/JohnC
```

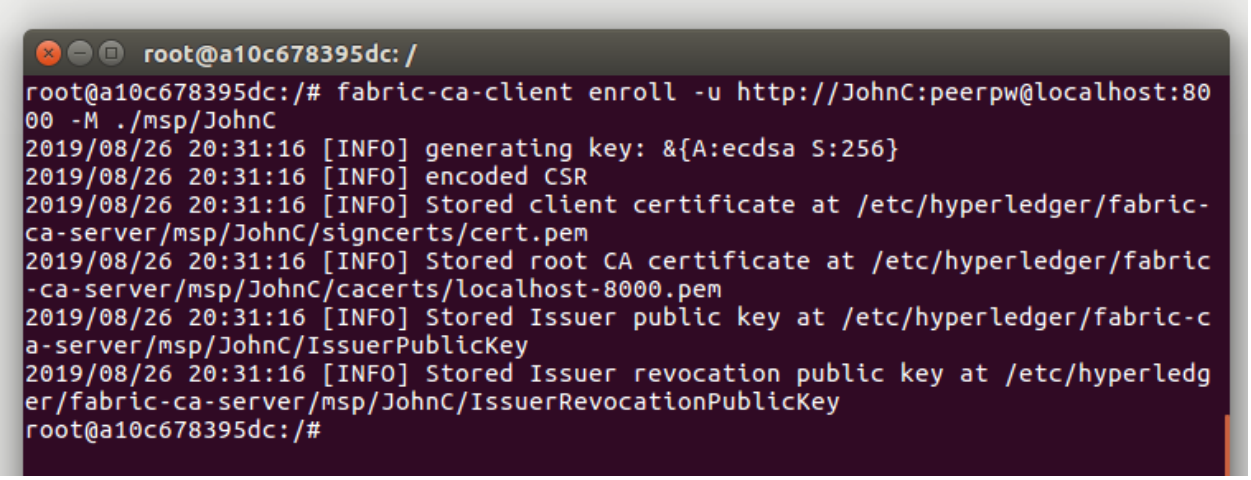

Figure 185 - Snippet 8.16

Open a new terminal window and navigate to the "Desktop/fabric/network" folder location.

```
cd ~/Desktop/fabric/network/
```

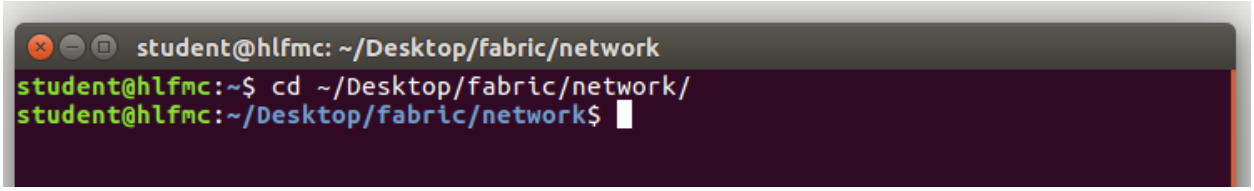

Figure 186 - Snippet 8.17

Connect to the *courseParticipantsCA* container using the command below.

```
docker exec -it courseParticipantsCA bash
```

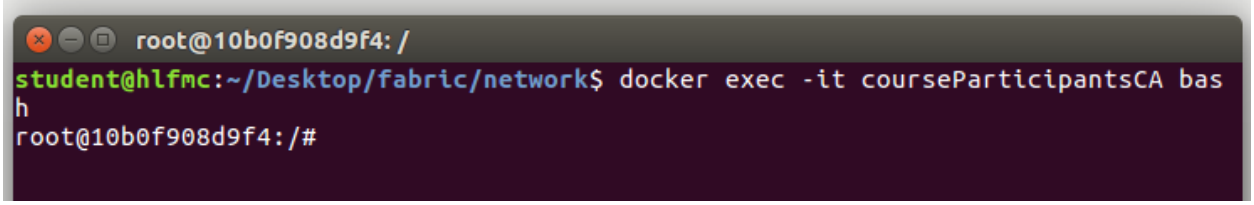

Figure 187 - Snippet 8.18

Perform initial setup tasks using the three commands below.

```
cd $FABRIC_CA_HOME
fabric-ca-server init -b AdminCourseParticipants:AdminsRock
fabric-ca-server start -b AdminCourseParticipants:AdminsRock -p 8000
```

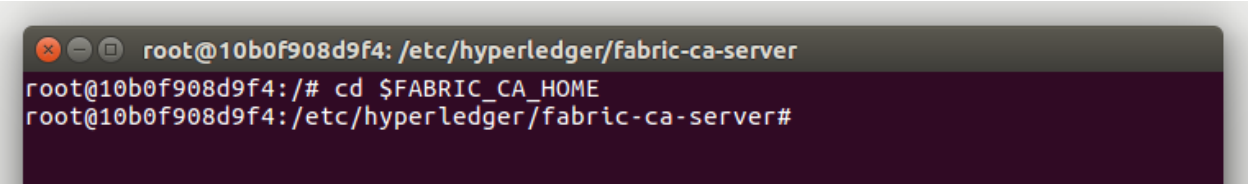

Figure 188 - Snippet 8.19

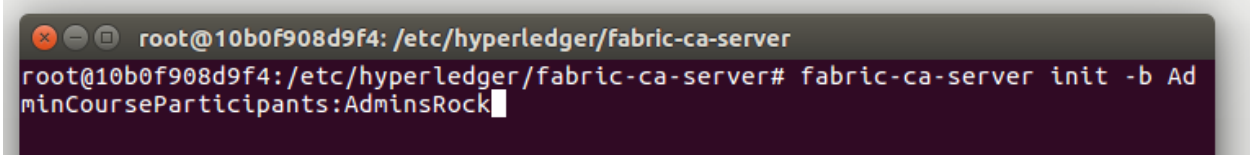

Figure 189 - Snippet 8.20

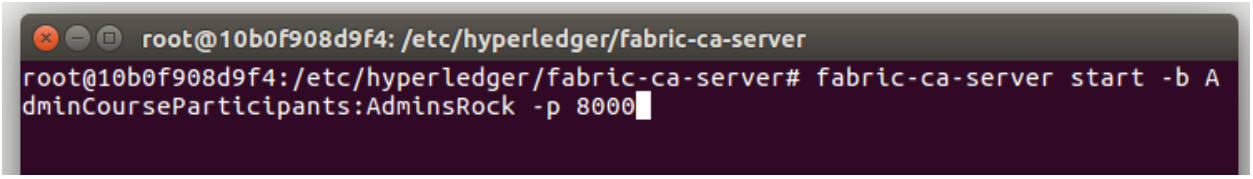

Figure 190 - Snippet 8.21

Open another new terminal window and navigate to the *Desktop/fabric/network* folder location.

```
cd ~/Desktop/fabric/network/
```

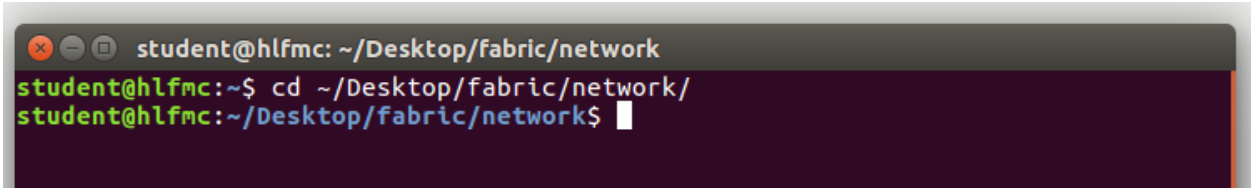

Figure 191 - Snippet 8.22

Connect to the *courseParticipantsCA* container using the command below.

```
docker exec -it courseParticipantsCA bash
```

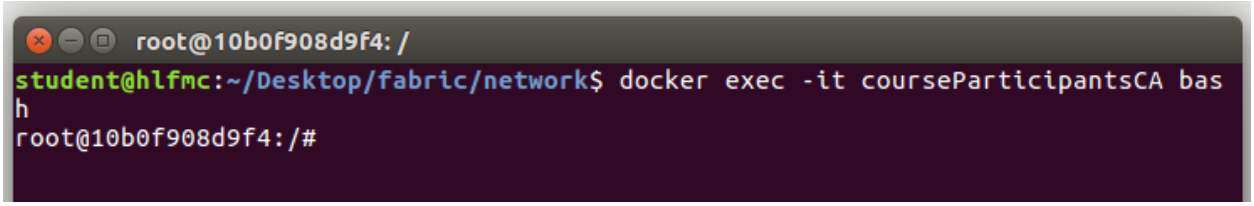

Figure 192 - Snippet 8.23

Enroll the bootstrap administrator using the command below.

```
fabric-ca-client enroll -u
http://AdminCourseParticipants:AdminsRock@localhost:8000
```

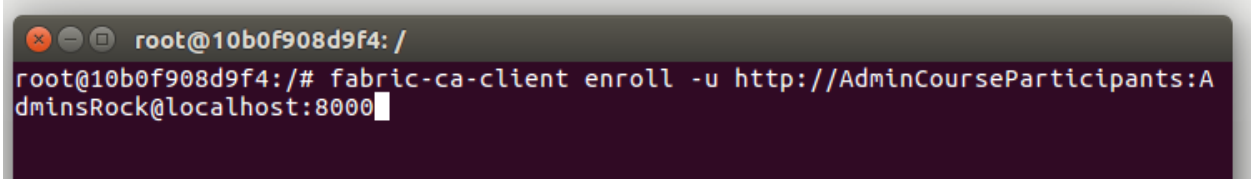

Figure 193 - Snippet 8.24

Now that the bootstrap administrator has been enrolled, an admin for the *courseParticipants* Organization can be created.

```
fabric-ca-client register --id.name ParticipantsAdministrator --
id.type admin --id.attrs '"hf.Registrar.Roles=peer,user",
hf.GenCRL=true,admin=true:ecert' --id.secret adminPassword -u
http://localhost:8000
```

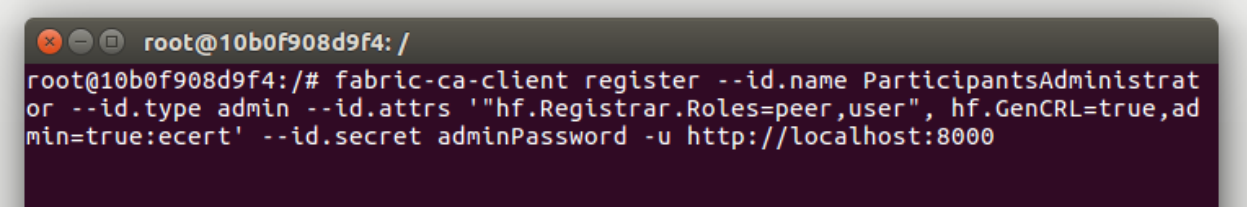

Figure 194 - Snippet 8.25

Now that the peer has been registered it can be enrolled using the command below.

```
fabric-ca-client enroll -u
http://ParticipantsAdministrator:adminPassword@localhost:8000 -M
./msp/ParticipantsAdministrator
```

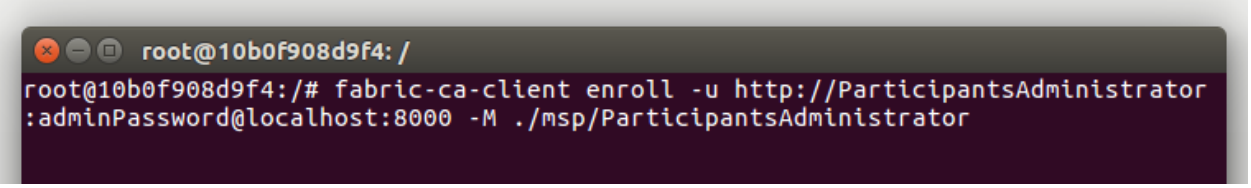

Figure 195 - Snippet 8.26

Now a new non-administrative identity will be created. Begin by registering the new identity *Rita* using the command below.

```
fabric-ca-client register --id.name Rita --id.type client --id.attrs
'admin=false:ecert, hf.Registrar.Roles=peer' --id.secret
SamplePassword -u http://localhost:8000
```

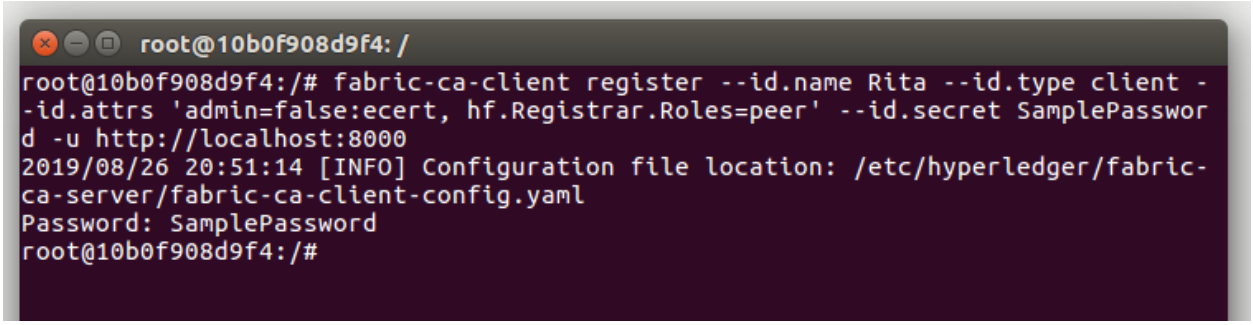

Figure 196 - Snippet 8.27

Now enroll the new *Rita* identity using the command below.

```
fabric-ca-client enroll -u http://Rita:SamplePassword@localhost:8000
```

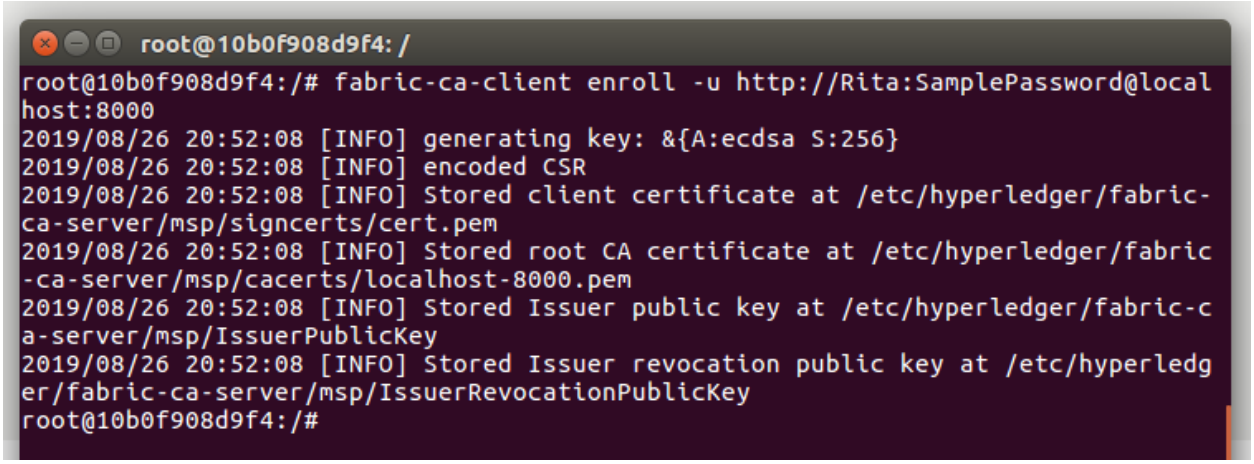

Figure 197 - Snippet 8.28

Updating identity information

Assume a mistake was made when collecting information for the new *Rita* identity. Assume that Rita should have had an *id.type* of *peer*, and Rita would like her password changed as well. Updates to both properties can be made with the command below.

```
fabric-ca-client identity modify Rita --type peer --secret
ritaIsTheBest -u http://localhost:8000
```

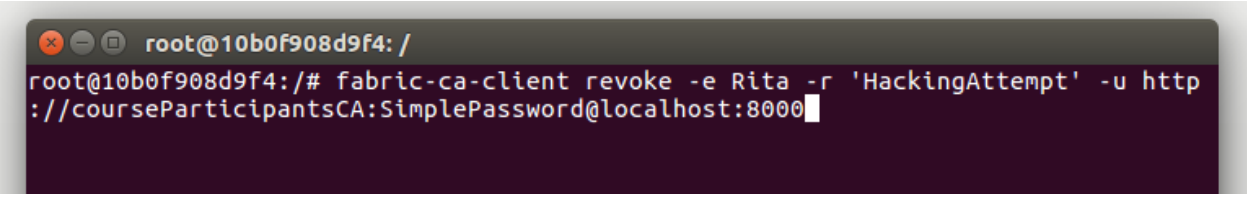

Figure 198 - Snippet 8.29

Revoking identities

For the purposes of this lab, assume the new Rita identity needs to be removed due to a report of malicious usage. The command below will revoke all certificates created for the Rita identity.

```
fabric-ca-client revoke -e Rita -r 'HackingAttempt' -u
http://courseParticipantsCA:SimplePassword@localhost:8000
```

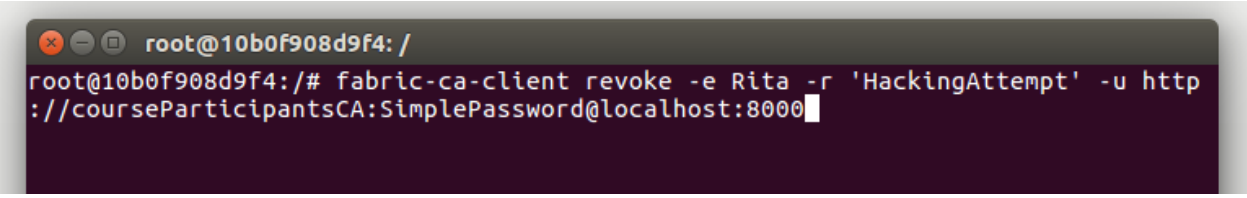

Figure 199 - Snippet 8.30