Zero To Blockchain Documentation

Release 1.4.2

Bob Dill

Oct 31, 2018

Contents:

CHAPTER 1

Objectives

Enable anyone who has a basic understanding of JavaScript, HTML and CSS to build real Blockchain applications using Hyperledger Composer and deploying their Demo or Proof of Concept to the IBM Cloud using a Free Cluster.

1.1 Course

1.1.1 Chapter 1: What is Blockchain? Concept and Architecture Overview

Introduction

Online, free tutorial on getting started with Blockchain and IBM Bluemix

Objectives

The tutorial will build a blockchain solution on IBM Bluemix using

- HyperLedger Composer V 0.10 or higher.
- Hyperledger Fabric V1.0 or higher
- NodeJS
- HTML, CSS, Javascript

Resources

- All code for the tutorial is in github: - Zero To Blockchain Git Repository
- A basic introduction to coding for Bluemix, getting an id, setting up your workstation can be found in the popular ZeroToCognitive series: - Zero To Cognitive YouTube Series

1.1.2 Chapter 2: What's the Story we'll implement?

Introduction

- Dispute resolution requires gathering and correlating facts from multiple parties. This is a slow and labor intensive effort, which ties up over $100M at any point in time.
- In this example, those parties are: - The Finance Organization - The Buyer - The Seller - The Provider - The Shipper

The Basic Stories

- As a Finance Organization, I want to see the finance related status of every order executed by my clients when they are using my credit services instantly and in real-time
- This will allow me to manage dispute resolution over the phone immediately rather than taking multiple weeks to resolve a dispute.
- As a seller, I want to see the order, shipping and finance status of every sale in the system.
- As a buyer, I want to see the real-time status of every order.
- As a buyer, I want to be able to initiate a dispute with the click of a single button and provide all required data automatically to my finance organization
- As a manufacturer, I want to be able to see all open orders and the shipment status on all orders.
- As a shipper, I want to be able to interact with this system with as little change as possible on my end.

1.1.3 Chapter 3: Creating the Blockchain Environment

Introduction

The HyperLedger Fabric development environment, which is used as the foundation for HyperLedger Composer development, is only available for direct installation on OSX and on on Ubuntu V14 LTS and V16 LTS. Knowing that there are many Windows based developers, this chapter includes installation instructions for Windows as well as OSX and Ubuntu. We will use the free VirtualBox tool to install Ubuntu and then do our development there.

The installation instructions have been tested in the following configurations:

- OSX Sierra, High Sierra
- Ubuntu 64bit (V16 LTS) Guest on OSX Sierra
- Ubuntu 64bit (V16 LTS) Native
- Ubuntu 64bit (V16 LTS) Guest on Windows 7

The first section tells you how to install on OSX, Ubuntu or Windows.

The end of this section covers getting your own personal copy of the full tutorial.

Install on OSX

The installation process for OSX has been automated and is designed to be invoked via a single command. You will need to be network connected for the duration of the installation process, which will take 15 minutes to a couple of hours, depending on your available network speed and capacity.

- Open a terminal window and navigate to your github folder.

- Execute the following command on a single line:

```
curl -H 'Accept: application/vnd.github.v3.raw' https://raw.
↪githubusercontent.com/rddill-IBM/ZeroToBlockchain/master/setup_OSX.sh |␣
↪bash
```

- This will download an installation exec from the ZeroToBlockchain repository and execute it on your system. This command will go through the following steps, informing you of each step in the process
  - Check for the presence of the Brew package manager and install it if it's missing. If it's present, execute the brew update and brew upgrade commands, install the dos2unix tool
  - Check for the presence of git and install it if it's missing
  - Install GitHub Desktop V 2.33
  - Check for the presence of nodeJS version 8 (Required for hyper ledger) and install it if it's missing
  - Install the nodejs SDK for hyper ledger composer
  - Install the hyper ledger fabric docker images
  - Install the fabric tools and update your .bash_profile
  - Install hyper ledger composer platform-specific binaries

If you do not want to automatically take all of these actions, then execute the following command, instead, to just download the

```
curl -H 'Accept: application/vnd.github.v3.raw' https://raw.githubusercontent.com/
↪rddill-IBM/ZeroToBlockchain/master/setup_OSX.sh -o setup_OSX.sh
```

then type in: `dos2unix setup_OSX.sh`, which will ensure that this will now run correctly as a script file on your OSX operating system.

You have the ability to turn off any of the installation services. This is dependent on using the Brew package manager to handle various installation requirements, so if you don't want to use Brew, you'll need to install each service manually.

- To get help for the script, just type in `./setup_OSX.sh -h`, which will cause the following information to print out:

```
   installation script for the Zero To Blockchain Series
   This is for Mac OSX ONLY
   This script will check to see if HomeBrew is installed
       and install it if it's not already present.
   It will then execute a brew update and brew upgrade to ensure
       that you are at the latest release of your brew installed packages
   dos2unix is installed by brew to correct scripts from hyperledger and␣
↪composer
   The exec will proceed with checking to ensure you are at Node V6
       which is required for working with HyperLedger Composer
   The script will then install the nodejs SDK for hyperledger and␣
↪composer
   The script will finish by downloading the docker images for␣
↪hyperledger

options for this exec are:
   -h Print this help information
   -g defaults to true. use -g false if you do not want to have git␣
↪installation checked
```

(continues on next page)

(continued from previous page)

```
  -b defaults to true. use -b false if you do not want to have brew␣
↪installation checked
  -n defaults to true. use -n false if you do not want to have node␣
↪installation checked
  -s defaults to true. use -s false if you do not want to have node SDK␣
↪installation checked
  -d defaults to true. use -d false if you do not want to have␣
↪hyperledger node images verified
  -p defaults to /Users/robertdill/fabric-tools. use -p /Users/
↪robertdill/your/preferred/path/fabric-tools/your/path/here if you want␣
↪to install hyperledger fabric tools elsewhere.
      only valid with -d true, ignored otherwise
```

- So if you want to install only the node services (that is, don't install git or brew), then you would type in the following: `./setup_OSX.sh -g false -b false`, which turns off the installation options for git and brew.

Once you have run this exec, your development environment will be installed. Before you can run it, you'll have to start Docker. Press CMD+spacebar and type in docker. Highlight Docker.app and press enter. On most OSX systems, Docker.app will automatically restart each time you start up, or restart OSX.

Install on Ubuntu

The installation process for Ubuntu has been automated and is designed to be invoked via a single command. You will need to be network connected for the duration of the installation process, which will take 15 minutes to a couple of hours, depending on your available network speed and capacity.

- Open a terminal window and navigate to your github folder.
- `sudo apt-get update`
- `sudo apt-get install -y curl`
- After curl has been installed, copy the following as a single line into a terminal window and press enter

```
curl -H 'Accept: application/vnd.github.v3.raw' https://raw.
↪githubusercontent.com/rddill-IBM/ZeroToBlockchain/master/setup_Ubuntu_Part_
↪1.sh | bash
```

- This will install all required software up through docker. If you do not want to automatically install this software, then first download the the script and then use the available options to limit what it does.
- To download the script, execute the following command:

```
curl -H 'Accept: application/vnd.github.v3.raw' https://raw.
↪githubusercontent.com/rddill-IBM/ZeroToBlockchain/master/setup_
↪Ubuntu_Part_1.sh -o setup_Ubuntu_Part_1.sh
```

- The type in `dos2unix setup_Ubuntu_Part_1.sh`
- Then type in `./setup_Ubuntu_Part_1.sh`
- Then type in `./setup_Ubuntu_Part_1.sh -h`, which will print out the following information:

```
    installation script for the Zero To Blockchain Series
   This is for Linux ONLY. It has been tested only on Ubuntu 16.04 LTS
   Other versions of Linux are not supported via this script.
   The following will be downloaded by this script
   dos2unix, to correct scripts from hyperledger and composer
   docker for Ubuntu
   The exec will proceed with checking to ensure you are at Node V6
   which is required for working with HyperLedger Composer
   The script will then install the nodejs SDK for hyperledger and␣
↪composer
   The script will finish by requesting that you reboot your system


options for this exec are:
   -h Print this help information
   -g defaults to true. use -g false if you do not want to have git␣
↪installation checked
   -n defaults to true. use -n false if you do not want to have node␣
↪installation checked
   -v defaults to true. use -v false if you do not want to have vscode␣
↪installed
   -s defaults to true. use -s false if you do not want to have node SDK␣
↪installation checked
   -d defaults to true. use -d false if you do not want to have docker␣
↪installed
```

- type in ./setup_Ubuntu_Part_1.sh followed by any flag (e.g. -g) followed by the word false to disable the installation of that service.
- ***You will then need to reboot your system prior to executing the following command:***

```
curl -H 'Accept: application/vnd.github.v3.raw' https://raw.
↪githubusercontent.com/rddill-IBM/ZeroToBlockchain/master/setup_Ubuntu_Part_
↪2.sh | bash
```

- This will complete the installation of the hyper ledger images and the supporting tools.
- This command will go through the following steps, informing you of each step in the process
  - Update the apt-get repositories and upgrade current software
  - Ensure that the base development environment is installed
  - Check for the presence of git and install it if it's missing
  - Check for the presence of nodeJS version 8 (Required for hyper ledger) and install it if it's missing
  - Install the nodejs SDK for hyper ledger composer
  - Install the VSCode editor
  - Install the hyper ledger fabric docker images
  - Install the fabric tools and update your .bash_profile
  - Install hyper ledger composer platform-specific binaries

If you do not want to automatically take all of these actions, then execute the following command, instead, to just download the file to your system:

```
 curl -H 'Accept: application/vnd.github.v3.raw' https://raw.
↪githubusercontent.com/rddill-IBM/ZeroToBlockchain/master/setup_Ubuntu_Part_
↪2.sh -o setup_Ubuntu_Part_2.sh

- then type in: ``dos2unix setup_Ubuntu_Part_2.sh``, which will ensure that␣
↪this will now run correctly as a script file on your  operating system.
```

You have the ability to turn off any of the installation services. This is dependent on using the Brew package manager to handle various installation requirements, so if you don't want to use Brew, you'll need to install each service manually.

- To get help for the script, just type in `./setup_Ubuntu_Part_2.sh -h`, which will cause the following information to print out:

```
   installation script for the Zero To Blockchain Series
   This is for Linux ONLY. It has been tested on Ubuntu 16.04 LTS
   Other versions of Linux are not supported via this script.
   The following will be downloaded by this script
   The script will finish by downloading the docker images for hyperledger


options for this exec are:
   -h Print this help information
   -f defaults to true. use -f false if you do not want to have hypleledger␣
↪fabric images verified
   -p defaults to /home/robertdill/fabric-tools. use -p /home/robertdill/your/
↪preferred/path/fabric-tools/your/path/here if you want to install hyperledger␣
↪fabric tools elsewhere.
   only valid with -f true, ignored otherwise
```

- Once you have run this exec, your development environment will be installed. Before you can run it, you'll have to start Docker. Press CMD+spacebar and type in docker. Highlight Docker.app and press enter. On most OSX systems, Docker.app will automatically restart each time you start up, or restart Ubuntu.

Install on Windows V7 or V10 (summary)

Because HyperLedger Fabric does not yet support Windows Native installation, we will install an Ubuntu guest operating system on your Windows computer. The following installation instructions use VirtualBox to create an environment for Ubuntu. You can also use VMWare or any technology of your choice which will allow you to install the 64bit version of Ubuntu V16 LTS (Long Term Service).

The installation process for Windows is: Step 1: download a 64-bit Ubuntu 16.04 image. The installation will fail if you install a 32-bit image

- Go here: https://www.ubuntu.com/download
- click on the Ubuntu Desktop option - this is a large download and will take 30+ minutes

Step 2: download and install VirtualBox

- Go here: https://www.virtualbox.org/wiki/Downloads
- And click on Windows Hosts. This will start the download of the VirtualBox installer

Step 3: Go to your downloads folder and run the VirtualBox .... .exe installer.

- Take the defaults

Step 4: Start Virtual Box

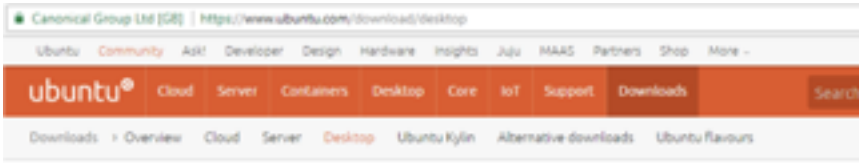
- Click on the "new" icon

- Take all the defaults EXCEPT memory.
- Give your virtual machine at least 2Gb rather than the 1Gb default. If you have sufficient memory, I recommend 4-8Gb.

Step 5: Follow the steps for an Ubuntu installation on the following pages.

Install on Windows V7 or V10 (details)

Step 1: download a 64-bit Ubuntu 16.04 image.



- **Go here: https://www.ubuntu.com/download**
- and download V16 of the Ubuntu Desktop, which provides a graphical UI similar to OSX and Windows.
- You may find that you have to choose between an Intel or I386 version and an AMD version. Regardless of what technology is on your computer, pick the AMD version. These are poorly named. AMD represents the 64bit version, which runs on Intel and AMD processors and the Intel or I386 version is the 32 bit version, which you do NOT want.

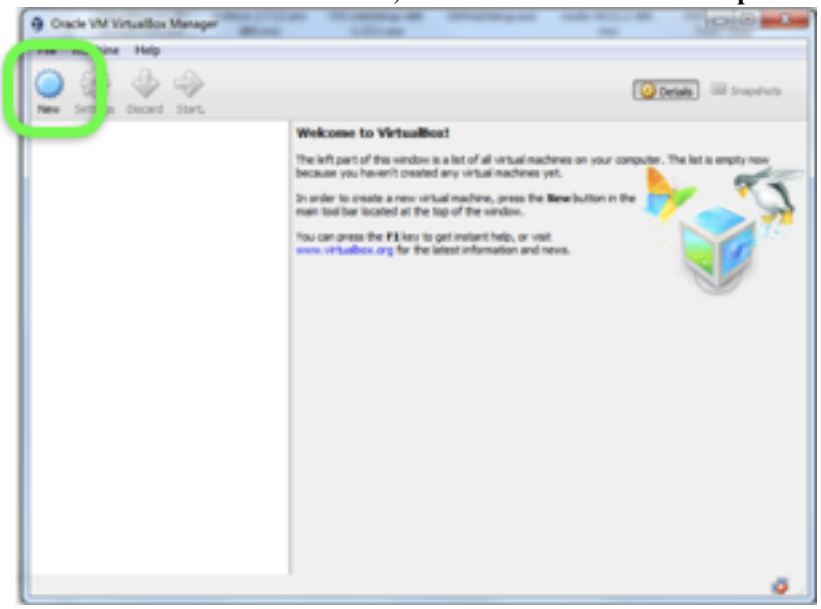
Step 2: download and install VirtualBox

- Go here: https://www.virtualbox.org/wiki/Downloads
- **And click on Windows Hosts. This will start the download of the VirtualBox installer**

- After the download completes, run the virtualbox exe which was placed in your download folder

Step 3: Go to your downloads folder and run the VirtualBox .... .exe installer

- **Now that Virtual Box has been installed, start it and select the 'New' option:**

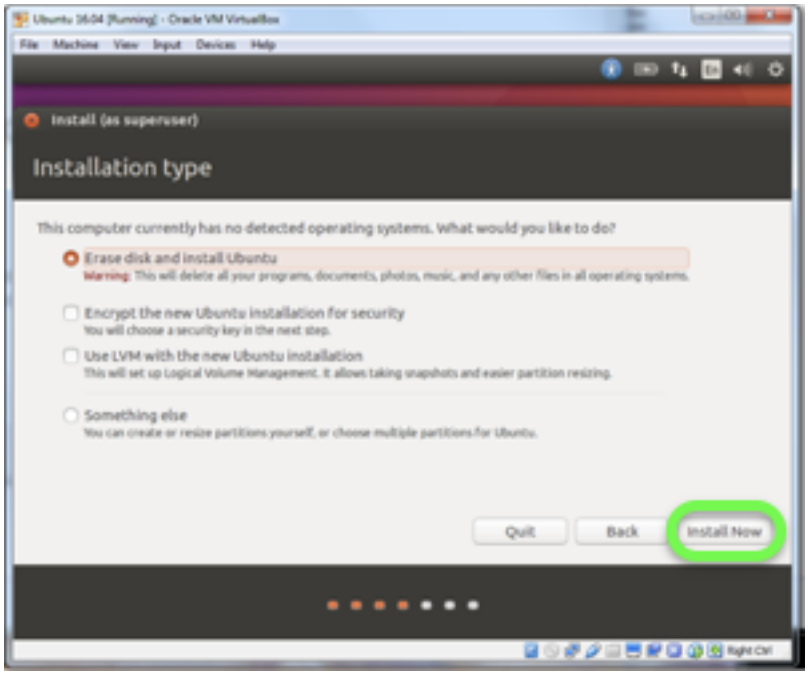


- Select Linux and Ubuntu 64 bit .. image:: ../../assets/WindowsVirtualBox3.png
- give yourself at least 2Gb of ram, more if you can afford it. .. image:: ../../assets/WindowsVirtualBox4.png
- Create a virtual drive .. image:: ../../assets/WindowsVirtualBox5.png
- and take the default VDI option .. image:: ../../assets/WindowsVirtualBox6.png
- Allocate storage dynamically .. image:: ../../assets/WindowsVirtualBox7.png
- **Give yourself at least 20Gb.** This is the minimum for developing these applications .. image:: ../../assets/WindowsVirtualBox8.png

- Select Next .. image:: ../../assets/WindowsVirtualBox9.png

Step 4: Start Virtual Box

- Select the image you downloaded .. image:: ../../assets/WindowsVirtualBox10.png
- After the machine starts, select the "Install Ubuntu Desktop" option .. image:: ../../assets/WindowsUbuntu2.png
- Select options to update the installation and to install 3rd party code .. image:: ../../assets/WindowsUbuntu3.png



Step 5: Follow the steps for an Ubuntu installation

1.1.4 Chapter 4: Building your first Business Network

Introduction

This chapter focuses on the following files:

```
Chapter 04
 network
permissions.acl
queries.qry
    lib
    sample.js
    models
    base.cto
    events.cto
    sample.cto
    test
    sample.js
```

Key Files

Key files:

- **models/sample.cto** - This is the 'model' file, which defines the types of members and assets in a business network and also establishes the speicif transactions which can be executed in a network.
- **lib/sample.js** - The sample.js file in the lib folder contains all of the code to implement each transaction defined in the sample.cto file.
- **test/sample.js** - This file provides the unit tests for the network.

Installing the code

- run this command from a terminal window: `` `npm install` `` - when you run `` `npm install` `` command, you will get the following error, which you can ignore. The error is caused by the installation process for the SDK tooling, which searches for all instances of files ending in `` `.cto` `` . Because we have those files in the answers folder, as well as in the (correctly placed) network/models folder, npm install gets confused. It's ok, the process will still work correctly.

```
Looking for package.json of Business Network Definition
    Input directory: /Users/robertdill/Documents/GitHub/Z2B_Master/Chapter04
    Error: namespace already exists
        at ModelManager.addModelFiles (/usr/local/lib/node_modules/composer-cli/
↪node_modules/composer-common/lib/modelmanager.js:241:31)
        at Function._processModelFiles (/usr/local/lib/node_modules/composer-cli/
↪node_modules/composer-common/lib/businessnetworkdefinition.js:457:43)
        at Promise.resolve.then (/usr/local/lib/node_modules/composer-cli/node_
↪modules/composer-common/lib/businessnetworkdefinition.js:620:18)
        at process._tickCallback (internal/process/next_tick.js:109:7)
        at Module.runMain (module.js:606:11)
        at run (bootstrap_node.js:383:7)
        at startup (bootstrap_node.js:149:9)
        at bootstrap_node.js:496:3
    Error: namespace already exists
        at ModelManager.addModelFiles (/usr/local/lib/node_modules/composer-cli/
↪node_modules/composer-common/lib/modelmanager.js:241:31)
        at Function._processModelFiles (/usr/local/lib/node_modules/composer-cli/
↪node_modules/composer-common/lib/businessnetworkdefinition.js:457:43)
        at Promise.resolve.then (/usr/local/lib/node_modules/composer-cli/node_
↪modules/composer-common/lib/businessnetworkdefinition.js:620:18)
        at process._tickCallback (internal/process/next_tick.js:109:7)
        at Module.runMain (module.js:606:11)
        at run (bootstrap_node.js:383:7)
        at startup (bootstrap_node.js:149:9)
        at bootstrap_node.js:496:3 ```
```

- then run `` `buildAndDeploy` `` (yes, case is important)
- then run `` `npm run test` ``
- this should deliver the following result:

```
Finance Network
    #createOrder
    ✓ should be able to create an order (134ms)
    #issueBuyRequest
    ✓ should be able to issue a buy request (76ms)
    #issueOrderFromSupplier
    ✓ should be able to issue a supplier order (64ms)
    #issueRequestShipment
    ✓ should be able to issue a request to ship product (51ms)
```

(continues on next page)

(continued from previous page)

```
    #issueDelivery
    ✓ should be able to record a product delivery
    #issueRequestPayment
    ✓ should be able to issue a request to request payment for a product (58ms)
    #issuePayment
    ✓ should be able to record a product payment
    #issueDispute
    ✓ should be able to record a product dispute
    #issueResolution
    ✓ should be able to record a dispute resolution (41ms)
    #issueBackorder
    ✓ should be able to record a product backorder


10 passing (2s)
```

The business network definitions are contained in the **sample.cto** file, which includes the *base.cto* and the *events.cto* file.

Participant - *Buyer - Seller - Provider - Shipper - FinanceCo*

Asset - *Order*

- *CreateOrder*
- *Buy*
- *OrderFromSupplier*
- *RequestShipping*
- *Deliver*
- *RequestPayment*
- *Pay*
- *Dispute*
- *Resolve*
- *Backorder*

Event - *(none yet)*

Orders are created by Buyers and executed by Sellers, who may work with a 3rd part (Provider) to fulfill the order. Either Sellers or Providers can RequestShipment, which is fulfilled by a Shipper who executes a Deliver transaction when complete. Orders can be Disputed. Disputed Orders can be resolved. Payments are made against either Delivered or Resolved Orders.

To test this Business Network Definition in the **Test** tab:

Create an *Order* asset:

```
asset Order identified by orderNumber {
    o String orderNumber
    o String status
    o Integer amount
    o String created
    o String bought
    o String ordered
    o String dateBackordered
```

(continues on next page)

(continued from previous page)

```
    o String requestShipment
    o String delivered
    o String disputeOpened
    o String disputeResolved
    o String paymentRequested
    o String orderRefunded
    o String paid
    o String[] vendors
    o String dispute
    o String resolve
    o String backorder
    o String refund
    --> Buyer buyer
    --> Seller seller
```

Create a participant:

```
participant Buyer identified by buyerID {
    o String buyerID
    o String companyName
}
participant Seller identified by sellerID {
    o String sellerID
    o String companyName
}

asset Order identified by orderNumber {
    o String orderNumber
    o String status
    o Integer amount
    o String created
    o String bought
    o String ordered
    o String dateBackordered
    o String requestShipment
    o String delivered
    o String disputeOpened
    o String disputeResolved
    o String paymentRequested
    o String orderRefunded
    o String paid
    o String[] vendors
    o String dispute
    o String resolve
    o String backorder
    o String refund
    --> Buyer buyer
    --> Seller seller

}
participant Shipper identified by shipperID {
    o String shipperID
    o String companyName
}
participant Provider identified by providerID {
    o String providerID
    o String companyName
```

(continues on next page)

(continued from previous page)

```
}
participant FinanceCo identified by financeCoID {
    o String financeCoID
    o String companyName
}
```

Submit a transaction:

```
transaction CreateOrder {
    o Integer amount
    --> Order order
    --> Buyer buyer
    --> Seller seller
}
transaction Buy {
    --> Order order
    --> Buyer buyer
    --> Seller seller
}
transaction OrderFromSupplier {
    --> Order order
    --> Provider provider
}
transaction RequestShipping {
    --> Order order
    --> Shipper shipper
}
transaction Deliver {
    --> Order order
    --> Shipper shipper
}
transaction BackOrder {
    o String backorder
    --> Order order
    --> Provider provider
}
transaction Dispute {
    o String dispute
    --> Order order
    --> Buyer buyer
    --> Seller seller
    --> FinanceCo financeCo
}
transaction Resolve {
    o String resolve
    --> Order order
    --> Buyer buyer
    --> Seller seller
    --> FinanceCo financeCo
}
transaction RequestPayment {
    --> Order order
    --> Buyer buyer
    --> Seller seller
    --> FinanceCo financeCo
}
transaction Pay {
```

(continues on next page)

(continued from previous page)

```
    --> Order order
    --> Seller seller
    --> FinanceCo financeCo
}
transaction Refund {
    o String refund
    --> Order order
    --> Buyer buyer
    --> Seller seller
    --> FinanceCo financeCo
}
```

1.1.5 Chapter 5: Building the Admin Services and User Experience

Introduction

In this chapter, we will build a nodeJS web server which will interact with the HyperLedger Composer defined Blockchain and create an administrative interface for the network. This runs on HyperLedger Fabric Version 1.0. The code that we will write is managed in this chapter and all the subsequent chapters in the following folder structure:

```
Chapter 05
 controller
    restapi
    router.js
        features
        composer
            autoLoad.js
            hlcAdmin.js
            queryBlockChain.js
            Z2B_Services.js
            Z2B_Utilities.js
         text
            multi-lingual.js
            resources.js
 HTML
    CSS
    pageStyles.css
    js
    z2b-admin.js
    z2b-events.js
    z2b-initiate.js
    z2b-utilities.js
 network
permissions.acl
queries.qry
    lib
    sample.js
    models
    base.cto
    events.cto
    sample.cto
```

Common Web Server Code

Some of these files are used by all chapters and some are specific to this chapter. Code explanations are embedded in the javascript files. Those files are:

- **index.js** this is the master, or controller, file for the web server. It loads a file called 'router.js' to implement all business functionality.
- **autoload.js** sometimes it helps to have members and assets preloaded into the business network. This module loads that data from a file called memberList.json in the /startup folder
- **queryBlockchain.js** As you are running the autoloader exec and working with chapters 5-12, you will see blocks being created. This modules listens to the blockchain events and sends that data to the browser.
- **Z2B_Services.js** This module contains transaction management routines which are called either by the autoLoader.js module or hlcClient.js module.
- **Z2B_Utilities.js** This module contains utility services, including socket creation, which are called by autoLoader, hlcClient, hlcAdmin, queryBlockChain modules
- **multi-lingual.js** This module contains services to support multiple languages on the browser. Language text is provided in subfolders in the composer/text folder. Which languages are available is managed in the languages.json file.

Common Web Browser code

- **z2b-utilities.js** This file contains routines which are used by the browser javascript.
- **z2b-initiate.js** This file contains the code necessary to start up the web application. This is where the default starting language is specified

Web Server Code Unique to this Chapter

- **hlcAdmin.js** This contains all code to support the administrative interface on the server side

Defining the business network

- **network/model/sample.cto** The CTO file contains the network description and includes member (buyer, seller, provider, shipper, financeCo), asset (Order) and transaction definitions. The sample.cto file uses two other files as it's foundation: **base.cto** and **events.cto**
- **network/lib/sample.js** This contains the transaction code which will be deployed to the blockchain. Although normally written in GO, hyperledger composer allows us to write all of the blockchain code in javascript and then uses a plug-in called DukTape to enable the javascript to be executed within the GO environment. This file must implement transactions as defined in the sample.cto file and is one of the places where we ensure that members have the authority to execute the requested transaction.
- **network/permissions.acl** This file determines precisely what transactions a member can execute and what assets a member can see. The basic rule is "If authority is not specifically granted in this file, then access is denied."

Web Browser Code Unique to this Chapter

- **z2b-admin.js** This file contains all of the browser functional code to support the administrative interface **Please Note** that with the update to hyperledger-composer v0.15, composer has transitioned from using connection profiles to using idCards. This means that the connection profile creation work is no longer required to connect to the network.

- **index.html** This file is the initial web page loaded by the application. Text on this page is determined by the selected language (default is US English)
- **admin.html** Contains the HTML to manage the admin interface Text on this page is determined by the selected language **Please Note** that with the update to hyperledger-composer v0.15, composer has transitioned from using connection profiles to using idCards. This means that the connection profile creation work is no longer required to connect to the network. Therefore, these items now show a yellow strike-through in this chapter and will not be in chapters 6-12.
- **createConnectionProfile.html** Contains the HTML to manage profile creation Text on this page is determined by the selected language
- **createMember.html** contains the HTML to manage member creation Text on this page is determined by the selected language
- **deleteConnectionProfile.html** contains the HTML to manage connection deletion Text on this page is determined by the selected language
- **getMemberSecret.html** contains the HTML to manage and display member secrets Text on this page is determined by the selected language
- **removeMember.html** contains the HTML to remove a member Text on this page is determined by the selected language

Where are the Answers?

Chapterxx/Documentation/answers

- all answers for each chapter are in the *Documentation/answers* folder in that chapter.
- The naming conventions used are:
- ***Folder Names:***
  - composer: these files are the answers to files located in the following path within each chapter: `controller/restapi/features/composer/`
  - cto: `network/model`
  - CSS: `HTML/CSS`
  - HTML: `/HTML/`
  - js: `HTML/js/`
  - lib: `network/lib`
  - test: `network/test`
- ***File Name:***
  - each file has the same prefix in the answers and in the target folder. The answers folder adds '_complete' to the end of the first part of the file name. For example:
  - the server code for administration is in the ***controller/restapi/features/composer*** folder and is called: hlcAdmin.js. The completed code for that file (in this chapter) is in the ***Documentation/answers/composer*** folder and is called hlcAdmin_complete.js

How do I get everything started once I've updated all the code?

You need to run the following command **once** for each chapter: `npm install`. This installs all of the node modules required to run your program.

Each time you restart your system, restart Docker, or bring your system back from a 'suspend' state, you need to run the follow

- buildAndDeploy (OSX) . . . or . . . ./buildAndDeploy (Ubuntu)
- npm start (or npm index)

The `buildAndDeploy` command creates the business archive, stops and restarts the docker containers and then deploys your network into those containers.

The `npm start` command loads and starts to execute the index.js file in your chapterxx folder. This, in turn, loads all of the services in the controller folder structure.

Why All The Changes? **HyperLedger Composer has gone through two significant changes in 4Q2017.**

- The first, Version 0.14, changed how security worked and clarified the difference between the business network administrator (admin) and the fabric Administrator (PeerAdmin). While this had little effect on what the Admin user interface could do, it did change significantly how the underlying code worked in the hlcAdmin.js file and in the autoLoad.js file. The biggest code change is that the autoLoad process started issuing Identities instead of just adding members to the registry. The other significant change required adding two statements to the permissions.acl file.
- Then Version 0.15 was released and the entire access structure was changed from connection profile + username/password (which is why we had the getSecret() method) to idCards. This necessitated changes in some of the scripts we use (buildAndDeploy, startup.sh, deployNetwork.sh), required a complete rewrite for how the test code loads and connects to the network and also required changes in the admin interface. You'll see a number of items which have a –>yellow strikethrough<–, indicating that they are no longer available.
- We aren't using connection profiles any more as part of the log in process, so that whole section is no longer useful. Because we're using cards, we don't need to use, nor do we need to capture, the user secrets associated with each Member. so that is deactivated.

We do, however, need more capability for Members. Along with the Add Member function, which we've had all along, we now need the ability to issue an Identity for that Member and then to create an idCard for that member. You'll see those new functions in the Resource Management part of the administrative user interface, along with a feature which will check to see if an idCard already exists for a specific member.

An Identity can only be issued for an existing Member, so the process is to:

- Add a Member
- Issue an Identity
- Create an idCard

Creating an idCard requires the password, or secret, which was generated during the issue Identity process. Each of these functions has been implemented individually. It would be a great idea to combine the issue Identity process with the Create idCard process, so that you don't have to remember to copy the generated secret and type it into the createCard page.

1.1.6 Chapter 6: Building the Buyer Services and User Experience

Introduction

In this chapter, we will create the Buyer Company view and add server-side services to support it. This runs on HyperLedger Fabric Version 1.0. The code that we will write is managed in this chapter and all the subsequent chapters in the following folder structure:

```
Chapter 06
 controller
    restapi
    router.js
        features
        composer
            autoLoad.js
            hlcAdmin.js
            hlcClient.js
            queryBlockChain.js
            Z2B_Services.js
            Z2B_Utilities.js
         text
            multi-lingual.js
            resources.js
 HTML
    index.html
    admin.html
    buyer.html
    ceateConnectionProfile.html
    createMember.html
    createOrder.html
    deleteConnectionProfile.html
    getMemberSecret.html
    removeMember.html
    CSS
    pageStyles.css
    js
    z2b-admin.js
    z2b-buyer.js
    z2b-events.js
    z2b-initiate.js
    z2b-utilities.js
 network
permissions.acl
queries.qry
    lib
    sample.js
    models
    base.cto
    events.cto
    sample.cto
```

Four new files are introduced in this chapter and one existing file is updated:

Web Server Code Unique to this Chapter

- **hlcClient.js** This contains all code to support the buyer interface on the server side

Defining the business network

- **network/permissions.acl** This file is ***updated*** to allow the buyer to access Orders and to execute selected transactions. It determines precisely what transactions a member can execute and what assets a member can see. The basic rule is "If authority is not specifically granted in this file, then access is denied."

Web Browser Code Unique to this Chapter

- **z2b-buyer.js** This file contains all of the browser functional code to support the buyer user experience
- **buyer.html** This contains the HTML framework for the buyer experience. Content for this framework is generated by the functions in the z2b-buyer.js file Text on this page is determined by the selected language (default is US English)
- **createOrder.html** Contains the HTML framework to manage buyer order creation. Content for this framework is generated by the functions in the z2b-buyer.js file Text on this page is determined by the selected language

1.1.7 Chapter 7: Building the Seller Services and User Experience

Introduction

In this chapter, we will create the Seller Company view and add server-side services to support it. This runs on HyperLedger Fabric Version 1.0. The code that we will write is managed in this chapter and all the subsequent chapters in the following folder structure:

```
Chapter 07
 controller
    restapi
    router.js
        features
        composer
            autoLoad.js
            hlcAdmin.js
            hlcClient.js
            queryBlockChain.js
            Z2B_Services.js
            Z2B_Utilities.js
         text
            multi-lingual.js
            resources.js
 HTML
    index.html
    admin.html
    buyer.html
    ceateConnectionProfile.html
    createMember.html
    createOrder.html
    deleteConnectionProfile.html
    getMemberSecret.html
    removeMember.html
    seller.html
    CSS
    pageStyles.css
    js
    z2b-admin.js
    z2b-buyer.js
    z2b-events.js
    z2b-initiate.js
    z2b-seller.js
    z2b-utilities.js
 network
permissions.acl
```

(continues on next page)

(continued from previous page)

```
queries.qry
    lib
    sample.js
    models
    base.cto
    events.cto
    sample.cto
```

Two new files are introduced in this chapter and two existing files are updated:

Web Server Code Unique to this Chapter

- **hlcClient.js** This file is update to support the seller interface on the server side

Defining the business network

- **network/permissions.acl** This file is ***updated*** to allow the seller to access Orders and to execute selected transactions. It determines precisely what transactions a member can execute and what assets a member can see. The basic rule is "If authority is not specifically granted in this file, then access is denied."

Web Browser Code Unique to this Chapter

- **z2b-seller.js** This file contains all of the browser functional code to support the seller user experience
- **seller.html** This contains the HTML framework for the seller experience. Content for this framework is generated by the functions in the z2b-seller.js file Text on this page is determined by the selected language (default is US English)

1.1.8 Chapter 8: Building the Provider Services and User Experience

Introduction

In this chapter, we will create the (third party) Provider Company view and add server-side services to support it. This runs on HyperLedger Fabric Version 1.0. The code that we will write is managed in this chapter and all the subsequent chapters in the following folder structure:

```
Chapter 08
 controller
    restapi
    router.js
        features
        composer
            autoLoad.js
            hlcAdmin.js
            hlcClient.js
            queryBlockChain.js
            Z2B_Services.js
            Z2B_Utilities.js
         text
            multi-lingual.js
            resources.js
```

(continues on next page)

(continued from previous page)

```
 HTML
    index.html
    admin.html
    buyer.html
    ceateConnectionProfile.html
    createMember.html
    createOrder.html
    deleteConnectionProfile.html
    getMemberSecret.html
    removeMember.html
    provider.html
    seller.html
    CSS
    pageStyles.css
    js
    z2b-admin.js
    z2b-buyer.js
    z2b-events.js
    z2b-initiate.js
    z2b-provider.js
    z2b-seller.js
    z2b-utilities.js
 network
permissions.acl
queries.qry
    lib
    sample.js
    models
    base.cto
    events.cto
    sample.cto
```

Two new files are introduced in this chapter and two existing files are updated:

Web Server Code Unique to this Chapter

- **hlcClient.js** This file is update to support the provider interface on the server side

Defining the business network

- **network/permissions.acl** This file is ***updated*** to allow the provider to access Orders and to execute selected transactions. It determines precisely what transactions a member can execute and what assets a member can see. The basic rule is "If authority is not specifically granted in this file, then access is denied."

Web Browser Code Unique to this Chapter

- **z2b-provider.js** This file contains all of the browser functional code to support the provider user experience
- **provider.html** This contains the HTML framework for the provider experience. Content for this framework is generated by the functions in the z2b-provider.js file Text on this page is determined by the selected language (default is US English)

1.1.9 Chapter 9: Building the Shipper Services and User Experience

Introduction

In this chapter, we will create the Shipping Company view and add server-side services to support it. This runs on HyperLedger Fabric Version 1.0. The code that we will write is managed in this chapter and all the subsequent chapters in the following folder structure:

```
Chapter 09
 controller
    restapi
    router.js
        features
        composer
            autoLoad.js
            hlcAdmin.js
            hlcClient.js
            queryBlockChain.js
            Z2B_Services.js
            Z2B_Utilities.js
         text
            multi-lingual.js
            resources.js
 HTML
    index.html
    admin.html
    buyer.html
    ceateConnectionProfile.html
    createMember.html
    createOrder.html
    deleteConnectionProfile.html
    getMemberSecret.html
    removeMember.html
    provider.html
    seller.html
    shipper.html
    CSS
    pageStyles.css
    js
    z2b-admin.js
    z2b-buyer.js
    z2b-events.js
    z2b-initiate.js
    z2b-provider.js
    z2b-seller.js
    z2b-shipper.js
    z2b-utilities.js
 network
permissions.acl
queries.qry
    lib
    sample.js
    models
    base.cto
    events.cto
    sample.cto
```

Two new files are introduced in this chapter and three existing files are updated:

Web Server Code Unique to this Chapter

- **hlcClient.js** This file is update to support the shipper interface on the server side

Defining the business network

- **network/permissions.acl** This file is ***updated*** to allow the shipper to access Orders and to execute selected transactions. It determines precisely what transactions a member can execute and what assets a member can see. The basic rule is "If authority is not specifically granted in this file, then access is denied."

Web Browser Code Unique to this Chapter

- **z2b-shipper.js** This file contains all of the browser functional code to support the shipper user experience
- **shipper.html** This contains the HTML framework for the provider experience. Content for this framework is generated by the functions in the z2b-shipper.js file Text on this page is determined by the selected language (default is US English)
- **pageStyles.css** This file is updated to support the accordian action on the Finance Company page

1.1.10 Chapter 10: Building the Cinance Company Services and User Experience

Introduction

In this chapter, we will create the Finance Company view and add server-side services to support it. This runs on HyperLedger Fabric Version 1.0. The code that we will write is managed in this chapter and all the subsequent chapters in the following folder structure:

```
Chapter 10
 controller
    restapi
    router.js
        features
        composer
            autoLoad.js
            hlcAdmin.js
            hlcClient.js
            queryBlockChain.js
            Z2B_Services.js
            Z2B_Utilities.js
         text
            multi-lingual.js
            resources.js
 HTML
    index.html
    admin.html
    buyer.html
    ceateConnectionProfile.html
    createMember.html
    createOrder.html
    deleteConnectionProfile.html
    financeCo.html
    getMemberSecret.html
    removeMember.html
```

(continues on next page)

(continued from previous page)

```
    provider.html
    seller.html
    shipper.html
    CSS
    pageStyles.css
    js
    z2b-admin.js
    z2b-buyer.js
    z2b-events.js
    z2b-financeCo.js
    z2b-initiate.js
    z2b-provider.js
    z2b-seller.js
    z2b-shipper.js
    z2b-utilities.js
 network
permissions.acl
queries.qry
    lib
    sample.js
    models
    base.cto
    events.cto
    sample.cto
```

Two new files are introduced in this chapter and two existing files are updated:

Web Server Code Unique to this Chapter

- **hlcClient.js** This file is update to support the Finance Company interface on the server side

Defining the business network

- **network/permissions.acl** This file is ***updated*** to allow the Finance Company to access Orders and to execute selected transactions. It determines precisely what transactions a member can execute and what assets a member can see. The basic rule is "If authority is not specifically granted in this file, then access is denied."

Web Browser Code Unique to this Chapter

- **z2b-financeCo.js** This file contains all of the browser functional code to support the Finance Company user experience
- **financeCo.html** This contains the HTML framework for the Finance Company experience. Content for this framework is generated by the functions in the z2b-financeCo.js file Text on this page is determined by the selected language (default is US English)

1.1.11 Chapter 11: Building the Unified User Experience

Introduction

In this chapter, we will integrate the User Experiences for Buyer, Seller, Provider and Shipper in a single web page to streamline demonstrating the network.

```
Chapter 11
 controller
    restapi
    router.js
        features
        composer
            autoLoad.js
            hlcAdmin.js
            hlcClient.js
            queryBlockChain.js
            Z2B_Services.js
            Z2B_Utilities.js
         text
            multi-lingual.js
            resources.js
 HTML
    index.html
    admin.html
    buyer.html
    ceateConnectionProfile.html
    createMember.html
    createOrder.html
    deleteConnectionProfile.html
    financeCo.html
    getMemberSecret.html
    removeMember.html
    provider.html
    seller.html
    shipper.html
    singleUX.html
    CSS
    pageStyles.css
    js
    z2b-admin.js
    z2b-buyer.js
    z2b-events.js
    z2b-financeCo.js
    z2b-initiate.js
    z2b-provider.js
    z2b-seller.js
    z2b-shipper.js
    z2b-utilities.js
 network
permissions.acl
queries.qry
    lib
    sample.js
    models
    base.cto
    events.cto
    sample.cto
```

One new file is created and two existing files are updated:

Web Server Code Unique to this Chapter

- **no updates**

Defining the business network

- **no updates**

Web Browser Code Unique to this Chapter

- **singleUX.html** This page loads a 2x2 table, with one cell each for Buyer (top left), Seller (top right), Provider (bottom left) and Shipper (bottom right).
- **z2b-events.js** This the loadSingleUX() function is updated to simultaneously display all four members simultaneously
- **CSS/pageStyles.css** CSS controls the look and feel of a web page. This file is updated to support putting all four roles into a single web page at the same time.

1.1.12 Chapter 12: Implementing Business Events

Introduction

In this chapter, we will update the code from Chapter 11 to include event creation and monitoring on the server and event subscription and notification in the browser.

```
Chapter 12
 controller
    restapi
    router.js
        features
        composer
            autoLoad.js
            hlcAdmin.js
            hlcClient.js          <== Updated
            queryBlockChain.js
            Z2B_Services.js       <== Updated
            Z2B_Utilities.js
         text
            multi-lingual.js
            resources.js
 HTML
    index.html
    admin.html
    buyer.html                     <== Updated
    ceateConnectionProfile.html
    createMember.html
    createOrder.html
    deleteConnectionProfile.html
    financeCo.html                 <== Updated
    getMemberSecret.html
    removeMember.html
    provider.html                  <== Updated
    seller.html                    <== Updated
    shipper.html                   <== Updated
    singleUX.html
    CSS
    pageStyles.css
    js
    z2b-admin.js
    z2b-buyer.js               <== Updated
    z2b-events.js              <== Updated
    z2b-financeCo.js           <== Updated
```

(continues on next page)

(continued from previous page)

```
    z2b-initiate.js
    z2b-provider.js             <== Updated
    z2b-seller.js               <== Updated
    z2b-shipper.js              <== Updated
    z2b-utilities.js
 network
permissions.acl
queries.qry
    lib
    sample.js                    <== Updated
    models
    base.cto
    events.cto
    sample.cto                   <== Updated
```

No new files are created. Fifteen existing files are updated:

Web Server Code Unique to this Chapter

- **hlcClient.jss**
  - add event monitoring to client routines
  - add event notification to browser via web socket. - it should be noted that this is a rough implementation for notifying the user of asynchronous events and is appropriate for a lightweight demo but inappropriate for a PoC or a production system.

Defining the business network

- **/models/sample.CTO** add events to sample.CTO file
- **/lib/sample.js** add event notification to transactions

Web Browser Code Unique to this Chapter

- **all UX files** Add placeholder for notification icon and add placeholder for notification counter.
- **all UX javascript files** use subscribe, unsubscribe and notifyMe routines
- **z2b-events.js** add event support for alert subscription, notification
- **CSS/pageStyles.css** add support for notification icon and text

1.1.13 Chapter 13: Deploying your demo Blockchain to the IBM Cloud

(1) network information

- Docker and the Kubernetes deploy use different names for the CA and for the channel
  - Docker CA: ca.example.org channel: composer
  - Kubernetes CA: CA channel: channel1
- The demo when running locally uses localhost:xxxx for all access - need to get the address of the kuberenetes cluster - and update the PeerAdmin.card with that address

(2) fabric information

HyperLedger fabric always uses ~/.hfc-key-store to hold keys for direct access to the fabric.

- This means that this folder must also exist in the IBM Cloud deployment
- And must have keys made just for this deployment.
- That requires creating the PeerAdmin information and storing it for later load to ~/.hfc-key-store
- and then creating an exec to create that folder during deploy
- the same is required for PeerAdmin.card and admin.card, which have to be stored in appropriate sub-folders in ~/.composer

(3) Web Sockets

- the web socket implementation was brute force for Chapters 1-12. In IBM Cloud, this approach does not work, as we cannot specify ports in the app. –> Everything in the app must run through a single port.
- This requires that the port creation be done in index.js while setting up http server and then shared with other modules in app.
  - app.locals provides that sharing ability
- Because multiple browsers can now attach to the app, must manage multiple client connect requests and keep them active until device departs.
- host_address for web socket connection from browser is no longer 'localhost:xxx', it is the URL for the bluemix app and needs to be dynamically acquired once the web app starts.

(4) IBM Cloud application start-up

- Up until this point, any application we have loaded into bluemix has been started with a simple 'node index' command. –> When an application starts in IBM Cloud which will interact with Hyperledger and Composer, we need to set up two folders in the IBM Cloud for our app before it can run. Those folders are:
- **~/.hfc-key-store** This folder will store the credential information necessary to gather blockchain events
- **~/.composer/cards** This folder initially has the PeerAdmin and admin cards
- **~/.composer/client-data** This folder initially has the credentials for the PeerAdmin and admin cards

(5) Server Code Changes

- **index.js** The web socket creation and management code is removed from Z2B_Services and implemented in index.js, as it must share the httpserver service. It is also necessary to implement better socket management, since this will now run with access for multiple concurrent browsers. Because we are managing communications with multiple browsers, we need a way to send messages to all of them, which is done through the clients array. We also need to remove clients from that array when the browser session ends. This is done through the on-close process using information provided to the application in the client browser url & port. Because we want to use the message sending ability from any module in our application, we extend the support to use app.locals to share the processMessages function.

```
let server = http.createServer();
let clients = [];
app.locals.index=-1;
```

(continues on next page)

(continued from previous page)

```
/**
* WebSocket server
*/
app.locals.wsServer = new ws({httpServer: server});
app.locals.wsServer.on('request', function(request)
{
    console.log((new Date()) + ' Connection from origin '+ request.origin + '.');
    app.locals.connection = request.accept(null, request.origin);
    // we need to know client index to remove them on 'close' event
    app.locals.index = clients.push(app.locals.connection) - 1;
    console.log('app.locals.index: ', app.locals.index);
    console.log((new Date()) + ' Connection accepted.');
    app.locals.connection.on('message', function(message)
    {
        console.log((new Date()) + ' Received Message: ' + message.utf8Data);
        let obj =
            {
                time: (new Date()).getTime(),
                text: message.utf8Data,
                author: app.locals.connection.socket._peername.address+':'+app.
↪locals.connection.socket._peername.port,
                color: 'blue'
            };
        // broadcast message to all connected clients
        let json = JSON.stringify({ type:'message', data: obj });
        app.locals.processMessages(json);
    });

    // user disconnected
    app.locals.connection.on('close', function(_conn) {
        console.log((new Date()) + ' Peer '+ app.locals.connection.socket._
↪peername.address+':'+app.locals.connection.socket._peername.port+' disconnected␣
↪with reason code: "'+_conn+'".');
        // remove user from the list of connected clients
        for (let each in clients)
            {(function(_idx, _arr)
                {   console.log('['+_idx+'] BEFORE has id: '+_arr[_idx].socket._
↪peername.address+':'+_arr[_idx].socket._peername.port);
                if ((_arr[_idx].socket._peername.address === app.locals.
↪connection.socket._peername.address) && (_arr[_idx].socket._peername.port ===␣
↪app.locals.connection.socket._peername.port))
                    {
                    console.log('Match found!');
                    clients.splice(_idx, 1);
                    let obj =
                        {
                            time: (new Date()).getTime(),
                            text: ' I have left the meeting',
                            author: app.locals.connection.socket._peername.
↪address+':'+app.locals.connection.socket._peername.port,
                            color: 'red'
                        };
                    let json = JSON.stringify({ type:'message', data: obj });
                    app.locals.processMessages(json);
                }
            })(each, clients);}
        for (let each in clients)
```

(continues on next page)

(continued from previous page)

```
        {(function(_idx, _arr)
        {
            console.log('['+_idx+'] AFTER has id: '+_arr[_idx].socket._peername.
↪address+':'+_arr[_idx].socket._peername.port);
        })(each, clients);}
    });
});

/**
* callable function to send messages over web socket
* @param {JSON} _jsonMsg - json formatted content to be sent as message data
*/
function processMessages (_jsonMsg)
{
    for (let i=0; i < clients.length; i++) {clients[i].send(JSON.stringify(_
↪jsonMsg));}
}
app.locals.processMessages = processMessages;
```

- **Z2B_admin** Administrative services are disabled so that the network is not accidentally taken off line.
- **Z2B_Services** Routines for creating and managing sockets are removed completely and replaced with a single, very short, function:

```
/**
* New code to support sending messages to socket clients
* @param {Object} _locals - shared variables and functions from index.js
* @param {String} type - type of event message to put on channel
* @param {Event} event - event message
*/
    send: function (_locals, type, event)
    {
        _locals.processMessages({'type': type, 'data': event} );
    }
};
```

- **autoLoad.js** During member creation, profile information needs to be loaded so that the member is able to connect with the correct network. This data is now loaded from a new `connections.json` file which is generated during the new kubernetes-deploy.sh script execution. You'll see this in line 126:

```
let tempCard = new hlc_idCard(_meta, admin_connection);
```

- **queryBlockchain.js** getChainInfo is updated to pull the address of your kubernetes cluster and use that during the connection process.

```
  /**
  * get chain info
  * @param {express.req} req - the inbound request object from the client
  * @param {express.res} res - the outbound response object for communicating back␣
↪to client
  * @param {express.next} next - an express service to enable post processing prior␣
↪to responding to the client
  * @function
  */
  exports.getChainInfo = function(req, res, next)
  {
      let channel = {};
```

(continues on next page)

(continued from previous page)

```
      let client = null;
      let wallet_path = path.join(__dirname, 'creds');
      Promise.resolve().then(() => {
          //
          // As of 9/28/2017 there is a known and unresolved bug in HyperLedger␣
↪Fabric
          // https://github.com/hyperledger/composer/issues/957
          // this requires that the file location for the wallet for Fabric version␣
↪1.0 be in the following location:
          // {HOME}/.hfc-key-store
          // therefore the wallet location will be ignored and any private keys␣
↪required to enroll a user in this process
          // must be located in {HOME}/.hfc-key-store
          // this is currently managed for you in the installation exec by copying␣
↪the private key for PeerAdmin to this location
          //
          client = new hfc();
          return hfc.newDefaultKeyValueStore({ path: wallet_path })
          .then((wallet) => {
              client.setStateStore(wallet);
              // change PeerAdmin in following line to adminID
              return client.getUserContext(config.composer.PeerAdmin, true);})
              .then((user) => {
                  if (user === null || user === undefined || user.isEnrolled() ===␣
↪false)
                  { console.error('User not defined, or not enrolled - error');}
                  channel = client.newChannel(hlf1_profile.channel);
                  channel.addPeer(client.newPeer(hlf1_profile.peers[0].requestURL));
                  channel.addOrderer(client.newOrderer(hlf1_profile.orderers[0].
↪url));
              })
                  .then(() => {
                      return channel.queryInfo()
                      .then((blockchainInfo) => {
                          if (blockchainInfo) {
                              res.send({'result': 'success', 'currentHash':␣
↪blockchainInfo.currentBlockHash.toString('hex'), blockchain: blockchainInfo});
                          } else {
                              console.log('response_payload is null');
                              res.send({'result': 'uncertain', 'message': 'response_
↪payload is null'});
                          }
                      })
                      .catch((_err) => {
                          console.log('queryInfo failed with _err = ', _err);
                          res.send({'result': 'failed', 'message': _err.message});
                      });
                  });
      });
  };


- **hlcClient**
  function calls which previously included the web socket to be used now reference␣
↪``req.app.locals``, which is how we get to the processMessages function.
```

Browser Code

- **z2b_events** Earlier chapters in the tutorial had a web socket connection being created for each of the 6 roles (admin + 5 participant types). In this cloud-compatible version, we only create a single socket and then parse out the content to each participant based on the inbound message type, of which there are three (Message, Alert, BlockChain):

```
      /**
      * connect to web socket
      */
      function wsConnect()
      {    if (!window.WebSocket) {console.log('this browser does not support web␣
↪sockets');}
           let content = $('#body');
           let blockchain = $('#blockchain');
           wsSocket = new WebSocket('ws://'+host_address);
           wsSocket.onerror = function (error) {console.log('WebSocket error on␣
↪wsSocket: ' + error);};
           wsSocket.onopen = function ()
           {console.log ('connect.onOpen initiated to: '+host_address); wsSocket.send(
↪'connected to client');};
           wsSocket.onmessage = function (message)
           {
               let incoming
               incoming = message.data;
               while (incoming instanceof Object === false){incoming = JSON.
↪parse(incoming);}
               switch (incoming.type)
               {
               case 'Message':
                   content.append(formatMessage(incoming.data));
                   break;
               case 'Alert':
                   let event = JSON.parse(incoming.data);
                   addNotification(event.type, event.ID, event.orderID);
                   break;
               case 'BlockChain':
                   _blctr ++;
                   if (incoming.data !== 'connected')
                   {
                       $(blockchain).append('<span class="block">block '+incoming.
↪data.header.number+'<br/>Hash: '+incoming.data.header.data_hash+'</span>');
                       if (_blctr > 4) {let leftPos = $(blockchain).scrollLeft();
↪$(blockchain).animate({scrollLeft: leftPos + 300}, 250);}
                   }
                   break;
               default:
                   console.log('Can Not Process message type: ',incoming.type);
               }
           };
      }

- **z2b_buyer (and other participants)**

  - all 'port' information is removed from the code as it is no longer relevant
  - calls to ``wsDisplay`` are removed as that service is now replaced by the␣
↪``wscConnect`` function displayed above.
```

Setup

- Use the ./install-bx-cli.sh exec to perform the 2 following steps:
  - Install kubectl on your local device:
    - OSX:
    - Linux:
  - Install the Bluemix CLI and cluster support on your local system.
- **`buildAndDeploy` replaced by `kuebernetes-deploy.sh`** kubernetes-deploy uses set up information from the [ibm-container-service](https://github.com/IBM-Blockchain/ibm-container-service) git repo. This chapter includes the scripts from the cs-offerings folder in that repo. There are a number of functions which are used in the kubernetes-deploy.sh script built for the Zero To Blockchain tutorial:
  - `./createArchive.sh` -n $NETWORK_NAME (this is the same version as before)
  - `getContext` (discovers your kube cluster ip address and sets the environment info)
  - `clearOldCards` (gets rid of now obsolete identity cards)
  - `setupCluster` (sets up the kube cluster - this is the code from ibm-containers-service)
  - `pauseForCard` (wait for you to access the automatically-loaded playground, launch it and then make a local copy of your PeerAdmin card)
  - `updateCard` (update the PeerAdmin card with the correct ip address)
  - `./getPEM.sh` (credential management)
  - `installNetwork` (using the PeerAdmin card, install your network into your new kube cluster)
- start-up in package.json (node index) replaced with (./init.sh)

1.1.14 Chapter 14: Debugging Hyperledger Composer inside Docker Containers

Introduction

This is a high level tutorial on how HyperLedger Composer uses Docker. For a thorough introduction to Docker, I recommend [The Docker Book](https://www.dockerbook.com/).

HyperLedger Composer v0.16 (the current version of this tutorial), creates 4 docker containers to run the local dev environment, one each for your network, the orderer, the peer, and one for couchdb (the noSQL database). This happens each time you run the `./buildAndDeploy` script, which calls `deployNetwork.sh`. During the execution of the deployNetwork script, the following information is displayed in your terminal window:

```
ARCH=$ARCH docker-compose -f "${DIR}"/composer/docker-compose.yml down
Stopping peer0.org1.example.com ... done
Stopping ca.org1.example.com    ... done
Stopping orderer.example.com    ... done
Stopping couchdb                ... done
Removing peer0.org1.example.com ... done
Removing ca.org1.example.com    ... done
Removing orderer.example.com    ... done
Removing couchdb                ... done
Removing network composer_default
ARCH=$ARCH docker-compose -f "${DIR}"/composer/docker-compose.yml up -d
Creating orderer.example.com ... done
Creating peer0.org1.example.com ... done
```

(continues on next page)

(continued from previous page)

```
Creating couchdb ...
Creating ca.org1.example.com ...
Creating peer0.org1.example.com ...
```

Three things are happening here:

1. If the network is up, each of the 4 Docker containers running are stopped.
2. Then each container is removed from your computer
3. Then each container is restarted.

All of this is controlled by the docker-compose.yml file, which is located in the fabric-tools folder installed during the set up process. Specifically, the file is in the following path: `fabric-tools/fabric-scripts/hlfv1/composer`. The purpoe of the docker compose yaml file is to capture all of the commands and parameters required to start up a group of containers in a readable and consistent manner. While everything in the yaml file could be executed directly from the command line, the commands become very unwieldy, so Docker Compose was created to allow all of the parameters to be placed into a yaml file and allow all of the container starts to be in the same yaml file. what containers are created? .....

You can discover this information by executing the following command: `docker ps -a`, which displays the following for our network:

```
CONTAINER ID        IMAGE
↪COMMAND                  CREATED             STATUS              PORTS
↪                         NAMES
0386e33b66c5        dev-peer0.org1.example.com-zerotoblockchain-network-0.16.0-...
↪"chaincode -peer.add..."   17 minutes ago      Up 18 minutes
↪                           dev-peer0.org1.example.com-zerotoblockchain-network-
↪0.16.0
fd0187712cfc        hyperledger/fabric-peer:x86_64-1.0.4
↪"peer node start --p..."   18 minutes ago      Up 18 minutes       0.0.0.0:7051->
↪7051/tcp, 0.0.0.0:7053->7053/tcp   peer0.org1.example.com
c6f532764faa        hyperledger/fabric-couchdb:x86_64-1.0.4
↪"tini -- /docker-ent..."   18 minutes ago      Up 18 minutes       4369/tcp, 9100/
↪tcp, 0.0.0.0:5984->5984/tcp       couchdb
cf3608360871        hyperledger/fabric-orderer:x86_64-1.0.4
↪"orderer"                18 minutes ago      Up 18 minutes       0.0.0.0:7050->7050/
↪tcp                      orderer.example.com
346f77c65284        hyperledger/fabric-ca:x86_64-1.0.4
↪"sh -c 'fabric-ca-se..."   18 minutes ago      Up 18 minutes       0.0.0.0:7054->
↪7054/tcp                 ca.org1.example.com
```

Contents of the docker-compose.yml file

```
yml
version: '2'

services:
```

The file starts out simply.

- version refers to the version of docker-compose being used, in this case it's version 2. This specifies both what can be put in the yaml file and how each term is to be grouped and specified.
- services refers to the set of docker containers which will be started by this yaml file.

Each service (container) has a similar structure and we will look at all four: ca - Certificate Authority ..

- ```
  ca.org1.example.com:
      image: hyperledger/fabric-ca:$ARCH-1.0.4
      environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca.org1.example.com

      ports:
      - "7054:7054"
      command: sh -c 'fabric-ca-server start --ca.certfile /etc/
  ↪hyperledger/fabric-ca-server-config/ca.org1.example.com-cert.pem --ca.
  ↪keyfile /etc/hyperledger/fabric-ca-server-config/
  ↪19ab65abbb04807dad12e4c0a9aaa6649e70868e3abd0217a322d89e47e1a6ae_sk -b␣
  ↪admin:adminpw -d'
      volumes:
      - ./crypto-config/peerOrganizations/org1.example.com/ca/:/etc/
  ↪hyperledger/fabric-ca-server-config
      container_name: ca.org1.example.com
  ```

  - image refers to the docker container which is to be started. In this case, it's owned by 'hyperledger' and is shown here as the 'fabric-ca:$ARCH-1.0.4' image where $ARCH is the architecture of your local operating system, which is retrieved and set in the startFabric.sh script. On my OSX laptop ARCH is set to x86_64, so the image to be retrieved would be: fabric-ca:x86_64-1.0.4 where fabric-ca is the image name and x86-64-1.0.4 is the version identifier.
  - ENVIRONMENT sets environment variables inside the container. These variables are not visible to applications running on your host operating system. - The '#' sign makes everything that follows it a comment
  - PORTS: identifies a port inside the container (left number) and links it to a port visible to your host operating system.
  - COMMAND lists the command to be executed when the container starts. In this case, the shell is called to execute fabric-ca-server start command with 3 options
  - VOLUMES links one or more specific folders on your local computer to a specific path inside the container. The './' prefix means 'start here', where 'here' is defined by the location of the docker-compose.yml file (shown earlier). This allows local information to be shared in a persistent manner with a docker container and, as needed, changed without having to rebuild the container.
  - CONTAINER_NAME is the name that the container will display when docker ps is used.

  You will observe an identical structure in each of the following services:

Create the orderer container

```
orderer.example.com:
 container_name: orderer.example.com
 image: hyperledger/fabric-orderer:$ARCH-1.0.4
 environment:
   - ORDERER_GENERAL_LOGLEVEL=critical
   - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
   - ORDERER_GENERAL_GENESISMETHOD=file
   - ORDERER_GENERAL_GENESISFILE=/etc/hyperledger/configtx/composer-genesis.block
   - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
   - ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp
```

(continues on next page)
```

(continued from previous page)

```
 working_dir: /opt/gopath/src/github.com/hyperledger/fabric
 command: orderer
 ports:
   - 7050:7050
 volumes:
     - ./:/etc/hyperledger/configtx
     - ./crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/
↪msp:/etc/hyperledger/msp/orderer/msp
```

Create the peer0 container

```
peer0.org1.example.com:
  container_name: peer0.org1.example.com
  image: hyperledger/fabric-peer:$ARCH-1.0.4
  environment:
    - CORE_LOGGING_PEER=critical
    - CORE_CHAINCODE_LOGGING_LEVEL=critical
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_PEER_ID=peer0.org1.example.com
    - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
    - CORE_PEER_LOCALMSPID=Org1MSP
    - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
    - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
    - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb:5984
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric
  command: peer node start --peer-defaultchain=false
  ports:
    - 7051:7051
    - 7053:7053
  volumes:
      - /var/run/:/host/var/run/
      - ./:/etc/hyperledger/configtx
      - ./crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.
↪com/msp:/etc/hyperledger/peer/msp
      - ./crypto-config/peerOrganizations/org1.example.com/users:/etc/hyperledger/msp/
↪users
  depends_on:
    - orderer.example.com
    - couchdb
```

Here you see an additional parameter:

- DEPENDS_ON which, as you might guess, lists docker container names which must be running before this container will successfully start.

Create the couchdb container:

```
couchdb:
 container_name: couchdb
 image: hyperledger/fabric-couchdb:$ARCH-1.0.4
 ports:
   - 5984:5984
```

(continues on next page)

(continued from previous page)

```
 environment:
   DB_URL: http://localhost:5984/member_db
```

OK, so now we have the four base containers for our network, but we don't yet have the network. These docker-compose commands are executed when the startFabric script runs:

```
ARCH=$ARCH docker-compose -f "${DIR}"/composer/docker-compose.yml down
ARCH=$ARCH docker-compose -f "${DIR}"/composer/docker-compose.yml up -d

# wait for Hyperledger Fabric to start
# incase of errors when running later commands, issue export FABRIC_START_TIMEOUT=
↪<larger number>
echo ${FABRIC_START_TIMEOUT}
sleep ${FABRIC_START_TIMEOUT}

# Create the channel
docker exec peer0.org1.example.com peer channel create -o orderer.example.com:7050 -c␣
↪composerchannel -f /etc/hyperledger/configtx/composer-channel.tx

# Join peer0.org1.example.com to the channel.
docker exec -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.
↪com/msp" peer0.org1.example.com peer channel join -b composerchannel.block
```

The script first takes down the network and then restarts it in detached (that's the -d) mode. The script then pauses for 15 seconds (timeout default) and then creates the peer0 channel (remember the depends_on) which connects peer0 to the orderer and records that connection in the couchdb database. We are missing a container. The one with our network and the first one in the docker ps -a list. That's created by the deployNetwork.sh script when the

```
composer network start -c PeerAdmin@hlfv1 -A admin -S adminpw -a $NETWORK_NAME.bna --
↪file networkadmin.card
```

script is executed.

So where does the oher network come from? The one with our business network? That is created when the deployNetwork.sh script is run at the end of buildAndDeploy. Each time you run the buildAndDeploy script, it stops, removes, and rebuilds all 5 containers. The first 4 are built based on the docker-compose.yml file, the last one, with your network, is created in the deploy script. You could do the same thing in the admin console by using the create and install network services. If you do that, please be sure to use a different network name.

OK, so, now we want to test and debug. The part of the docker environment where our code will run is in the 5th container - the one named: **dev-peer0.org1.example.com-zerotoblockchain-network-0.16.0**

In much of the code we've written, we will often use console.log to display what's going on. This works in all of our nodeJS code EXCEPT for the code in the Chapterxx/network/lib folder. For some reason, those console.log messaegs never print out. Well, actually, they do.

We're going to start by going to any chapter between Chapter5 and Chapter 12 where you've completed your coding. We'll run the buildAndDeploy script to create a clean instance of the network and then we're going to start following the log for our business network. We'll need two terminal windows open at the same time. One will show the log output in real time from our business network container, the other will show the log output from our nodejs application. The examples in this script were created using Chapter 6.

1. go to `ZeroToBlockChain\Chaper06`
2. run `./buildAndDeploy`
3. run `docker ps -a` which will return something like the following:

```
CONTAINER ID        IMAGE                                                                ␣
↪               COMMAND                  CREATED              STATUS              ␣
↪             PORTS                                                                  NAMES
0386e33b66c5        dev-peer0.org1.example.com-zerotoblockchain-
↪network-0.16.0-... "chaincode -peer.add..."   17 minutes ago                        ␣
↪Up 18 minutes                                                                       ␣
↪  dev-peer0.org1.example.com-zerotoblockchain-network-0.16.0
fd0187712cfc        hyperledger/fabric-peer:x86_64-1.0.4                             ␣
↪               "peer node start --p..."   18 minutes ago      Up 18␣
↪minutes       0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp                        ␣
↪peer0.org1.example.com
c6f532764faa        hyperledger/fabric-couchdb:x86_64-1.0.4                          ␣
↪               "tini -- /docker-ent..."   18 minutes ago      Up 18␣
↪minutes       4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp                            ␣
↪couchdb
cf3608360871        hyperledger/fabric-orderer:x86_64-1.0.4                          ␣
↪               "orderer"                18 minutes ago      Up 18␣
↪minutes       0.0.0.0:7050->7050/tcp                                                ␣
↪orderer.example.com
346f77c65284        hyperledger/fabric-ca:x86_64-1.0.4                               ␣
↪               "sh -c 'fabric-ca-se..."   18 minutes ago      Up 18␣
↪minutes       0.0.0.0:7054->7054/tcp                                                ca.
↪org1.example.com
```

4. run `docker logs dev-peer0.org1.example.com-zerotoblockchain-network-0.16.0 -f`
5. open a new terminal window and go to `ZeroToBlockChain\Chaper06`
6. run `npm start`

```
> zerotoblockchain-network@0.1.6 start /Users/rddill/Documents/GitHub/Z2B_
↪Master/Chapter06
> node index

Listening locally on port 6002
```

7. go to your browser and connect to localhost:PORT where PORT is the number displayed after npm start - 6002 in this example
8. Select Admin –> Preload Network

- When you do this, you'll see messages starting to scroll in both windows. You'll see that everytime the ERROR message appears in the npm start window, a similar message appears in the docker logs window. OK, boring, but nice to see the consistency.

Add a Debug Message

Now let's add a console message to the sample.js file in `Chapter06/network/lib` - one we've tried before and never been ab

9. press CTRL-c in both terminal windows to stop both the logging process and the nodejs server
10. In your favorite editor, open the sample.js file and update the CreateOrder function by adding this line to it:

```
    console.log('Order for '+purchase.buyer+' created wih amount: '+purchase.amount);

- your CreateOrder function should now look like this:
```

```
function CreateOrder(purchase) {
    purchase.order.buyer = purchase.buyer;
    purchase.order.amount = purchase.amount;
    purchase.order.financeCo = purchase.financeCo;
    purchase.order.created = new Date().toISOString();
    purchase.order.status = JSON.stringify(orderStatus.Created);
    console.log('Order for '+purchase.buyer+' created wih amount: '+purchase.amount);
    return getAssetRegistry('org.acme.Z2BTestNetwork.Order')
        .then(function (assetRegistry) {
            return assetRegistry.update(purchase.order);
        });
}
```

11. Save your updated sample.js file
12. run `./buildAndDeploy`
13. when complete, run `docker logs dev-peer0.org1.example.com-zerotoblockchain-network-0.16.0 -f`
14. go to your 2nd terminal window and run `npm start`
15. go to your browser and load/reload the browser page for localhost:PORT

- please note, the port number changes sometimes

16. select Admin –> Preload Network and watch the logs.

- ***WOOHOO*** Our messages now show up in the log listing from docker:
  - Order for Resource {id=org.acme.Z2BTestNetwork.Buyer#eric@bornonthecloudinc.com} created wih amount: 6055
  - Order for Resource {id=org.acme.Z2BTestNetwork.Buyer#yes@softwaresolutionsinc.com} created wih amount: 9125
  - Order for Resource {id=org.acme.Z2BTestNetwork.Buyer#yes@softwaresolutionsinc.com} created wih amount: 38800
  - Order for Resource {id=org.acme.Z2BTestNetwork.Buyer#yes@softwaresolutionsinc.com} created wih amount: 6055
  - Order for Resource {id=org.acme.Z2BTestNetwork.Buyer#yes@softwaresolutionsinc.com} created wih amount: 9125
  - Order for Resource {id=org.acme.Z2BTestNetwork.Buyer#yes@softwaresolutionsinc.com} created wih amount: 6055
  - Order for Resource {id=org.acme.Z2BTestNetwork.Buyer#eric@bornonthecloudinc.com} created wih amount: 38800
  - Order for Resource {id=org.acme.Z2BTestNetwork.Buyer#eric@bornonthecloudinc.com} created wih amount: 9125
  - Order for Resource {id=org.acme.Z2BTestNetwork.Buyer#eric@bornonthecloudinc.com} created wih amount: 38800

Congratulations

You now have the ability to debug not only your code in your browser and in the nodejs server, you now, also, have the ability to trace and debug your chaincode running in the docker container!

CHAPTER 2

Indices and tables

- genindex
- modindex
- search