Development and Environment Guide

For IBM HyperLedger Fabric – Composer

Cognizant IPM Hyperledger Team

Contents

[**Document Versions and Reviews** 3](#_Toc480999773)

[**Purpose** 4](#_Toc480999774)

[**About Chaincode** 4](#_Toc480999775)

[**Prerequisites** 4](#_Toc480999776)

[**Technical skills identified (so far) needed to develop IBM Blockchains** 4](#_Toc480999777)

[**1.** **IBM Blockchain Component Architecture** 5](#_Toc480999778)

[***1.1.1*** *Diagram :***IBM Blockchain Fabric Basic Developer Component Architecture** 5](#_Toc480999779)

[***1.1.2*** *Diagram :***Cloud IBM Blockchain HL Fabric service and Golang chaincoding** 6](#_Toc480999780)

[***1.1.3***  ***Diagram :* IBM Blockchain Fabric Architecture** 7](#_Toc480999781)

[**2.** **Overview of Blockchain Environments Setup** 8](#_Toc480999782)

[**3.1 Step One: Setup the Fabric-Golang source code environment to your local Blockchain Development machine** 8](#_Toc480999783)

[**3.1.1** **Setting up Golang environment and build chain code** 8](#_Toc480999784)

[**3.1.2 Setup Summary for Blockchain Apps deployed on Bluemix Blockchain service** 8](#_Toc480999785)

[**3.1.3 Setup Summary for Blockchain Apps deployed on Local Machine Playground** 9](#_Toc480999786)

[**4. Step By Step Component Install-Config Walkthrough** 10](#_Toc480999787)

[**4.1 Setup/Build Your Chaincode Development Environment** 10](#_Toc480999788)

[**4.1.1** **Set up your chaincode development local pipeline** 14](#_Toc480999789)

[**4.1.2** **Implement the chaincode interface APIs** 17](#_Toc480999790)

[**4.1.3** **Interacting with your chaincode** 23](#_Toc480999791)

[**4.1.4** **Deploying your chaincode** 25](#_Toc480999792)

[**4.2** **HL Fabric Composer – Playground - CLI Development** 37](#_Toc480999793)

[**4.2.1 Set-up FabricComposer Playground on my machine using Docker ToolBox** 37](#_Toc480999794)

[**4.2.2** **Fabric Composer CLI setup** 41](#_Toc480999795)

[**4.3 Docker Images for IBM HL Fabric Networks** 44](#_Toc480999796)

[**Appendix** 45](#_Toc480999797)

[**Coding guidelines** 45](#_Toc480999798)

[**References** 45](#_Toc480999799)

[**Follow on Todos** 45](#_Toc480999800)

# **Document Versions and Reviews**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author(s) | Reviewed By | Notes |
| 0.1 | 2-26-17 | R Lee Cook,  Madhu Reddy |  | Initial draft |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# **Purpose**

This document is a getting started guide specifying the component architecture and detailed setup and good practices for two IBM Blockchain Hyperledger Fabric development environments : Bluemix Cloud Fabric Golang and local machine Fabric Composer playground.

The current version of IBM HL Fabric used is V0.6.

### **About Chaincode**

is software defining an asset or assets, and the transaction instructions for modifying the asset(s). In other words, it’s the business logic. Chaincode enforces the rules for reading or altering key value pairs or other state database information. Chaincode functions execute against the ledger current state database and are initiated through a transaction proposal. Chaincode execution results in a set of key value writes (write set) that can be submitted to the network and applied to the ledger on all peers.

Each ledger transaction results in a set of asset key-value pairs that are committed to the ledger as creates, updates, or deletes.

Chaincode gets installed only on peers that need to access the asset states to perform reads and writes (in other words, if a chaincode is not installed on a peer, it will not be able to properly interface with the ledger). To further obfuscate the data, values within chaincode can be encrypted (in part or in total) using common cryptographic algorithms such as SHA0-256, etc. before appending to the ledger.

The blockchain is a distributed system consisting of many nodes that communicate with each other. The blockchain runs programs called chaincode, holds state and ledger data, and executes transactions. The chaincode is the central element as transactions are operations invoked on the chaincode. Transactions have to be “endorsed” and only endorsed transactions may be committed and have an effect on the state. There may exist one or more special chaincodes for management functions and parameters, collectively called system chaincodes.

# **Prerequisites**

Please see : (<https://developer.ibm.com/courses/all-courses/blockchain-for-developers/>) to access the IBM eduction Blockchain Hyperledger basic developer (3 part) courses below which provide a technical introduction to Blockchain so that it is easy understand all the config and setup steps in this cook book.

**Part 1.** Apply blockchain to a business network

**Part 2.** Explore blockchain and the Hyperledger Fabric project

**Part 3.** Build your first chaincode

\* Additional online devworks articles, github documents will be listed as needed in this guide.

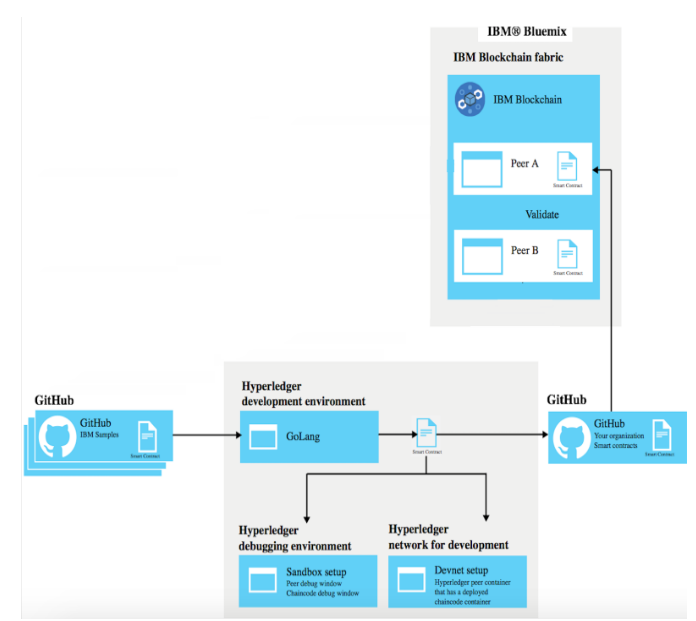
\* Also see the list of [References](#_References) at the end of this document.

## **Technical skills identified (so far) needed to develop IBM Blockchains**

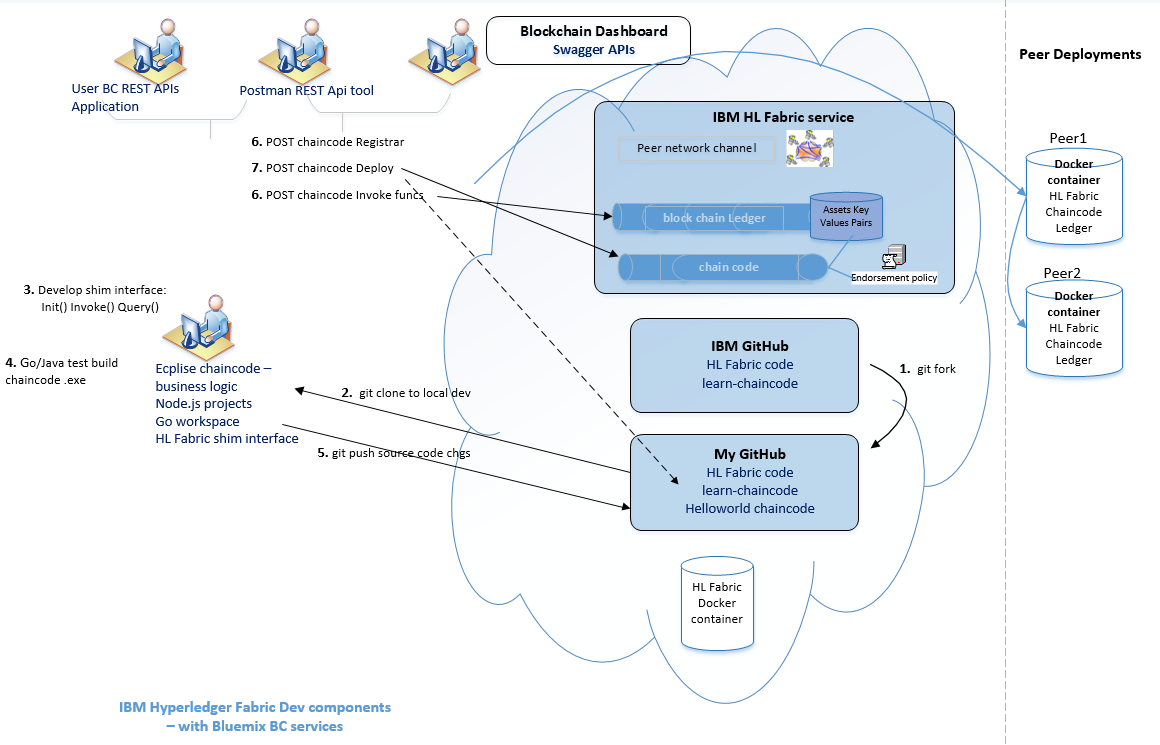
1. IBM Bluemix certification –or- heavy hands on usage
2. IBM Hyperledger Fabric architecture and components – IBM BC devoworks course is a good intro
3. Hyperledger Fabric chaincode interfaces
4. Node.js 0.12.0+ and npm v2+
5. Go (Google lang) dev programming & environment (required to build chaincode) –
6. Java support is in Beta now.. we need to look at what is the best language to write Chain code to achieve better performance.
7. Github and Bmix DevOps
8. Swagger APIs
9. Postman REST Api test tool – chrome extension
10. IBM Fabric Composer
11. IBM HL Blockchain fabric V1.0 needs to be explored next.

# **IBM Blockchain Component Architecture**

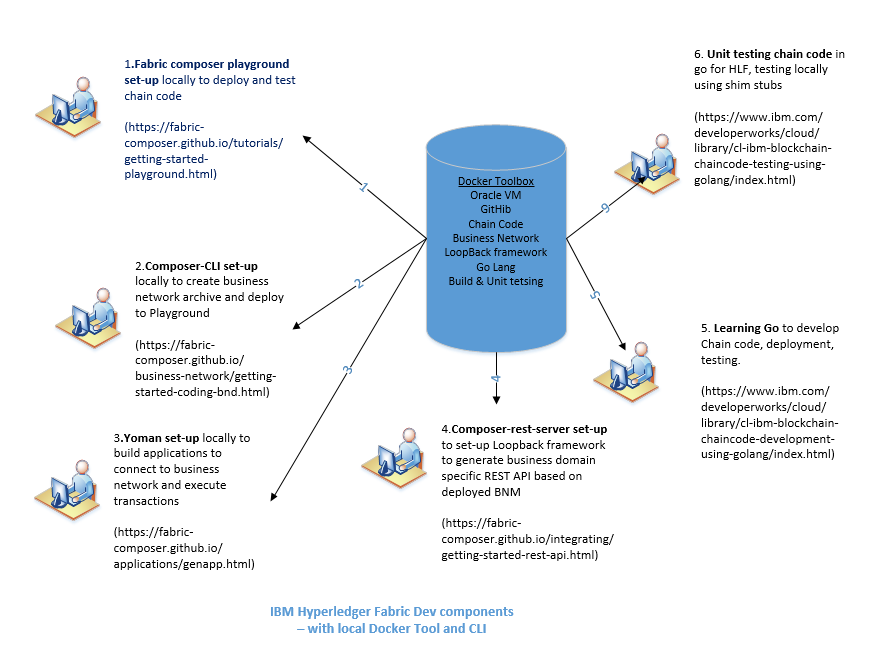
## *Diagram :***IBM Blockchain Fabric Basic Developer Component Architecture**



## *Diagram :***Cloud IBM Blockchain HL Fabric service and Golang chaincoding**



## ***1.1.3 Diagram :* IBM Blockchain Fabric Architecture**



# **Overview of Blockchain Environments Setup**

* Some of the config steps may vary slightly depending on your local machine’s installation packages – ie having Ecplise for NodeJS etc.
* Your initial development setup path ends with working chaincode that creates generic assets for exchange on a blockchain network.

## **3.1 Step One: Setup the Fabric-Golang source code environment to your local Blockchain Development machine**

The following is the common setup for both the IBM Bluemix Cloud Fabric and Composer playground environments

### **3.1.1 Setting up Golang environment and build chain code**

Please follow details set-ups mentioned in **“Build your chain code**” course in pre-requisite section. Below are the key common steps involved in setting up a developer environment on a new machine for development and test chain code in block chain business network.

1. Follow: [Configuring your Windows development environment](https://github.com/Microsoft/nodejs-guidelines/blob/master/windows-environment.md#compiling-native-addon-modules)
2. Setting up a go lang- fabric-github blockhain dev env for coding IBM Hyperledger Fabirc shim interface methods to build a basic Golang app.
3. Use a <common CTS –or- private> github as source code repos and chaincode deployment source
4. Create your local dev pipeline
5. Us Bluemix BC Swapper REST APIs –or- POSTMAN browser plugin to Enroll a peer userid, Deploy and Init your chaincode with simple “asset” key- value pairs, and altering the blockchain asset values by calling the chaincode invoke()/write() functions.
6. Use POSTMAN chrome plugin for REST API testing

Then you’ll focus on successful set up for both chaincode dev environments.

* IBM HL Fabric V0.6 with a local fabric-Golang dev env deployed on Bluemix Blockchain service
* IBM HL Fabric Composer using a docker ‘playground’, a CLI interface deployed local Windows machine

### **3.1.2 Setup Summary for Blockchain Apps deployed on Bluemix Blockchain service**

1. Do common Fabric-Golang setup steps above.
2. Register with Cognizant’s public IBM Bluemix server – raise OneNSS request.
3. Get access to IPM IBM BC github repository.

* Helloworld" Blockchain chaincode working sample app. Its deployed on this route (for now) :

public Bmix - Org: COG-EAS-IPM-IBM Space: KG-EAS-IPM-GDC-IBM serviceName: “Blockchain-lee-helloworld”

### **3.1.3 Setup Summary for Blockchain Apps deployed on Local Machine Playground**

1. Do common Fabric-Golang setup steps above.
2. Install GitBash
3. Install Docker toolbox - contains Oracle VirtualBox
4. Install fabriccomposer/composer-playground in your local docker instance.
5. Setup Fabric Composer CLI command line
6. Setup Yoman
7. Setup Composer-rest-server

# **4. Step By Step Component Install-Config Walkthrough**

## **4.1 Setup/Build Your Chaincode Development Environment**

See source & steps : <https://github.com/IBM-Blockchain/learn-chaincode> - Do **Setup steps once**

List of dependencies and recommended tools that you should install in order to develop chaincode

**1- Install Git Client**

* Its installed with Eclipse NodeJs plugin

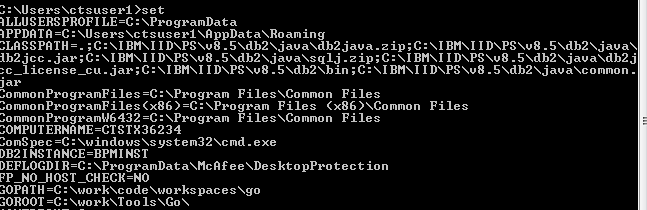
**2- Install Go on windows**

 GOPATH environment variable - When you run go build to verify that your chaincode compiles, Go looks in the $GOPATH/src directory for any non-standard dependencies that you list in the import block in your chaincode. The [Go installation instructions](https://golang.org/doc/install) guide you through GOPATH environment variable setup

* The GOPATH environment variable specifies the location of your workspace for chain code projects.

set GOPATH=C:\work\code\workspaces\go - for temp cmdline

* Define a Windows Env var for perm – Plus Git enabled



**3- Install IBM Hyperledger Fabric on my local windows %GOPATH%**

* To enable read and write functions on the ledger, your chaincode App project must import the chaincode shim from Hyperledger Fabric.
* To compile BC chaincode locally, you must have the Hyperledger Fabric code location present in your GOPATH environment variable.

Your chaincode version must align with the Hyperledger Fabric version that you will deploy your chaincode to. For example, the network depicted in Figure 1 requires cloning the Hyperledger Fabric v0.6-preview codebase.

The fabric codebase, for either version, must be stored in :

your $GOPATH/src/hyperledger/fabric path - mkdir %GOPATH%/“src”

mkdir -p $GOPATH/src/github.com/hyperledger

cd $GOPATH/src/github.com/hyperledger - C:\work\code\workspaces\go\src\

To install the Hyperledger Fabric v0.6 codebase & shim, use the following git clone command:

~~git clone github.com/hyperledger/fabric/core/chaincode/shim -> works~~

# The v0.6 release exists as a branch inside the Gerrit fabric repository

git clone –b v0.6 [**http://gerrit.hyperledger.org/r/fabric**](http://gerrit.hyperledger.org/r/fabric)

~~-> fatal: repository 'github.com/hyperledger/fabric/core/chaincode/shim' does not e~~

~~xist~~

~~GOPATH=C:\work\code\workspaces\go - works too~~

~~C:\Users\ctsuser1>git clone -b v0.6 http://gerrit.hyperledger.org/r/fabric~~

~~Cloning into 'fabric'...~~

~~remote: Counting objects: 3861, done~~

~~remote: Finding sources: 100% (7/7)~~

~~remote: Total 29013 (delta 0), reused 29011 (delta 0)~~

~~Receiving objects: 100% (29013/29013), 49.50 MiB | 2.44 MiB/s, done.~~

~~Resolving deltas: 100% (13163/13163), done.~~

~~Checking connectivity... done.~~

~~Checking out files: 100% (2285/2285), done.~~

~~But the clone installed the files in > C:\Users\ctsuser1\fabric~~

If the fabric is not properly installed in your GOPATH, building your chaincode will return an error similar to the following example:

$ go build .

chaincode\_example02.go:27:2: cannot find package "github.com/hyperledger/fabric/core/chaincode/shim" in any of:

C:\Go\src\github.com\hyperledger\fabric\core\chaincode\shim (from $GOROOT)

C:\gopath\src\github.com\hyperledger\fabric\core\chainco

**4- Install Postman to chrome browser**

Version 2.0.14 of Postman has basic support for the Swagger 2.0 format. Some aspects (like refs to remote schemas) are not supported, but the basic structure of any single schema should be import-able to a large extent.

Good for adding REST API body text – complex APIs

See: <http://www.tjmaher.com/2016/07/introduction-to-api-testing-with.html>

See: <https://developer.ibm.com/recipes/tutorials/hello-world-using-ibm-blockchain-on-bluemix-part-1/>

* [Home page](https://www.getpostman.com/)

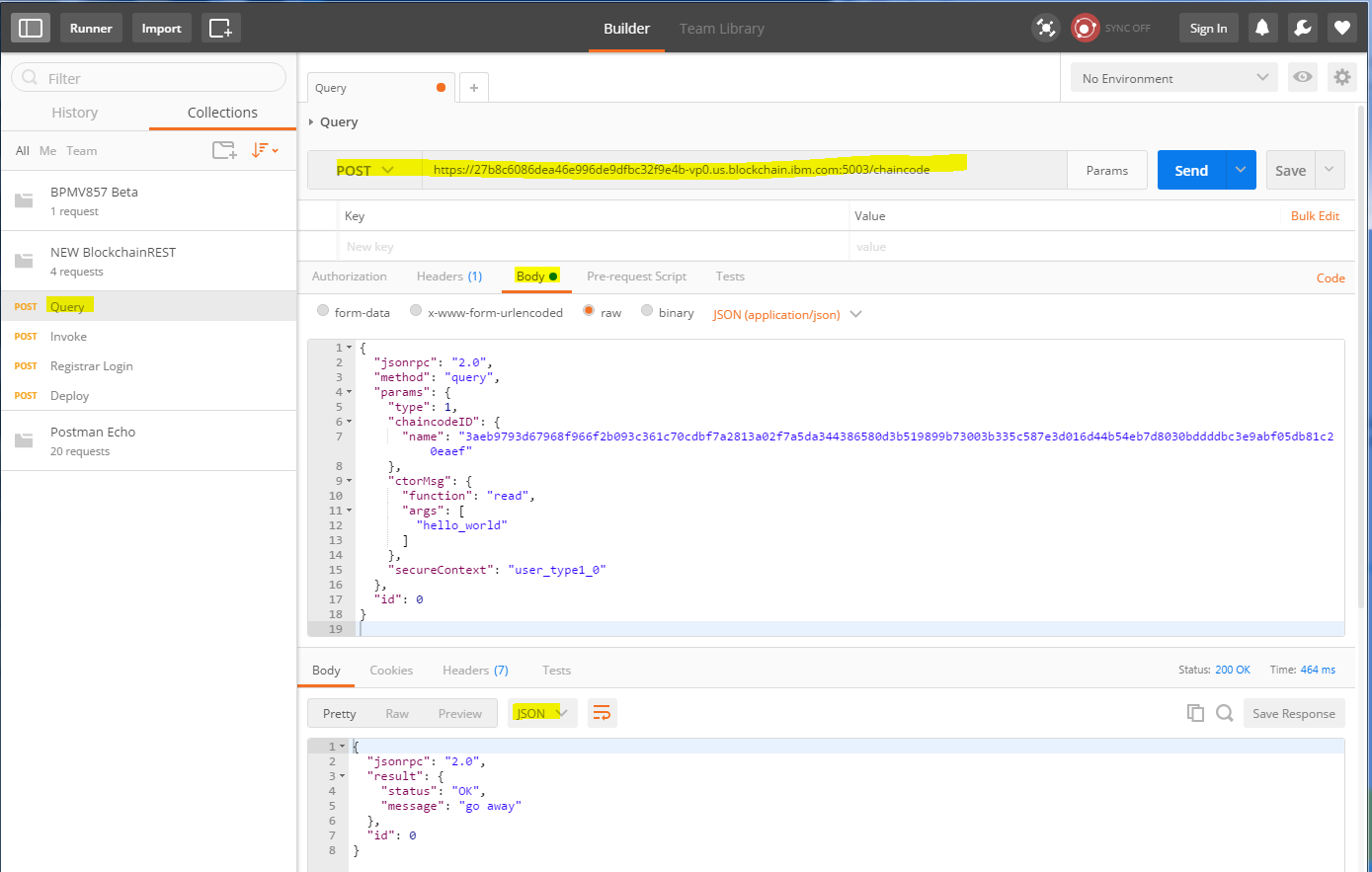
Postman is a REST API testing tool. Though it is deprecated (by Swagger) , we still use the REST API in the fabric for this tutorial because it allows you to deploy and test your chaincode without needing to use the fabric SDK. You'll learn more about the fabric SDK in our other examples.

**Instructions**

Download the [Postman tool](https://www.getpostman.com/). Depending on your operating system, you may also need to install Chrome to use Postman. Once you have the tool running, import the [request collection](https://github.com/IBM-Blockchain/learn-chaincode/blob/master/LearnChaincodeREST.postman_collection.json) included in this repository. This collection contains requests for enrolling a user on a peer, as well as deploying, invoking, and querying chaincode. The collection repository contains all the REST calls need to complete this tutorial.

In chrome issue url - chrome://apps/ > Import > C:\work\code\workspaces\go\src\github.com\princeoftides\learn-chaincode\ LearnChaincodeREST.postman\_collection.json

Click Body to insert same json text as for Swagger APIs:



1. **Install Node.js**

Comes with Eclipse Node.js plugin

Node.js is NOT necessary to develop chaincode, but most of our demos are built on Node.js, so it might be handy to go ahead and install it now. Also, you'll need it when you start using the fabric SDK.

**Instructions**

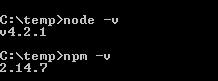
Download the appropriate installation package and make sure the following commands work on your machine:

$ node -v

v4.4.7

$ npm -v

3.10.5



**Todo: update my Eclipse Node.js**

1. **IDE**

~~Visual Studio Code Atom~~

Eclipse… / windows cmdline

## **4.1.1 Set up your chaincode development local pipeline**

See source & steps : <https://github.com/IBM-Blockchain/learn-chaincode>

* Use the following tasks take you through the process of building a pipeline that will allow you to build chaincode effectively : for writing, building and testing your chaincode.
* your pipeline for iterating on chaincode will consist of the following steps:

Write / Make changes to the given chaincode on your local machine

Verify the code compiles.

Push/upload your updates to My GitHub.

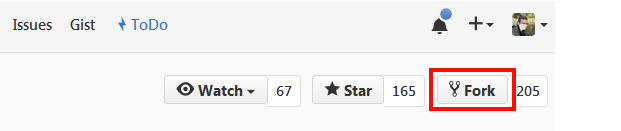
Deploy your updated chaincode to your Bluemix network BC service using the fabric REST API.

Test your chaincode using the fabric REST API.

Repeat.

? Why not using Bmix DevOps ?

* 1- **Fork** from IBM github learnchaincode to my github
* **IBM Blockchain service on Bluemix** requires chaincode to be in a (My) GitHub repository for deployment through the REST API.
* Click on “Fork” the appropriate version of the learn chaincode repository for your network version to your GitHub account. Fork v2.0 for a v0.6 fabric network. One option is to use the Fork button, located at the top right of the repository page. Forking copies the entire repository to your local machine, including all branches, which are shown by clicking the Branch: button at the upper left of the page



Forking" the repository means creating a copy of this repository under your GitHub account. Note that the fork will fork the entire repository including all the branches. Toggle the Branch button on the left to see the available branches.

2- **Clone your fork** from my github into my local windows $GOPATH/src

Bmix DevOps [URL:https://hub.jazz.net/project/leegitrepos/**learn-chaincode**](URL:https://hub.jazz.net/project/leegitrepos/learn-chaincode)

<https://hub.jazz.net/code/edit/edit.html#/code/file/leegitrepos-OrionContent/leegitrepos%2520%257C%2520learn-chaincode/>

Git [URL: https://github.com/princeoftides/learn-chaincode.git](URL:%20%20%20https://github.com/princeoftides/learn-chaincode.git)

To fork using the CLI, enter the following commands into your Git Bash shell:

cd %GOPATH%

mkdir -p src/github.com/<YOUR\_GITHUB\_ID\_HERE>/

cd src/github.com/<YOUR\_GITHUB\_ID\_HERE>/

git clone -b v2.0 https://github.com/<YOUR\_GITHUB\_ID\_HERE>/learn-chaincode.git

Cloning into 'learn-chaincode'...

remote: Counting objects: 317, done.

Receiving objects: 99% (314/317) (delta 0), pack-reused 317Receiving objec

Receiving objects: 100% (317/317), 412.16 KiB | 0 bytes/s, done.

Resolving deltas: 100% (135/135), done.

Checking connectivity... done.

C:\work\code\workspaces\go>dir

Volume in drive C is Windows

Volume Serial Number is 641C-CC5B

**Directory of C:\work\code\workspaces\go**

03/20/2017 02:59 PM <DIR> .

03/20/2017 02:59 PM <DIR> ..

03/20/2017 02:42 PM <DIR> hyperledger

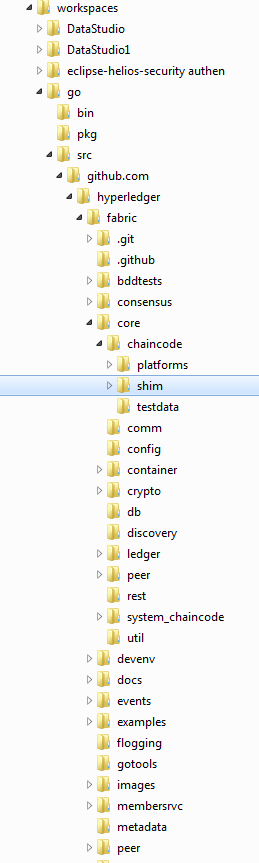
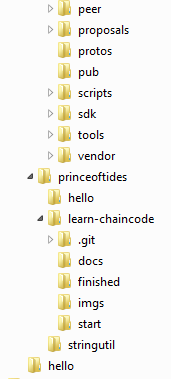
03/20/2017 02:59 PM <DIR> learn-chaincode

0 File(s) 0 bytes

Dir(s) 428,660,666,368 bytes free

You now have a copy of your learn chaincode fork on your local machine.

You will develop your chaincode by making changes to these local files, pushing them to your fork on GitHub, and then deploying the code onto your blockchain network using the REST API on one of your peers.

3- Two different versions of the chaincode used in this tutorial are provided: \**start** is the skeleton chaincode that you will start from ( can also deploy after making API coding chgs), and \**finished is your completed chaincode that is ready to build. First, make sure that**\**start**  builds in your local environment even before coding:

1. cd $GOPATH/src/github.com/<YOUR\_GITHUB\_ID\_HERE>/learn-chaincode/start
2. go build

**The start version of learn-chaincode should compile with no errors or messages. If it does not, review the prior instructions for installing Go correctly.**

Transactions may be of two types:

Deploy transactions create new chaincode and take a program as parameter. When a deploy transaction executes successfully, the chaincode has been installed “on” the blockchain.

Invoke transactions perform an operation in the context of previously deployed chaincode. An invoke transaction refers to a chaincode and to one of its provided functions. When successful, the chaincode executes the specified function - which may involve modifying the corresponding state, and returning an output.

As described later, deploy transactions are special cases of invoke transactions, where a deploy transaction that creates new chaincode, corresponds to an invoke transaction on a system chaincode, and execs Init()

## **4.1.2 Implement the chaincode interface APIs**

#### **Implement the chaincode interface**

In order to turn a piece of Golang (or Java) code into chaincode, at least provide and implemention of the chaincode (fabric) shim interface in your Golang code. The three main functions (to at least provide) are **Init**, **Invoke** and **Query**. - All three functions have the same func prototype: take a function name and an array of strings as input, but the main difference is when they are called.

Your development path ends with building a working chaincode to create generic assets for exchange on a blockchain network.

**Dependencies**

The import statement lists the dependencies for building your chaincode:

1. fmt - contains Println for debugging/logging.
2. errors - standard Go error format.
3. github.com/hyperledger/fabric/core/chaincode/shim - contains the definition for the chaincode interface and the chaincode stub, which you will need to interact with the ledger. Its code that interfaces your Golang code with a network peer.

**Todo: how to create Go chaincode projects in Eclipse?**

#### **Init()**

The Init function is called when you first deploy your chaincode. As the name implies, use this function to initialize your chaincode. In this example, Init configures the initial state of a single key/value pair **on the ledger.**

In your chaincode\_start.go file, change the Init function so that it stores the first args element to the key "hello\_world":

func (t \*SimpleChaincode) Init(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) {

if len(args) != 1 {

return nil, errors.New("Incorrect number of arguments. Expecting 1")

}

err := stub.PutState("hello\_world", []byte(args[0]))

if err != nil {

return nil, err

}

return nil, nil

}

This is done by using the stub function stub.PutState. This function interprets the first argument sent in the deployment request as the value to be stored under the key 'hello\_world'. If an error occurs because the wrong number of arguments was passed in, or because something went wrong when writing to the ledger, this function returns an error. Otherwise, it exits cleanly, returning no messages.

#### **Invoke()**

Use the Invoke function to call chaincode functions to do "real work" on the blockchain network.

An invocation results in a transaction being added to the chain. Invocations are captured as transactions, which get grouped into blocks for writing to the chain ledger. Updating the ledger is achieved by **invoking** (Invoke() ) your chaincode.

The structure of Invoke() is simple; it **receives a function and** based on this func arg, calls Go functions in the chaincode, or returns an error.

? security window of opportunity?

In your chaincode\_start.go file, change the Invoke function to call a generic write function:

func (t \*SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) {

fmt.Println("invoke is running " + function)

// Handle different functions

if function == "init" {

return t.Init(stub, "init", args)

} else if function == "write" {

return t.write(stub, args)

}

fmt.Println("invoke did not find func: " + function)

return nil, errors.New("Received unknown function invocation")

}

C:\work\code\workspaces\go\src\github.com\princeoftides\learn-chaincode\start>go

build

# github.com/princeoftides/learn-chaincode/start

.\chaincode\_start.go:88: t.write undefined (type \*SimpleChaincode has no field o

r method write)

.\chaincode\_start.go:88: not enough arguments to return

have (<T>)

want ([]byte, error)

The code is now looking for write, so add the write function to your chaincode\_start.go file:

func (t \*SimpleChaincode) write(stub shim.ChaincodeStubInterface, args []string) ([]byte, error) {

var name, value string

var err error

fmt.Println("running write()")

if len(args) != 2 {

return nil, errors.New("Incorrect number of arguments. Expecting 2. name of the variable and value to set")

}

name = args[0] //rename for fun

value = args[1]

err = stub.PutState(name, []byte(value)) //write the variable into the chaincode state

if err != nil {

return nil, err

}

return nil, nil

}

This write function should look similar to your previous Init change. You can now set the key and value for PutState, which allows you to store any key/value pair on the blockchain ledger.

#### **Query()**

The Query function is called to query your chaincode state, and does not add blocks to the chain (ledger). Only **deploy and invoke functions add new blocks**. Use Query to read the value of your chaincode state's key/value pairs.

In your chaincode\_start.go file, change the Query function so that it calls a generic read function:

func (t \*SimpleChaincode) Query(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) {

fmt.Println("query is running " + function)

// Handle different functions

if function == "read" { //read a variable

return t.read(stub, args)

}

fmt.Println("query did not find func: " + function)

return nil, errors.New("Received unknown function query")

}

The code is now looking for read, so add the 'read' function to your chaincode\_start.go file:

func (t \*SimpleChaincode) read(stub shim.ChaincodeStubInterface, args []string) ([]byte, error) {

var name, jsonResp string

var err error

if len(args) != 1 {

return nil, errors.New("Incorrect number of arguments. Expecting name of the var to query")

}

name = args[0]

valAsbytes, err := stub.GetState(name)

if err != nil {

jsonResp = "{\"Error\":\"Failed to get state for " + name + "\"}"

return nil, errors.New(jsonResp)

}

return valAsbytes, nil

}

This read function uses GetState**, which is the complement to**PutState. This shim function takes only one string argument: the **name of the key to retrieve**. Next, this function **returns the value**, as an array of bytes, to Query, which in turn sends it to the REST handler.

#### Main()

The main function executes when each peer deploys its instance of the chaincode. It starts the chaincode and registers it with the peer. No code updates are required for 'main'; both chaincode\_start.go and chaincode\_finished.go include a main function at the top of each file:

func main() {

err := shim.Start(new(SimpleChaincode))

if err != nil {

fmt.Printf("Error starting Simple chaincode: %s", err)

}

}

**Build the Final API code changes**

go build

Write code changes to your local chaincode files, verify clean compile and push the updated files to my GitHub repos learn-chaincode fork.

The BC bmix service will read the chaincode and adds it to the BC network needs to do that from a GitHub repository.

Write changes to your local chaincode files, and push the updated files to your GitHub fork:

C:\work\code\workspaces\go\src\github.com\princeoftides\learn-chaincode\start>

**git status**

On branch v2.0

Your branch is up-to-date with 'origin/v2.0'.

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

**modified: chaincode\_start.go**

no changes added to commit (use "git add" and/or "git commit -a")

**git commit -m "Compiled my code 4-5-17"**

On branch v2.0

Your branch is up-to-date with 'origin/v2.0'.

Changes not staged for commit:

modified: chaincode\_start.go

**no changes added to commit**

**git add --all**

**git commit -m "Compiled my code 4-5-17"**

[v2.0 9a3482e] Compiled my code 4-5-17

1 file changed, 3 insertions(+)

# Push local commits back to https://github.com/<YOUR\_GITHUB\_ID\_HERE>/learn-chaincode/

**git push**

warning: push.default is unset; its implicit value has changed in

Git 2.0 from 'matching' to 'simple'. To squelch this message

and maintain the traditional behavior, use:

git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

**git config --global push.default simple**

When push.default is set to 'matching', git will push local branches

to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'

behavior, which only pushes the current branch to the corresponding

remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.

(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode

'current' instead of 'simple' if you sometimes use older versions of Git)

**Username for 'https://github.com': princeoftides**

**Password for 'https://princeoftides@github.com':**

Counting objects: 4, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (4/4), 446 bytes | 0 bytes/s, done.

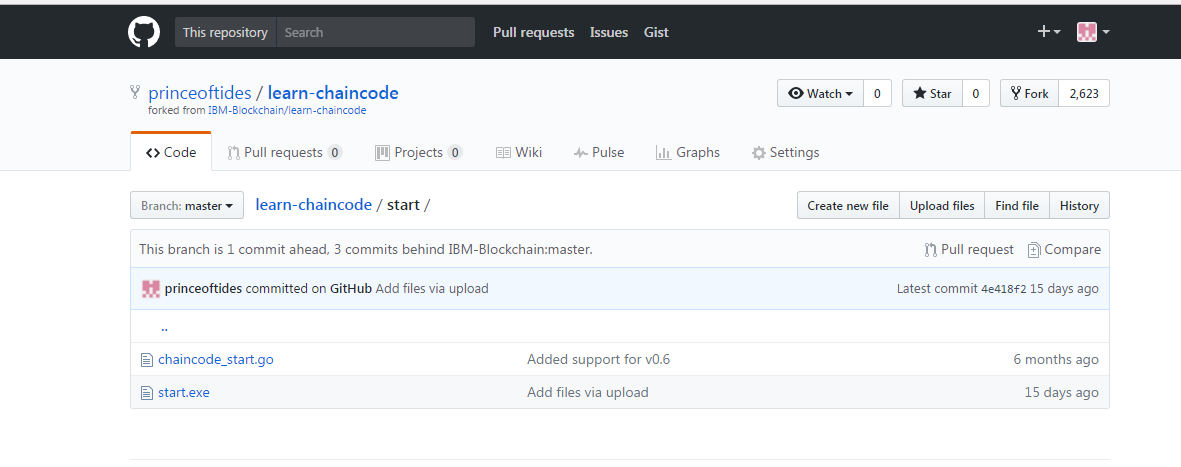
Total 4 (delta 2), reused 0 (delta 0)

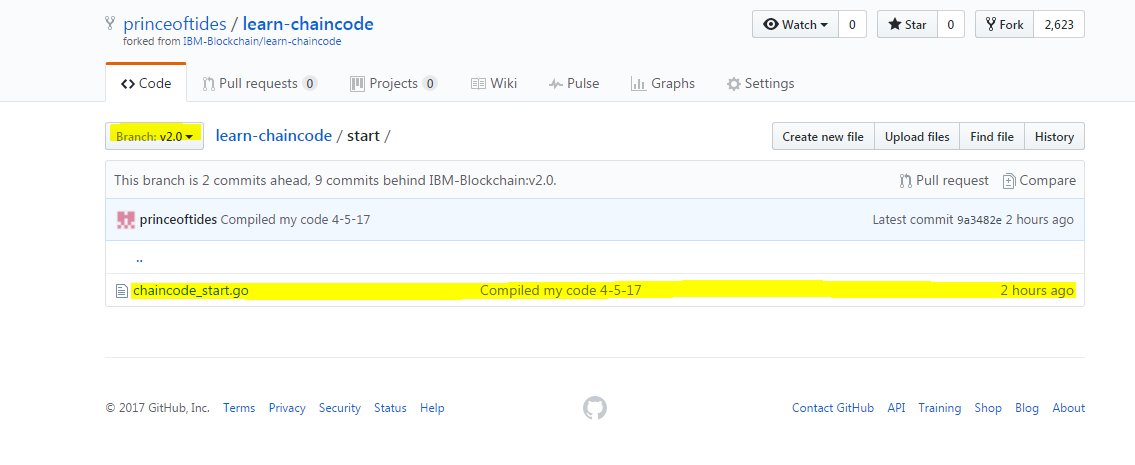
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.

**To https://github.com/princeoftides/learn-chaincode.git**

4e8491a..9a3482e v2.0 -> v2.0

-or- “Everything up-to-date” if no chgs needed.





* My chaincode is up on my GitHub ready to be imported into my Blockchain service.

**?: Does start.exe** from my windows local machine get pushed to my github? Cant manually pull a .exe from github.

## **4.1.3 Interacting with your chaincode**

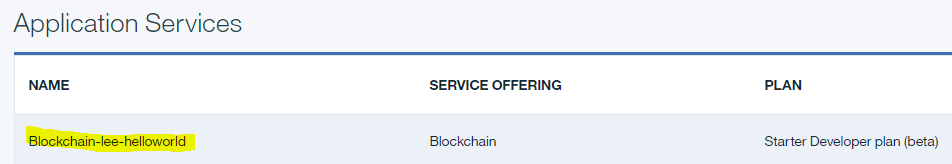
The fastest way to test your chaincode is to use the REST interface on your peers. The Swagger UI on your Bluemix dashboard monitor allows you to experiment with deploying chaincode, without writing any additional code.

Deploy the chaincode into the BC service network and interact with it.

#### **To use BC Swagger API**

Complete the following steps to use the Swagger API:

1. Log in to [Bluemix](https://console.ng.bluemix.net/login) and ensure that you are on the **Dashboard** tab.
2. Check that you are in the same Bluemix "space" that contains your IBM Blockchain service. The space navigation is on the left.
3. There is a **Services** panel near the bottom; click on your IBM Blockchain service.
   * Blockchain only available from CTS public Bmix server – early BC adopter – not my personal Bmix acct.



1. You should see a "Welcome to the IBM Blockchain..." message; click on the **LAUNCH** button on the right.
2. On the monitor page, you should see two tables; the bottom table may be empty.
   * **Network Tab:**
     + **Logs Tab - Peer Logs** are in the top table. In the row for **peer 1**, click the file icon to view the log. In addition to this static view, there are live **streaming peer logs** in the **View Logs** tab near the top of the page.
     + **ChainCode Logs** are in the bottom table. Each chaincode is labeled with the chaincode hash that was returned when it was deployed. Select the peer in any chaincode row, and then click the file icon to view the log.
   * **APIs Tab**: Displays the Swagger APIs documentation page.
3. Continue with the following steps to **implement secure enrollment**

* Calls to the /chaincode endpoint in the (ie. Swagger) REST interface require a **secure context ID**.

For most REST calls to be accepted, you must pass a **registered enrollID from the service credentials list**:

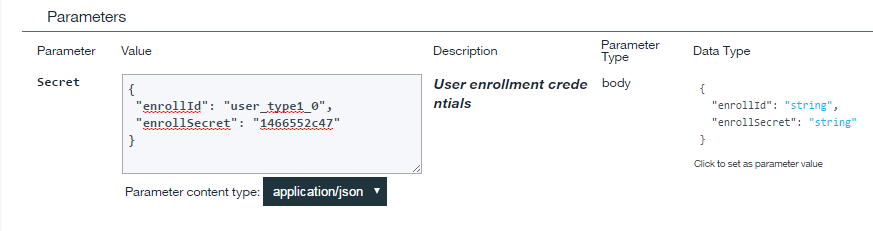
* + ~~Click~~ **~~Network's Enroll IDs~~** BC service credentials to expand the list of enrollID values and their secrets.
  + Copy the set of credentials to a text file for later use.
  + Expand the **Registrar** API section.
  + Expand the POST /registrar section.
  + Populate the Value field with JSON that specifies an enrollID and `enrollSecret' from your credentials:

Starter network ID = 27b8c6086dea46e996de9dfbc32f9e4b

"users": [  
 {  
 **"enrollId": "user\_type1\_0",  
 "enrollSecret": "1466552c47",** "affiliation": "group1",  
 "username": "user\_type1\_0",  
 "secret": "1466552c47"  
 },

After deploying this is my chaincode id: "3aeb9793d67968f966f2b093c361c70cdbf7a2813a02f7a5da344386580d3b519899b73003b335c587e3d016d44b54eb7d8030bddddbc3e9abf05db81c20eaef"

Try It out!



The Response Body :

#### Curl

#### curl -X POST --header "Content-Type: application/json" --header "Accept: application/json" -d "{

#### \"enrollId\": \"user\_type1\_0\",

#### \"enrollSecret\": \"1466552c47\"

#### }

#### " "https://27b8c6086dea46e996de9dfbc32f9e4b-vp0.us.blockchain.ibm.com:5003/registrar"

#### Request URL

#### https://27b8c6086dea46e996de9dfbc32f9e4b-vp0.us.blockchain.ibm.com:5003/registrar

#### Response Body

#### {

#### "OK": "Login successful for user 'user\_type1\_0'."

#### }

#### Response Code

#### 200Response Headers

{  
 "content-type": "application/json"  
}

Now that you have an enrollID set up, you can **use this ID when deploying, invoking, and querying chaincode** in the following steps.

## **4.1.4 Deploying your chaincode**

In order to deploy your chaincode through the REST interface, your chaincode must be stored in a public GitHub repository. When you send a deploy request to a peer, you specify the URL for the chaincode repository, and the parameters to initialize the chaincode.

**Before you deploy** your chaincode, verify that it builds locally:

1. Open a command prompt, and browse to the directory that contains chaincode\_start.go. Enter the following command:
2. go build ./
3. Expand the APIs **Chaincode** API section.
4. Expand the POST /chaincode section.
5. Set the DeploySpec text field (make the other fields blank) with the example code below, specifying your chaincode repository path, and the enrollID from the previous /registrar step. The "path": should look similar to: "https://github.com/johndoe/learn-chaincode/finished". This is the path to your repository fork, plus the path to your chaincode\_finished.go file:

~~{~~

~~"jsonrpc": "2.0",~~

~~"method": "deploy",~~

~~"params": {~~

~~"type": 1,~~

~~"chaincodeID": {~~

~~// "path": “~~[~~https://github.com/princeoftides/learn-chaincode/start~~](https://github.com/princeoftides/learn-chaincode/start)~~”~~

“path”: “https://github.com/princeoftides/learn-chaincode/blob/v2.0/start”

~~},~~

~~"ctorMsg": {~~

~~"function": "init",~~

~~"args": [~~

~~"hi there"~~

~~]~~

~~},~~

~~"secureContext": "user\_type1\_0"~~

~~},~~

~~"id": 1~~

~~}~~

{

"jsonrpc": "2.0",

"method": "deploy",

"params": {

"type": 1,

"chaincodeID": {

"path": "https://github.com/princeoftides/learn-chaincode/finished"

},

"ctorMsg": {

"function": "init",

"args": [

"hi there"

]

},

"secureContext": "user\_type1\_0"

},

"id": 0

}

#### Curl

curl -X POST --header "Content-Type: application/json" --header "Accept: application/json" -d "{

\"jsonrpc\": \"2.0\",

\"method\": \"deploy\",

\"params\": {

\"type\": 1,

\"chaincodeID\": {

\"path\": \"https://github.com/princeoftides/learn-chaincode/finished\"

},

\"ctorMsg\": {

\"function\": \"init\",

\"args\": [

\"hi there\"

]

},

\"secureContext\": \"user\_type1\_0\"

},

\"id\": 0

}" "https://27b8c6086dea46e996de9dfbc32f9e4b-vp0.us.blockchain.ibm.com:5003/chaincode"

#### Request URL

https://27b8c6086dea46e996de9dfbc32f9e4b-vp0.us.blockchain.ibm.com:5003/chaincode

#### Response Body

{

"jsonrpc": "2.0",

"result": {

**"status": "OK",**

"message": **"3aeb9793d67968f966f2b093c361c70cdbf7a2813a02f7a5da344386580d3b519899b73003b335c587e3d016d44b54eb7d8030bddddbc3e9abf05db81c20eaef" (this is my new chaincode id)**

},

"id": 0

}

#### Response Code

200

#### Response Headers

{  
 "content-type": "application/json"  
}

But why using this my github path: “path”: “https://github.com/princeoftides/learn-chaincode/blob/v2.0/start”

Get this error:

#### Response Body

{

"jsonrpc": "2.0",

"error": {

"code": -32001,

"message": "Deployment failure",

"data": "Error when deploying chaincode: Error getting chaincode package bytes: Error getting code 'go get' failed with error: \"exit status 1\"\npackage github.com/princeoftides/learn-chaincode/blob/v2.0/start: cannot find package \"github.com/princeoftides/learn-chaincode/blob/v2.0/start\" in any of:\n\t/opt/go/src/github.com/princeoftides/learn-chaincode/blob/v2.0/start (from $GOROOT)\n\t/opt/gopath/\_usercode\_/945117110/src/github.com/princeoftides/learn-chaincode/blob/v2.0/start (from $GOPATH)\n\t/opt/gopath/src/github.com/princeoftides/learn-chaincode/blob/v2.0/start\n"

},

"id": 0

}

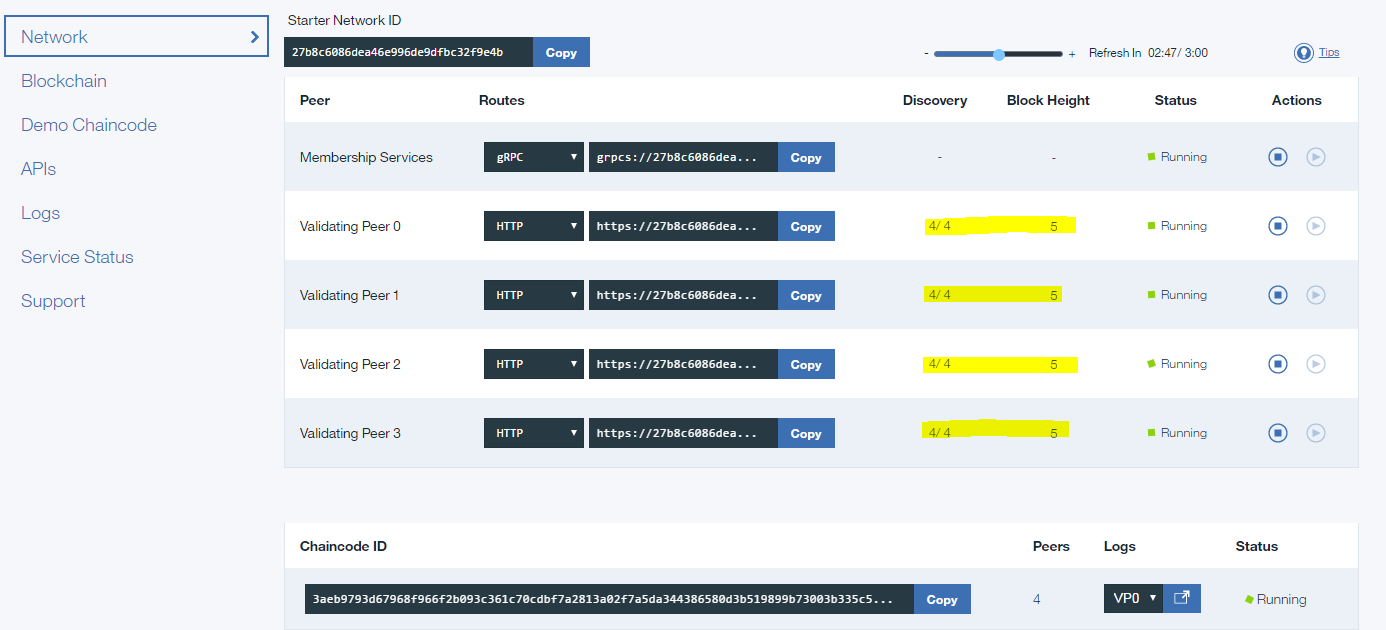
#### Response Code

200

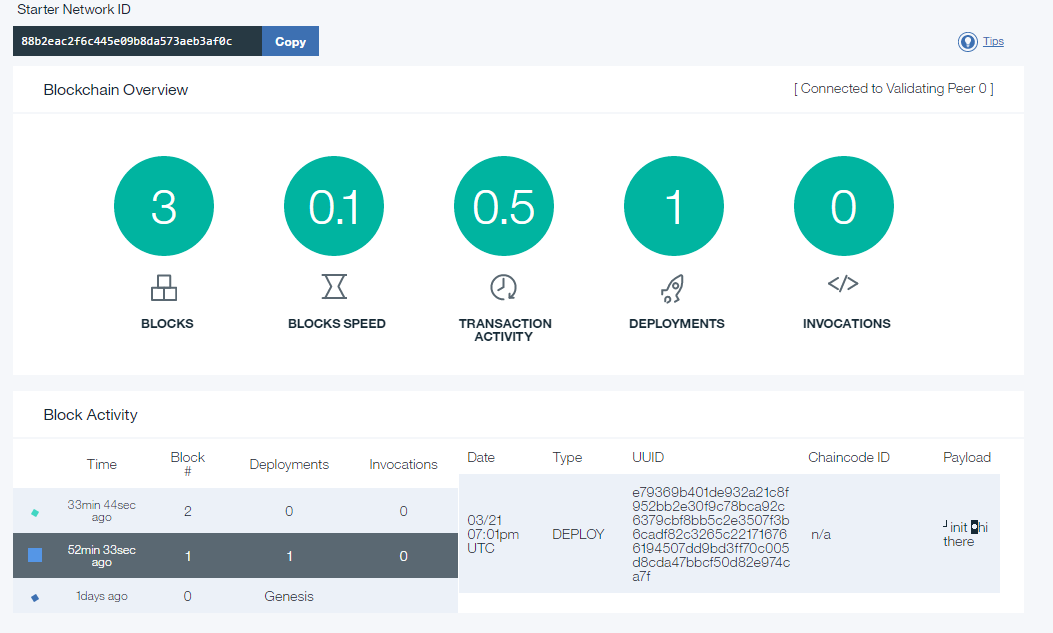
The good Response Body should look similar to the following example:



The response for the deployment will contain the ID for this chaincode. The ID is a 128 character alphanumeric hash, which you can use for future invoke or query requests. Save this ID.



**Use Browser + Postman to debug errors**



#### Query

Query your chaincode for the value of the hello\_world key, which you set with the Init function:

1. Expand the **Chaincode** API section.
2. Expand the POST /chaincode section.
3. Populate the QuerySpec text field (make the other fields blank) with the following example, specifying the chaincode ID from the previous deployment step, and the enrollID from the previous /registrar step:

{

"jsonrpc": "2.0",

"method": "query",

"params": {

"type": 1,

"chaincodeID": {

"name": "3aeb9793d67968f966f2b093c361c70cdbf7a2813a02f7a5da344386580d3b519899b73003b335c587e3d016d44b54eb7d8030bddddbc3e9abf05db81c20eaef"

},

"ctorMsg": {

"function": "read",

"args": [

"hello\_world"

]

},

"secureContext": "user\_type1\_0"

},

"id": 0

}

This error bcos my finished chaincode was not pushed to my github.

~~{~~

~~"jsonrpc": "2.0",~~

~~"error": {~~

~~"code": -32003,~~

~~"message": "Query failure",~~

~~"data": "Error when querying chaincode: Error:Transaction or query returned with failure: Received unknown function query: read"~~

~~},~~

~~"id": 2~~

~~}~~

This error bcos of a simple “ “ space before the secureContext userid

#### Curl

curl -X POST --header "Content-Type: application/json" --header "Accept: application/json" -d "{

\"jsonrpc\": \"2.0\",

\"method\": \"query\",

\"params\": {

\"type\": 1,

\"chaincodeID\": {

\"name\": \"3aeb9793d67968f966f2b093c361c70cdbf7a2813a02f7a5da344386580d3b519899b73003b335c587e3d016d44b54eb7d8030bddddbc3e9abf05db81c20eaef\"

},

\"ctorMsg\": {

\"function\": \"read\",

\"args\": [

\"hello\_world\"

]

},

\"secureContext\": \" user\_type1\_0\"

},

\"id\": 0

}

" "https://27b8c6086dea46e996de9dfbc32f9e4b-vp0.us.blockchain.ibm.com:5003/chaincode"

#### Request URL

https://27b8c6086dea46e996de9dfbc32f9e4b-vp0.us.blockchain.ibm.com:5003/chaincode

#### Response Body

{

"jsonrpc": "2.0",

"error": {

"code": -32000,

"message": "Registration missing",

"data": "User not logged in. Use the '/registrar' endpoint to obtain a security token."

},

"id": 0

}

#### Response Code

200

#### Response Headers

{  
 "content-type": "application/json"  
}

Good response:

#### Curl

curl -X POST --header "Content-Type: application/json" --header "Accept: application/json" -d "{

\"jsonrpc\": \"2.0\",

\"method\": \"query\",

\"params\": {

\"type\": 1,

\"chaincodeID\": {

\"name\": \"3aeb9793d67968f966f2b093c361c70cdbf7a2813a02f7a5da344386580d3b519899b73003b335c587e3d016d44b54eb7d8030bddddbc3e9abf05db81c20eaef\"

},

\"ctorMsg\": {

\"function\": \"read\",

\"args\": [

\"hello\_world\"

]

},

\"secureContext\": \"user\_type1\_0\"

},

\"id\": 0

}

" "https://27b8c6086dea46e996de9dfbc32f9e4b-vp0.us.blockchain.ibm.com:5003/chaincode"

#### Request URL

https://27b8c6086dea46e996de9dfbc32f9e4b-vp0.us.blockchain.ibm.com:5003/chaincode

#### Response Body

{

"jsonrpc": "2.0",

"result": {

"status": "OK",

**"message": "hi there" (The Golang code Init() function call initialized “hello\_world” key to this value string)**

},

"id": 0

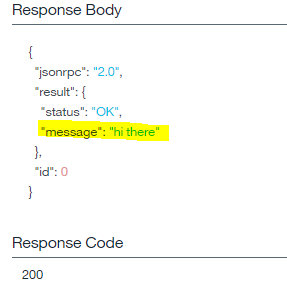
}

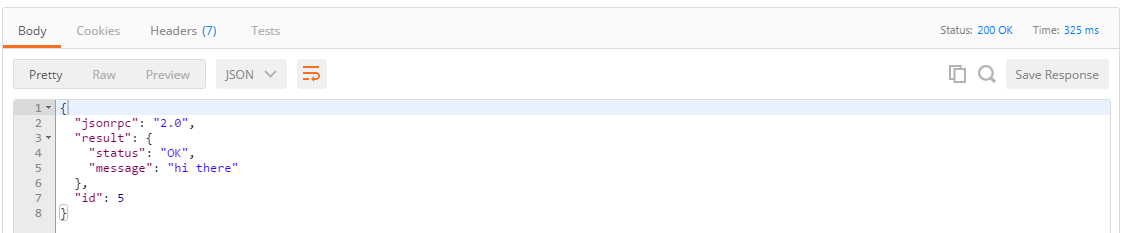
#### Response Code

200

#### Response Headers

{  
 "content-type": "application/json"  
}





The value of hello\_world is "hi there", which was set by the body of your previous deploy call.

#### Invoke

<https://github.com/IBM-Blockchain/learn-chaincode>

Now, call the generic write() finction with invoke(). Change the value of “hello\_world” to “go away”.

>Expand Chaincode Api section

>Expand POST /chaincode

>Set InvokeSpec text field (all others blank) Copy the example, subs yr chaincode name(hashed ID from deploy), same enrollID from /registrar step.

* Create a POST request body InvokeSpec like the example below.

{

"jsonrpc": "2.0",

"method": "invoke",

"params": {

"type": 1,

"chaincodeID": {

"name": "3aeb9793d67968f966f2b093c361c70cdbf7a2813a02f7a5da344386580d3b519899b73003b335c587e3d016d44b54eb7d8030bddddbc3e9abf05db81c20eaef"

},

"ctorMsg": {

"function": "write",

"args": [

"hello\_world", "go away"

]

},

"secureContext": "user\_type1\_0"

},

"id": 0

}

* The body for the request:

{

"jsonrpc": "2.0",

"method": "invoke",

"params": {

"type": 1,

"chaincodeID": {

"name": "<CHAINCODE\_HASH\_HERE>"

},

"ctorMsg": {

"function": "write",

"args": [

"hello\_world", "go away"

]

},

"secureContext": "<YOUR\_USER\_HERE>"

},

"id": 3

}

* Send the request. If everything goes smoothly, you will see a response like the one below:

#### Response Body

{

"jsonrpc": "2.0",

"result": {

"status": "OK",

"message": "07b6c35c-4f14-4a2c-a93c-1311dcd942fb"

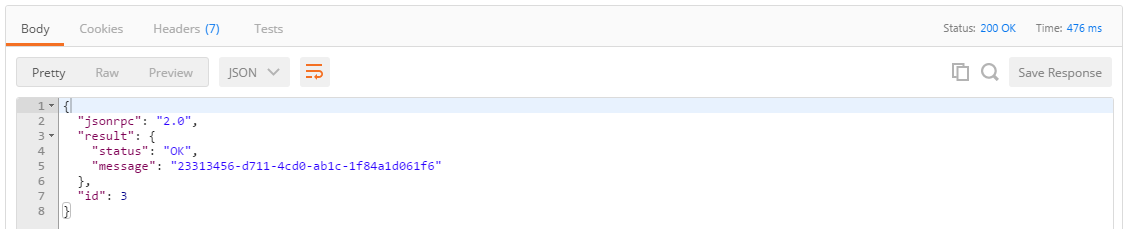
},

"id": 0

}

#### Response Code

200

[](https://github.com/IBM-Blockchain/learn-chaincode/blob/master/imgs/invoke_response.PNG)

* Test if our change stuck by sending another query like the one from before.

#### Response Body

{

"jsonrpc": "2.0",

"result": {

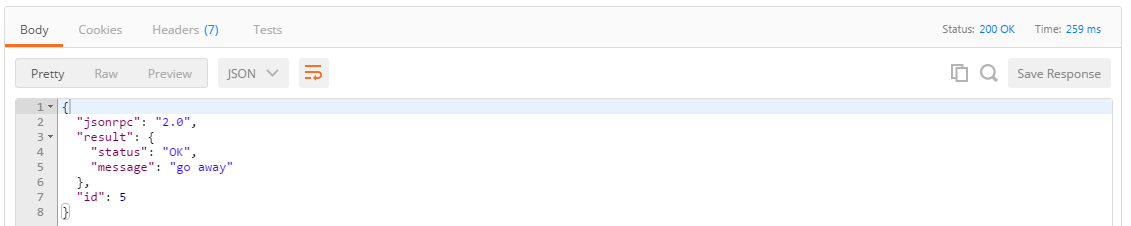
"status": "OK",

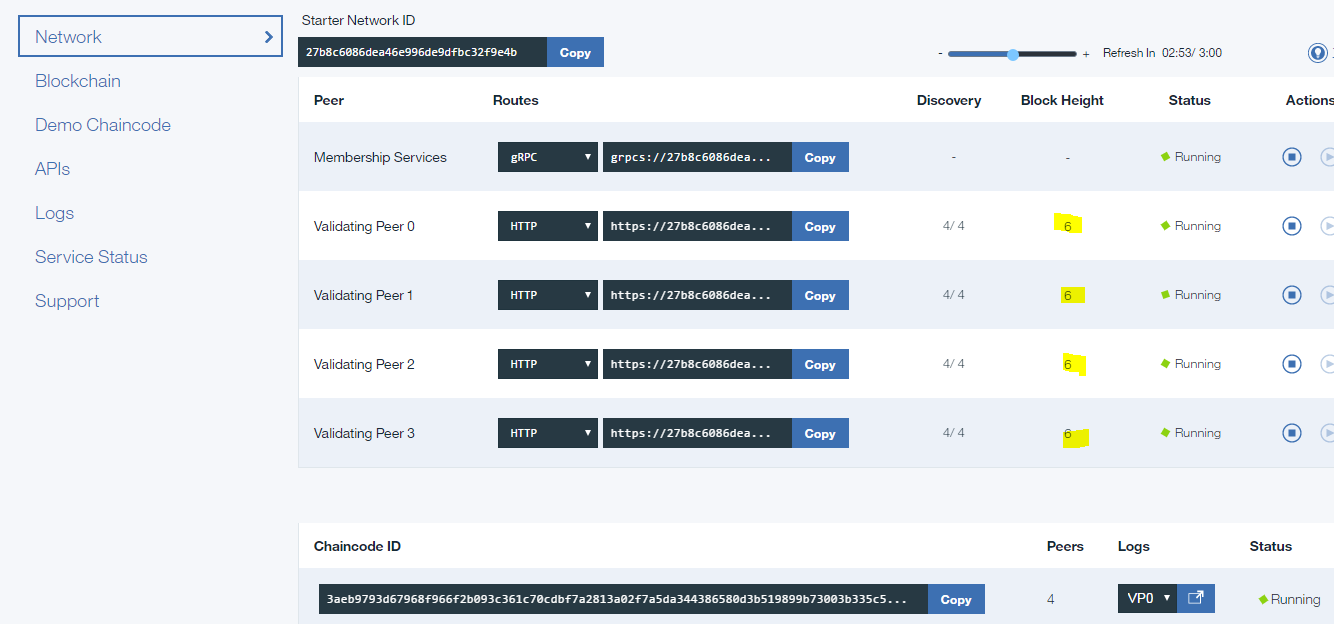
**"message": "go away"**

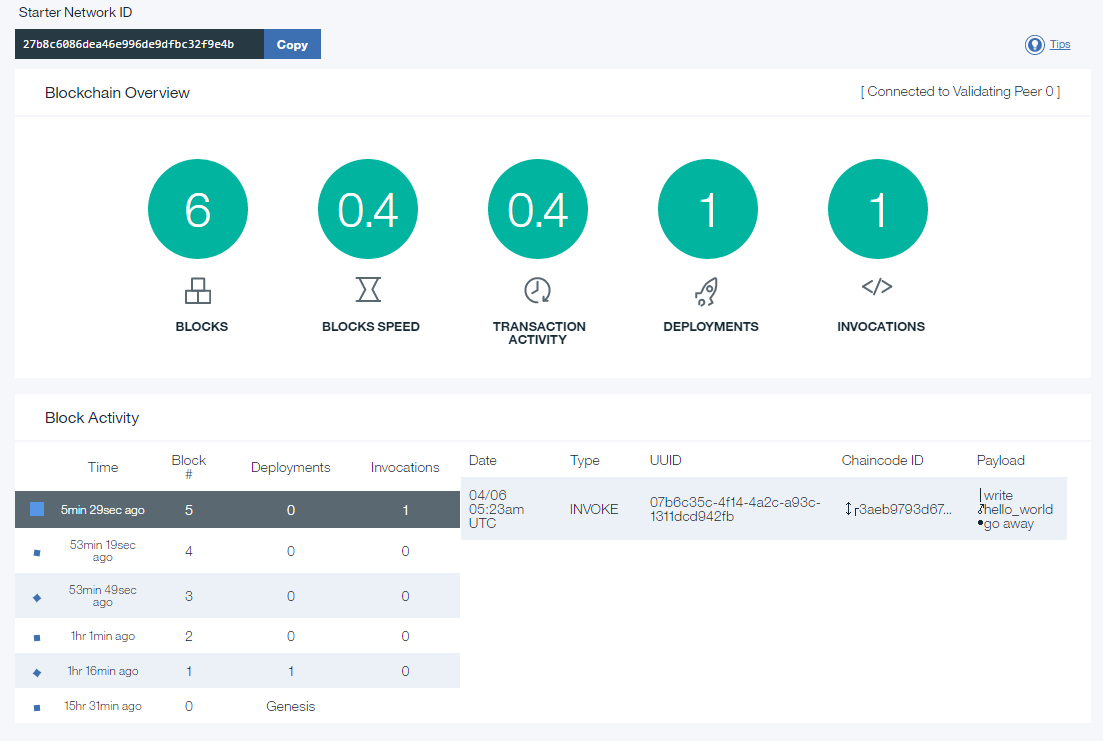
},

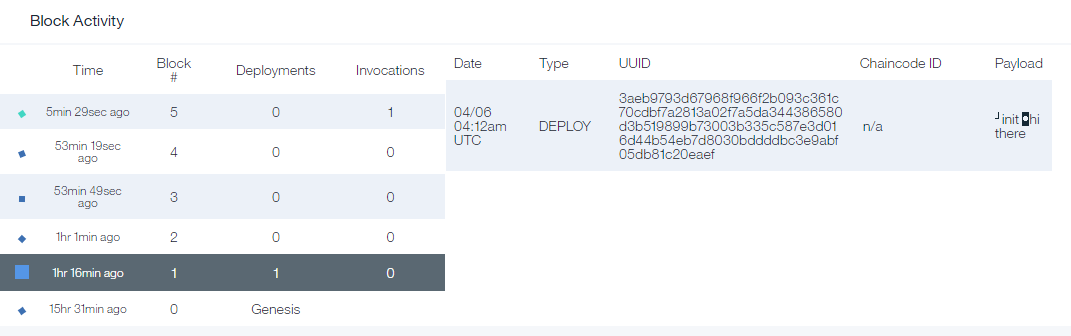
"id": 0

}

[](https://github.com/IBM-Blockchain/learn-chaincode/blob/master/imgs/query2_response.PNG)







* No blocks for the queries – those arent transactions –just reads.

That's all it takes to write basic chaincode. ;)

* "Helloworld" Blockchain chaincode working sample app. Its deployed on this route (for now) :

public Bmix - Org: COG-EAS-IPM-IBM Space: KG-EAS-IPM-GDC-IBM serviceName: “Blockchain-lee-helloworld”

# **HL Fabric Composer – Playground - CLI Development**

## **4.2.1 Set-up FabricComposer Playground on my machine using Docker ToolBox**

To set-up FabricComposer Playground on my machine using Docker ToolBox – also works in Windows 7.

 Windows 10 beyond this point in this guide refer to all 32-bit editions and 64-bit non-Pro or non-Ent editions. All other versions of Windows 10 already have Hyper-V and therefore [docker can be installed natively](https://www.htpcbeginner.com/install-docker-on-windows-10/) instead of with Docker Toolbox.

Setting up docker using Docker Toolbox does not make Docker run natively on Windows. Instead, it uses what is called a docker-machine to create a virtual machine (VM) on VirtualBox. Therefore, if you do not have (Oracle VM) VirtualBox, Docker Toolbox automatically installs it for you. You do not have to mess with VirtualBox. Docker Toolbox automatically creates a Linux VM on VirtualBox that hosts Docker on your Windows system.

It looks like Playground is combination of Fabric composer & Composer-UI. We don’t need UI separately.

Next step is to play around with composer to get some hands-on.

Pre-reqs : to install in your machine

* Install **GitBash**
* Install **Docker toolbox**

Before you begin, [enable Intel VT-x hardware virtualization in BIOS or UEFI firmware](https://www.htpcbeginner.com/enable-hardware-virtualization-vt-x-amd-v/).

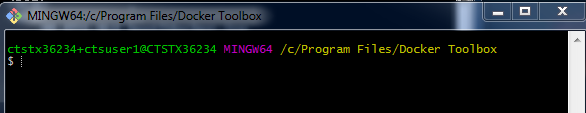
This is required for Docker to run. Once this is done, installing Docker on Windows is as simple as downloading the installer and running it as you would for any software installation.

Installs Docker Client, Machine, Compose and Kitematic.

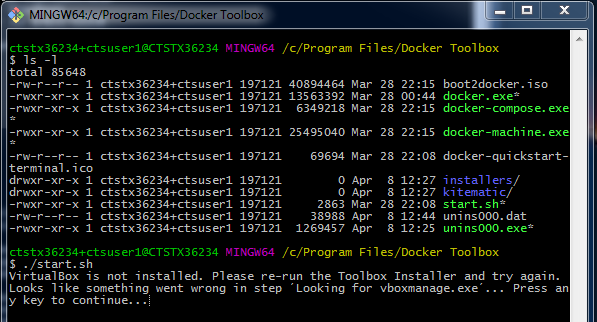
C:\Program Files\Docker Toolbox

Once you install follow below steps

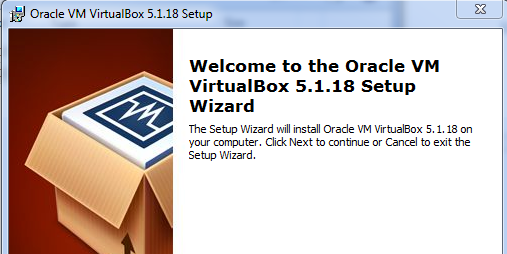
Open cmdline window Go to  Docker installed folder : “C:\Program Files\Docker Toolbox” and start Gitbash from that folder: "C:\Program Files\Git\git-bash.exe"



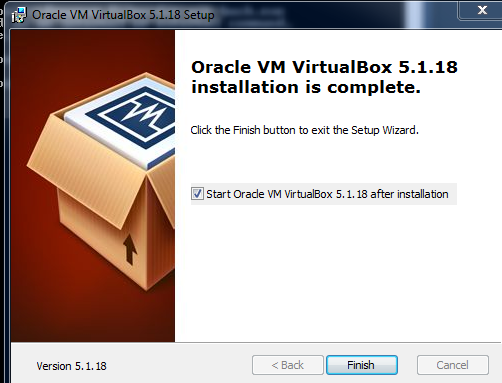
1. run start.sh file



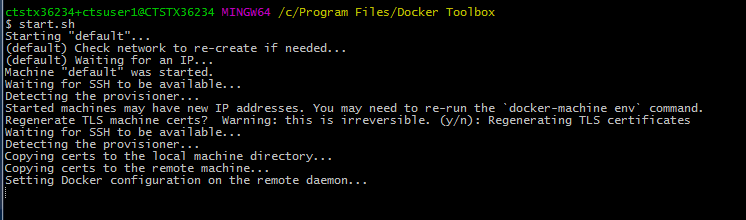
C:\Program Files\Docker Toolbox\installers\virtualbox

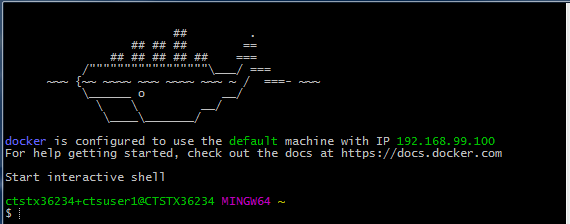


“Choose Repair”



Start.sh again >





1. docker run --rm hello-world   cleanup and fresh start

$ docker run --rm hello-world

time="2017-04-09T15:11:17-04:00" level=info msg="Unable to use system certificate pool: crypto/x509: system root pool is not available on Windows"

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

78445dd45222: Pulling fs layer

78445dd45222: Verifying Checksum

78445dd45222: Download complete

78445dd45222: Pull complete

Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://cloud.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/engine/userguide/

1. docker run -d -p 8080:8080 fabriccomposer/composer-playground

ctstx36234+ctsuser1@CTSTX36234 MINGW64 ~

$ docker run -d -p 8080:8080 fabriccomposer/composer-playground

time="2017-04-09T15:12:56-04:00" level=info msg="Unable to use system certificate pool: crypto/x509: system root pool is not available on Windows"

Unable to find image 'fabriccomposer/composer-playground:latest' locally

latest: Pulling from fabriccomposer/composer-playground

709515475419: Pulling fs layer

cb714116d354: Pulling fs layer

3dd304580c96: Pulling fs layer

9106602805f7: Pulling fs layer

009f02e4eb0f: Pulling fs layer

5c93c68c8bfb: Pulling fs layer

9106602805f7: Waiting

009f02e4eb0f: Waiting

5c93c68c8bfb: Waiting

709515475419: Verifying Checksum

709515475419: Download complete

3dd304580c96: Verifying Checksum

3dd304580c96: Download complete

709515475419: Pull complete

cb714116d354: Verifying Checksum

cb714116d354: Download complete

cb714116d354: Pull complete

3dd304580c96: Pull complete

5c93c68c8bfb: Verifying Checksum

5c93c68c8bfb: Download complete

9106602805f7: Verifying Checksum

9106602805f7: Download complete

009f02e4eb0f: Verifying Checksum

009f02e4eb0f: Download complete

9106602805f7: Pull complete

009f02e4eb0f: Pull complete

5c93c68c8bfb: Pull complete

Digest: sha256:f4b3d75a8a8b882e5d034f3487d474b85b50ee0bb2ba4d13c09122e83cb36ad4

Status: Downloaded newer image for fabriccomposer/composer-playground:latest

81692a4ccfc45e79f484b20f3fd7a225f5c1e7bca9c22c5bdabc436d7217271a

After 1st run, subsequent sysout :

$ docker run -d -p 8080:8080 fabriccomposer/composer-playground

time="2017-04-19T12:47:02-04:00" level=info msg="Unable to use system certificate pool: crypto/x509: system root pool is not available on Windows"

dc049bd41aad9957ee872c2c79fe655f18c5278e69cad15dc878369700869b73

5. docker ps  shows all containers running

$ docker ps

time="2017-04-09T15:16:37-04:00" level=info msg="Unable to use system certificate pool: crypto/x509: system root pool is not available on Windows"

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

81692a4ccfc4 fabriccomposer/composer-playground "pm2-docker compos..." 3 minutes ago Up 3 minutes 0.0.0.0:8080->8080/tcp kind\_pasteur

6. <http://192.168.99.100:8080/editor> - open the Compser UI…

7. docker stop <docker containerid>  to stop docker container - get id from docker ps

## **4.2.2 Fabric Composer CLI setup**

See: <https://fabric-composer.github.io/reference/commands.html>

install composer CLI in my Cognizant laptop

* create my own Smart contracts
* import it to ComposerPlayground successfully and submit some transactions.

Contains the Fabric composer CLIs for administering business networks.

Type composer --help to list the available commands.

npm install -g composer-cli

**Unexpected Install error :**

C:\Program Files\nodejs\node\_modules\composer-cli\node\_modules\hashtable>if not

defined npm\_config\_node\_gyp (node "C:\work\tools\nvm\v6.1.0\node\_modules\npm\bin

\node-gyp-bin\\..\..\node\_modules\node-gyp\bin\node-gyp.js" configure build )  e

lse (node "" configure build )

Building the projects in this solution one at a time. To enable parallel build,

please add the "/m" switch.

C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V140\**Microsoft.CppCommon.targ**

**ets(356,5): error MSB6006: "CL.exe" exited with code -1073741515.** [C:\Program F

iles\nodejs\node\_modules\composer-cli\node\_modules\hashtable\build\native.vcxpr

oj]

**gyp ERR! build error**

**gyp ERR! stack Error: `C:\Program Files (x86)\MSBuild\14.0\bin\msbuild.exe` fail**

**ed with exit code: 1**

gyp ERR! stack     at ChildProcess.onExit (C:\work\tools\nvm\v6.1.0\node\_modules

\npm\node\_modules\node-gyp\lib\build.js:285:23)

gyp ERR! stack     at emitTwo (events.js:106:13)

gyp ERR! stack     at ChildProcess.emit (events.js:191:7)

gyp ERR! stack     at Process.ChildProcess.\_handle.onexit (internal/child\_proces

s.js:204:12)

gyp ERR! System Windows\_NT 6.1.7601

gyp ERR! command "C:\\Program Files\\nodejs\\node.exe" "C:\\work\\tools\\nvm\\v6

.1.0\\node\_modules\\npm\\node\_modules\\node-gyp\\bin\\**node-gyp.js" "configure" "**

**build"**

**gyp ERR! cwd C:\Program Files\nodejs\node\_modules\composer-cli\node\_modules\hash**

**table**

gyp ERR! node -v v6.1.0

gyp ERR! node-gyp -v v3.6.0

gyp ERR! not ok

C:\Program Files\nodejs

`-- (empty)

npm WARN optional Skipping failed optional dependency /composer-cli/chokidar/fse

vents:

npm WARN notsup Not compatible with your operating system or architecture: fseve

[nts@1.1.1](mailto:nts@1.1.1)

npm ERR! Windows\_NT 6.1.7601

npm ERR! argv "C:\\Program Files\\nodejs\\node.exe" "C:\\Program Files\\nodejs\\

node\_modules\\npm\\bin\\npm-cli.js" "install" "-g" "composer-cli"

npm ERR! node v6.1.0

npm ERR! npm  v3.8.6

npm ERR! code ELIFECYCLE

npm ERR! [hashtable@2.0.2](mailto:hashtable@2.0.2) install: `node-gyp configure build`

npm ERR! Exit status 1

npm ERR!

npm ERR! Failed at the [hashtable@2.0.2](mailto:hashtable@2.0.2) install script 'node-gyp configure build'

.

npm ERR! Make sure you have the latest version of node.js and npm installed.

npm ERR! If you do, this is most likely a problem with the hashtable package,

npm ERR! not with npm itself.

npm ERR! Tell the author that this fails on your system:

npm ERR!     node-gyp configure build

npm ERR! You can get information on how to open an issue for this project with:

npm ERR!     npm bugs hashtable

npm ERR! Or if that isn't available, you can get their info via:

npm ERR!     npm owner ls hashtable

npm ERR! There is likely additional logging output above.

npm ERR! Please include the following file with any support request:

npm ERR!     C:\work\code\workspaces\go\src\github.com\hyperledger\fabric\core\c

haincode\shim\npm-debug.log

npm ERR! code 1

C:\work\code\workspaces\go\src\github.com\hyperledger\fabric\core\chaincode\shim

>

Follow:

[Configuring your Windows development environment](https://github.com/Microsoft/nodejs-guidelines/blob/master/windows-environment.md#compiling-native-addon-modules)

Options #1 Run : npm install --global --production windows-build-tools - didn’t help

Option #2 :

**Coding a Business Network Definition**

See: <https://fabric-composer.github.io/business-network/getting-started-coding-bnd.html>

For: **function** onRegisterPropertyForSale(propertyForSale) {

returAssetRegistry('net.biz.digitalPropertyNetwork.LandTitle').then(**function**(result) {

\*\* chg to getAssetRegistry()

# **4.3 Docker Images for IBM HL Fabric Networks**

1. One scenario is the use of pre-built IBM HL Blockchain docker images which have Fabric and a Ledger for BC Peers chaincode endpoints.

# **Appendix**

## [**Coding guidelines**](https://hyperledger-fabric.readthedocs.io/en/latest/Style-guides/go-style.html)

* Coding Golang - We code in Go™ and strictly follow the best practices and will not accept any deviations. You must run the following tools against your Go code and fix all errors and warnings: - golint - go vet – goimports.
* Create a new gitreposity for each Blockchain project.
* Many others…

## **References**

"[IBM Blockchain For Developers](https://developer.ibm.com/courses/all-courses/blockchain-for-developers/)" IBM devworks training course (6 hours credit)

[Configuring your Windows development environment](https://github.com/Microsoft/nodejs-guidelines/blob/master/windows-environment.md#compiling-native-addon-modules)

Samples: <https://github.com/IBM-Blockchain/>

<https://fabric-composer.github.io/reference/commands.html>

<https://chat.hyperledger.org/channel/fabric-composer>  browser web interface for RocketChat  [<https://chat.hyperledger.org/channel/fabric-composer>] - at lot of the contributors and other users are available for live discussion

Our IBM BC Coe Yammer Group :

Sharepoint :

## **Follow on Todos**

|  |  |  |
| --- | --- | --- |
| 1. | Get IBM Bluemix org:space for IBM BC dev | PoCs, sandbox, demos etc  Submitted NSS ticket: INC000018007028  Bluemix space ORG: "COG-EAS-IPM-IBM" Space: "KG-EAS-IPM-IBM-BC" - re-submit request to OneNSS after our Bmix public subscription is renewed. |
| 2. | Deploy and explore with IBM HL Fabric Docker images |  |
| 3. | Updgrade Guide content with HL Fabric V1.0 content | where needed |
| 4. | Need to design and make libraries for framework innovations |  |
| 5. | How to use, deploy fabric composer BNM format with Bluemix Blockchain service? |  |
| 6. | Define a IPM IBM BC github repository |  |
|  |  |  |
|  |  |  |