



BLOCKCHAIN ON LOCAL SETUP INSTRUCTIONS

Abstract

This document covers instructions for setting up Blockchain network on local Windows machine

Contents

Introduction	3
A. Install Docker Toolbox on Windows Machine	3
B. Setup Blockchain (Hyperledger) network using docker images	7
C. Run a Sample Application – Marbles Application	9
D. Test Blockchain Network – If you want to test your chaincode.....	13
Through CLI	13
Through REST API	14

Introduction

This document covers setting up a Blockchain Network (with one peer + membership service) on your local machine for various development and testing activities on Windows machine. It will list the instructions that simply leverages the Docker images that the Hyperledger Fabric project publishes to [DockerHub](#).

Follow the instructions below to setup Blockchain network locally on Windows machine.

Note : Sections A, B and C are mandatory to setup Blockchain Network and get a sample application running. Section D can be referred when you want to develop chaincode and test it on the network

A. Install Docker Toolbox on Windows Machine

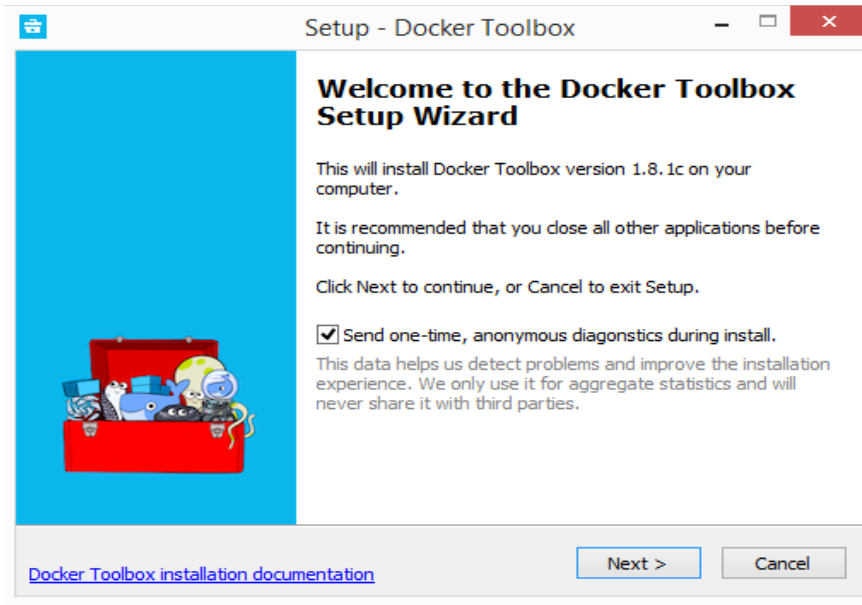
1. To run Docker, your machine must have a 64-bit operating system running Windows 7 or higher. Additionally, you must make sure that virtualization is enabled on your machine.

Follow the steps mentioned on the link (https://docs.docker.com/toolbox/toolbox_install_windows/) to verify the requirements.

2. Go to the [Docker Toolbox](#) page.
3. Click the installer link to download.
4. Install Docker Toolbox by double-clicking the installer.

The installer launches the “Setup - Docker Toolbox” dialog.

If Windows security dialog prompts you to allow the program to make a change, choose **Yes**. The system displays the **Setup - Docker Toolbox for Windows** wizard.

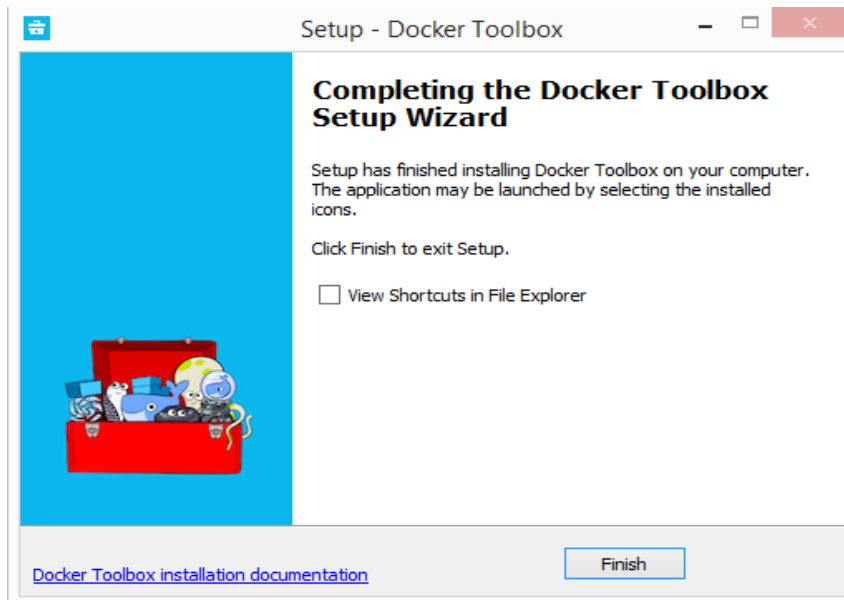


5. Press **Next** to accept all the defaults and then **Install**.

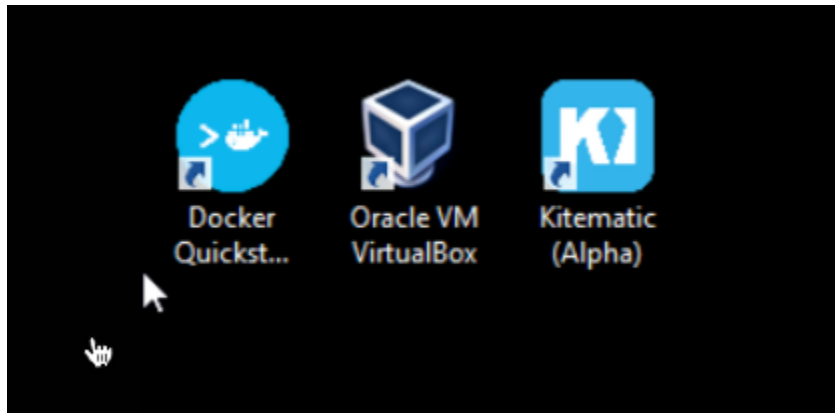
Accept all the installer defaults. The installer takes a few minutes to install all the components:

6. When notified by Windows Security the installer will make changes, make sure you allow the installer to make the necessary changes.

When it completes, the installer reports it was successful:



7. On your Desktop, find the Docker Toolbox icon.



8. Click the icon to launch a Docker Toolbox terminal.

[illegible]

9. Note down IP address of the docker machine printed in the screen as above.

```
ibm_admin@IBM815-PBXCW03 MINGW64 ~  
$ docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
c04b14da8d14: Pull complete  
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cddf9431a1feb60fd9  
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 Hub account:  
https://hub.docker.com  
  
For more examples and ideas, visit:  
https://docs.docker.com/engine/userguide/  
  
ibm_admin@IBM815-PBXCW03 MINGW64 ~  
$
```

B. Setup Blockchain (Hyperledger) network using docker images

1. Execute following commands to pull docker images required to run Blockchain locally

```
docker pull hyperledger/fabric-peer
```

```
docker pull hyperledger/fabric-membersrv
```

```
docker pull hyperledger/fabric-baseimage:x86_64-0.2.1
```

2. Change tag of fabric-baseimage to latest.

This is required due to the issue <https://jira.hyperledger.org/browse/FAB-300> as a workaround

```
docker tag <image-id> hyperledger/fabric-baseimage:latest
```

image-id : image id of baseimage can be found by running command "docker images"

3. Create a directory

```
e.g mkdir blockchain-compose
```

```
cd blockchain-compose
```

4. Create docker-compose.yml file with the below contents

```
membersvc:
```

```
  image: hyperledger/fabric-membersvc
```

```
  ports:
```

```
    - "7054:7054"
```

```
  command: membersvc
```

```
vp0:
```

```
  image: hyperledger/fabric-peer
```

```
  ports:
```

```
    - "7050:7050"
```

```
    - "7051:7051"
```

- "7053:7053"

environment:

- CORE_PEER_ADDRESSAUTODETECT=true
- CORE_VM_ENDPOINT=unix:///var/run/docker.sock
- CORE_LOGGING_LEVEL=DEBUG
- CORE_PEER_ID=vp0
- CORE_PEER_PKI_ECA_PADDR=membersvc:7054
- CORE_PEER_PKI_TCA_PADDR=membersvc:7054
- CORE_PEER_PKI_TLSCA_PADDR=membersvc:7054
- CORE_SECURITY_ENABLED=true
- CORE_SECURITY_ENROLLID=test_vp0
- CORE_SECURITY_ENROLLSECRET=MwYpmSRjupbT

volumes:

- /var/run/docker.sock:/var/run/docker.sock

links:

- membersvc

command: sh -c "sleep 5; peer node start"

5. Run following command to start the Blockchain network with the images

(More information about docker compose can be found at :

<https://docs.docker.com/compose/>)

Docker-compose up

Make sure that to errors or failed messages displayed on the screen.

6. You can check running container by executing following command

docker ps -a

Example output :

```
ibm_admin@IBM815-PRCMB3 MINGW64 ~
$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
000a2a894d89   hyperledger/fabric-peer            "sh -c 'sleep 5; peer"  9 minutes ago  Up 9 minutes  0.0.0.0:7050-7051->7050-7051/tcp, 0.0.0.0:7053->7053/tcp  docker_vp0_1
620afe431ebe   hyperledger/fabric-membersvc      "membersvc"             9 minutes ago  Up 9 minutes  0.0.0.0:7054->7054/tcp  docker_membersvc_1
ibm_admin@IBM815-PRCMB3 MINGW64 ~
```


C. Run a Sample Application – Marbles Application

Refer: <https://github.com/spansare/marbles>

This is a nodejs based application which has been written using Nodejs SDK for Blockchain. This application demonstrates transferring marbles between two users leveraging IBM Blockchain

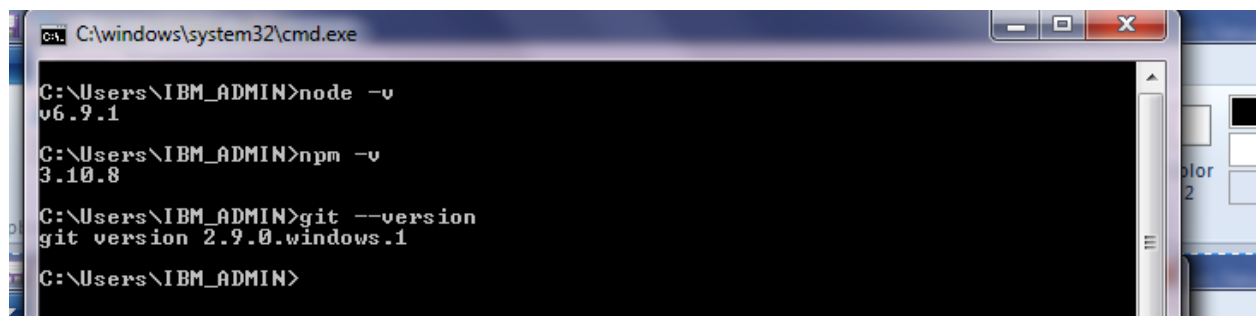
Follow the Steps below to run the application

1. Install Nodejs Runtime

- a. Download the Windows installer from the [Nodes.js® web site](https://nodejs.org/en/)
- b. Run the installer (the .msi file you downloaded in the previous step.)
- c. Follow the prompts in the installer (Accept the license agreement, click the NEXT button a bunch of times and accept the default installation settings).
- d. Restart your computer. You won't be able to run Node.js® until you restart your computer.
- e. Test Node. To see if Node is installed, open the Windows Command Prompt, Powershell or a similar command line tool, and type node -v

2. Install git for windows

- a. Download the Windows installer from <https://git-for-windows.github.io/>
- b. Run the exe
- c. Verify the installation



```
C:\Users\IBM_ADMIN>node -v
v6.9.1

C:\Users\IBM_ADMIN>npm -v
3.10.8

C:\Users\IBM_ADMIN>git --version
git version 2.9.0.windows.1

C:\Users\IBM_ADMIN>
```

3. Clone the Sample Marbles application on the local Windows machine

- a. Open command prompt and create a directory to clone the application

```
mkdir local-marbles
```

```
cd local-marbles
```

- b. Run below command to clone the application code from github

```
git clone https://github.com/spansare/marbles
```

4. Run the application – Execute command below

- a. Change to marbles directory – `cd marbles`
- b. Open `mycreds.json` file. This is the file which will contain all the details about your blockchain network.


Verify the IP Address. It should be the one noted in Section A Step 9.

If not, change it accordingly.

- c. `npm install gulp -g`
- d. `npm install`
- e. `gulp`
- f. If all goes well you should see this message in the console:

```
----- Server Up - localhost:3000 -----
```

- g. The app is already coded to auto deploy the chaincode. You should see further message about it deploying. **[IMPORTANT]** You will need to wait about 60 seconds for the cc to fully deploy. The SDK will do the waiting for us by stalling our callback.
- h. Once you see this message you are good to go:
[ibc-js] Deploying Chaincode - Complete
----- Websocket Up -----



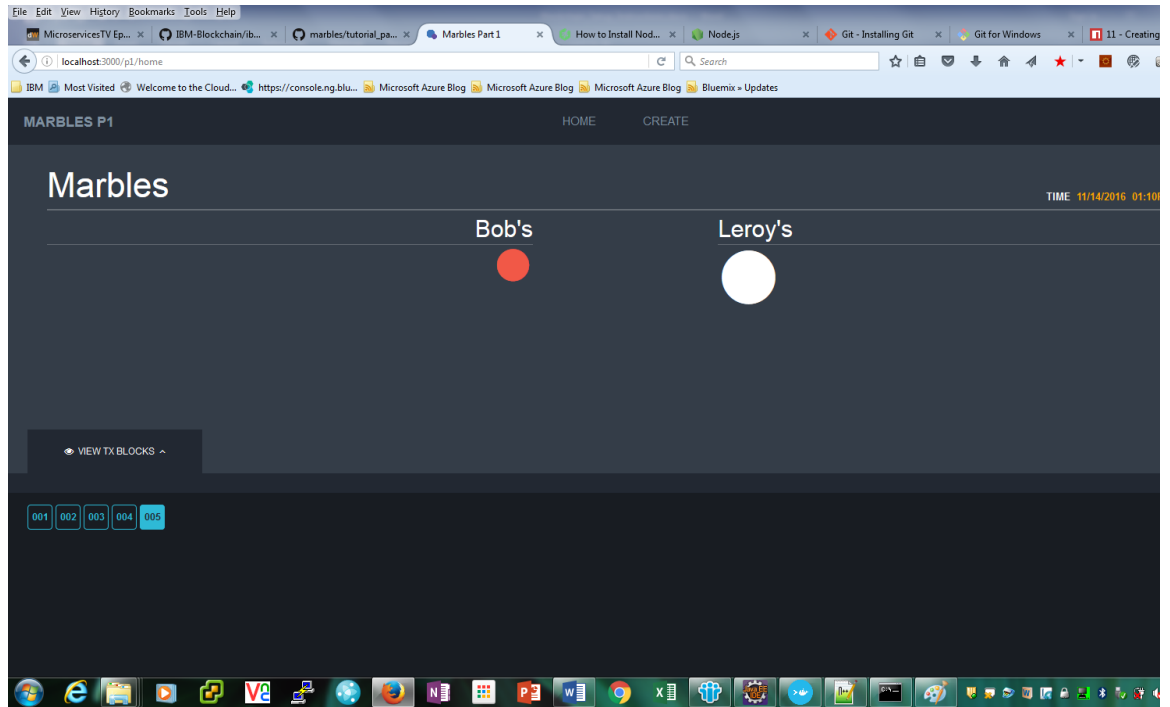
```
cache busting bash.js b6c91c67aac91b8a72d197266ff4b4 css 5c5213ac16de93e881c4aaf7a2eb956b
Running using developer settings
Tracking Deployment
loading hardcoded peers
loading hardcoded users
[ < username: 'alice',
  secret: 'CMS10pEQ1B16',
  enrollId: 'alice',
  enrollSecret: 'CMS10pEQ1B16' > ]
[libc-js] Peer: wp8-up0...:7050
[libc-js] Registering wp8-up0...:7050 w/enrollId - alice
[libc-js] Registration success x1 : alice
[libc-js] removing temp dir
[libc-js] Downloading zip
redirect... https://code.load.github.com/spansare/marbles-chaincode/zip/master
[libc-js] Downloading zip
[libc-js] Unzipping zip
[libc-js] Unzip done
[libc-js] Scanning files [ 'part2_chaincode.go' ]
[libc-js] Parsing file for shin version
[libc-js] Found shin version: github.com/hyperledger/fabric/core/chaincode/shin
[libc-js] Parsing file for invoke functions - part2_chaincode.go
[libc-js] Found cc invoke function: init
[libc-js] Found cc invoke function: delete
[libc-js] Found cc invoke function: write
[libc-js] Found cc invoke function: init_marble
[libc-js] Found cc invoke function: set_user
[libc-js] Found cc invoke function: open_trade
[libc-js] Found cc invoke function: perform_trade
[libc-js] Found cc invoke function: remove_trade
[libc-js] Parsing file for query functions - part2_chaincode.go
[libc-js] Found cc query function: read
[libc-js] load_chaincode() finished
[libc-js] Deploy Chaincode - Starting
[libc-js] function: init , arg: [ '99' ]

Waiting...

deploy success (waiting another 50 seconds)
95c0caa50f5441d0684b8ca6f9c641a53c1db78af11c278e0c60b1c519ffba42637176c1af782ea99e9b03e2916fb48096e610865decc4b142238991de33964
[libc-js] libc.save() warning < Error: ENOENT: no such file or directory, open 'C:\Users\IBM_ADMIN\local-nb\marbles\cc_summaries\95c0caa50f5441d0684b8ca6f9c641a53c1db78af11c278e0c60b1c519ffba42637176c1af782ea99e9b03e2916fb48096e610865decc4b142238991de33964' >
at Error (native)
errno: -4058,
code: 'ENOENT',
syscall: 'open',
path: 'C:\Users\IBM_ADMIN\local-nb\marbles\cc_summaries\95c0caa50f5441d0684b8ca6f9c641a53c1db78af11c278e0c60b1c519ffba42637176c1af782ea99e9b03e2916fb48096e610865decc4b142238991de33964'
[libc-js] Deploy Chaincode - Complete
[preflight check] 1 : testing if chaincode is ready
[libc-js] read - success: { jsonrpc: '2.0',
  result: { status: 'OK', message: 'null' },
  id: 1479128827604 }
[preflight check] 1 : success
----- Websocket Up -----
[libc-js] Chain Stats - success
[libc-js] New block! 2
hey new block, lets refresh and broadcast to all 1
[libc-js] Block Stats 1 - success
[libc-js] read - success: { jsonrpc: '2.0',
  result: { status: 'OK', message: 'null' },
  id: 1479128828691 }
[libc-js] read - success: { jsonrpc: '2.0',
  result: { status: 'OK', message: '{"open_trades":null}' },
  id: 1479128828692 }
```

5. Use the application

- Open up your browser and browse to <http://localhost:3000>
- Click the "Create" link on the top navigation bar
- Fill out all the fields, then click the "Create" button
- It should have flipped you back to "Home" and you should see that a new marble has been created
- Next let's trade a marble. Click and drag one marble from one person's list to another. It should temporarily disappear and then auto reload the marbles in their new state.
- You can view all the transactions at the bottom of the application page



You have the sample application ready to use.

Note : Next Section is optional. If you have developed your own chaincode and want to test that on the network, you can refer next section, else it can be skipped

D. Test Blockchain Network – If you want to test your chaincode

There are 2 ways to test the network as listed below. You can use either of them

Through CLI

- Open another Quickstart Docker Terminal.

Login to peer container

Docker exec -ti <container-id> bash

container-id : container id of the peer can be found by running “docker ps -a” command.

- Execute following command to login into the network with user ‘test_user0’

Note : Users are pre-created for the network and their details can be found in file membersrvcl.yaml in membersrvcl container.

peer network login test_user0 -p MS9qrN8hFjIE

```
tm_admin@170815-P88CH03-MINCH64 ~
$ docker exec -ti 000a2a894d89 hash
root@000a2a894d89:/opt/gopath/src/github.com/hyperledger/fabric# peer network login test_user0 -p MS9qrN8hFjIE
07:28:46.955 [logging] LoggingInit -> DEBUG 001 Setting default logging level to DEBUG for command 'network'
07:28:46.956 [networkCmd] networkLogin -> INFO 002 CLI client login...
07:28:46.956 [networkCmd] networkLogin -> INFO 003 Local data store for client loginToken: /var/hyperledger/production/client/
07:28:46.957 [networkCmd] networkLogin -> INFO 004 Logging in user 'test_user0' on CLI interface...
07:28:47.063 [networkCmd] networkLogin -> INFO 005 Storing login token for user 'test_user0'.
07:28:47.064 [networkCmd] networkLogin -> INFO 006 Login successful for user 'test_user0'.
07:28:47.064 [main] main -> INFO 007 Exiting.....
peer chaincode deploy -u test_user0 -p github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02 -c '{"Args":["init","a","100","b","200"]}'
```

- Execute following command to deploy chaincode

peer chaincode deploy -u test_user0 -p

github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02 -c
'{"Args":["init","a","100","b","200"]}'

github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02 :
Sample chaincode which can be used to test the blockchain network

```
peer chaincode deploy -u test_user0 -p github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02 -c '{"Args":["init","a","100","b","200"]}'
07:29:10.241 [logging] LoggingInit -> DEBUG 001 Setting default logging level to DEBUG for command 'chaincode'
07:29:10.242 [chaincodeCmd] getChaincodeSpecification -> INFO 002 Local user 'test_user0' is already logged in. Retrieving login token.
07:29:13.522 [chaincodeCmd] chaincodeDeploy -> INFO 003 Deploy result: type:GOLANG chaincodeID:<path:github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02 name:"ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2203ed93401d7d05eb0dd194fb7070549c5dc31eb63f4e654dbd5a1d86cbb30e48e3ab1812590cd0f78539"
07:29:13.524 [main] main -> INFO 004 Exiting.....
root@000a2a894d89:/opt/gopath/src/github.com/hyperledger/fabric# exit
```

It will return hashcode for the deployed chaincode.

This process takes sometime even though command is completed.

Note : To verify chaincode deploy was successful, execute “docker ps -a” command (from Quickstart Docker terminal) and check if one more container is added to the list and is in running state.

Also, make sure there are no errors thrown on the terminal where “docker-compose up” command was initiated.

```
ilm_admin@IDB815-PRCMA3 MINGW64 ~
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
6608147a66a5       dev-vp0-ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed93401d7d05eb0dd194fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539  "/opt/gopath/bin/ee5b"  29 minu
000a2a894d89       hyperledger/fabric-peer  "sh -c 'sleep 5; peer"  51 minu
620afe431e8e       hyperledger/fabric-membervc  "membersvc"  51 minu
```

- d. Execute following command to invoke chaincode

```
peer chaincode invoke -u test_user0 -n
ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed93401d7d05eb0dd19
4fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539 -c
'{"Args": ["invoke", "a", "b", "10"]}'
```

```
$ docker exec -ti 000a2a894d89 bash
<22e39ddba12e5d2e203ed93401d7d05eb0dd194fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539 -c '{"Args": ["invoke", "a", "b", "10"]}'
08:04:37.781 [logging] LoggingInit -> DEBU 001 Setting default logging level to DEBUG for command 'chaincode'
08:04:37.782 [chaincodeCmd] getChaincodeSpecification -> INFO 002 Local user 'test_user0' is already logged in. Retrieving login token.
08:04:38.148 [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 003 Successfully invoked transaction: chaincodeSpec:(type:GOLANG chaincodeID:(name:"ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed
08:04:38.151 [main] main -> INFO 004 Exiting.....
```

- e. Execute following command to query chaincode

```
peer chaincode query -u test_user0 -n
ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed93401d7d05eb0dd19
4fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539 -c
'{"Args": ["query", "a"]}'
```

```
<22e39ddba12e5d2e203ed93401d7d05eb0dd194fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539 -c '{"Args": ["query", "a"]}'
08:05:02.891 [logging] LoggingInit -> DEBU 001 Setting default logging level to DEBUG for command 'chaincode'
08:05:02.892 [chaincodeCmd] getChaincodeSpecification -> INFO 002 Local user 'test_user0' is already logged in. Retrieving login token.
08:05:03.359 [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 003 Successfully queried transaction: chaincodeSpec:(type:GOLANG chaincodeID:(name:"ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203e
Query Result: 90
08:05:03.361 [main] main -> INFO 004 Exiting.....
root@000a2a894d89:/opt/gopath/src/github.com/hyperledger/fabric#
```

Through REST API

- a. Use the IP address printed on Quickstart Docker Terminal (as mentioned in Section A Step 9) as REST API Endpoint for REST API invocation.
- b. Use any REST Client (e.g. REST Client for Firefox or Postman for Chrome)
- c. Use the GET call below to make sure REST API endpoint is configured correctly.

<http://<IP Address>:7050/network/peers>

The screenshot shows a REST client interface. At the top, the Method is set to GET and the URL is http://192.168.99.100:7050/network/peers. Below this, there is a section for the Request Body, which is currently empty. The main section is titled '[-] Response' and contains four tabs: 'Response Headers', 'Response Body (Raw)', 'Response Body (Highlight)', and 'Response Body (Preview)'. The 'Response Body (Raw)' tab is selected, displaying the following JSON response:

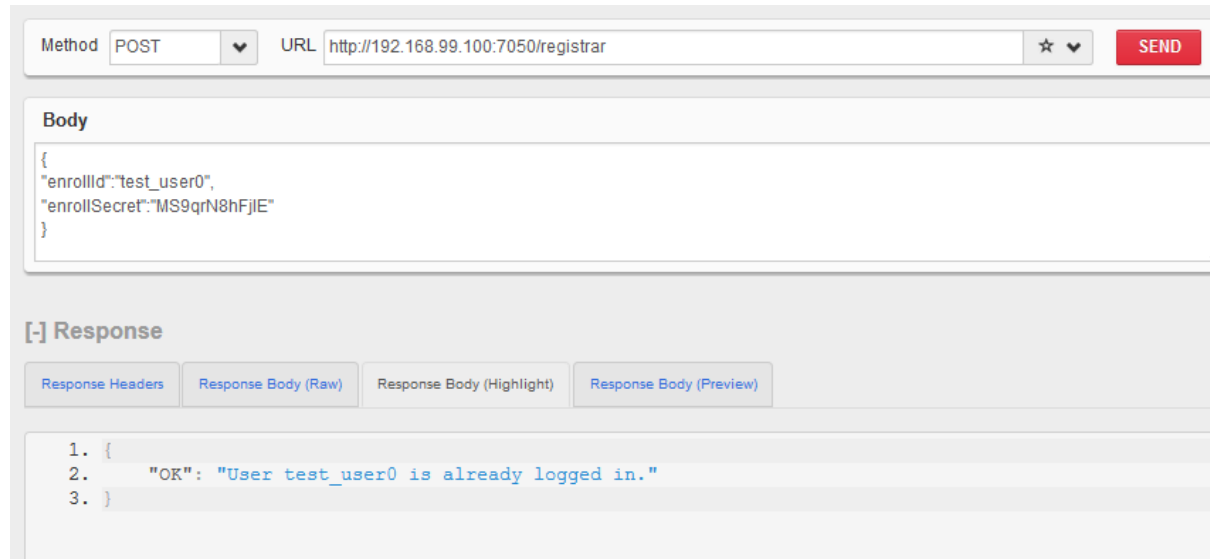
```
1. {
2.   "peers":
3.   [
4.     {
5.       "ID":
6.       {
7.         "name": "vp0"
8.       },
9.       "address": "172.17.0.3:7051",
10.      "type": 1,
11.      "pkiID": "k9o8D/VQTIO/UXBpRexQAAjp/mLPLdY4KcMg16zeiBc="
12.    }
13.  ]
14. }
```

d. Use POST call to register the User

<http://<IP Address>:7050/registrar>

with Body as:

```
{
  "enrollId": "test_user0",
  "enrollSecret": "MS9qrN8hFjIE"
}
```



e. Deploy Chaincode with REST API

<http://<IP Address>:7050/chaincode>

With Body as:

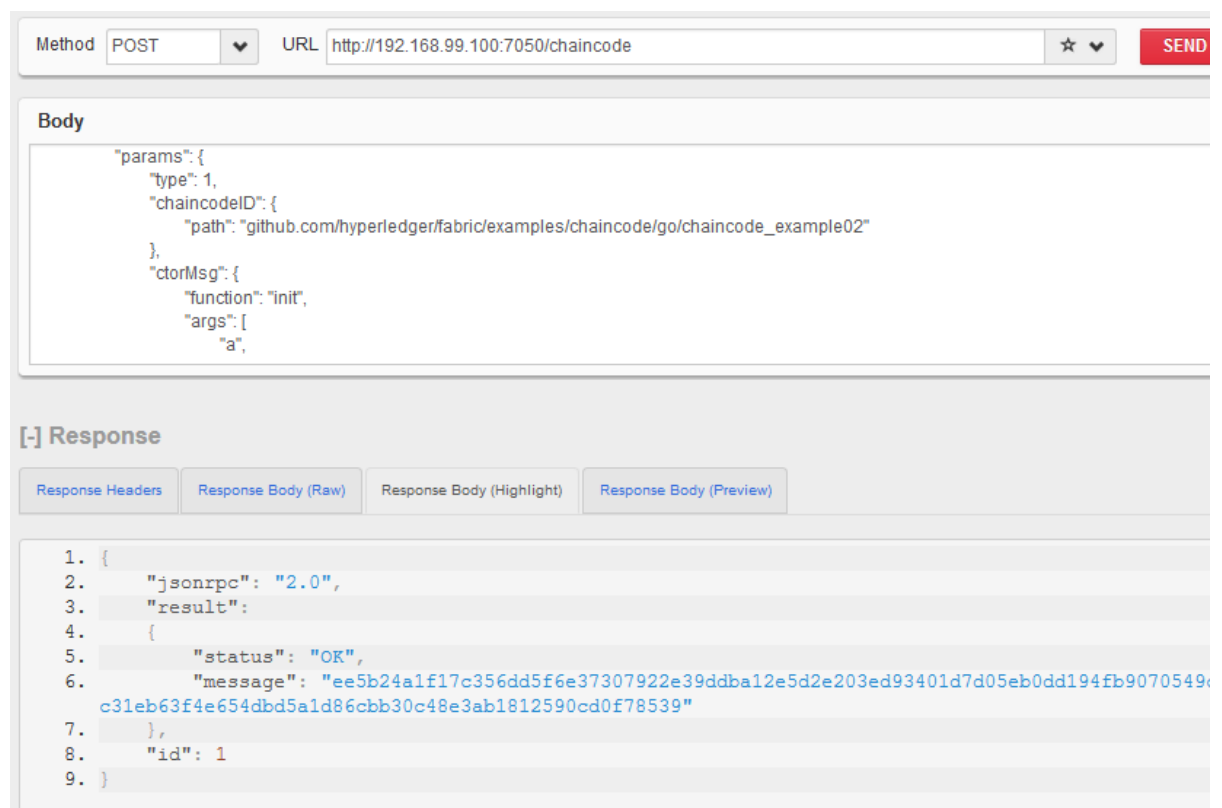
```
{
  "jsonrpc": "2.0",
  "method": "deploy",
  "params": {
    "type": 1,
    "chaincodeID": {
      "path":
        "github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02"
    },
    "ctorMsg": {
      "function": "init",
      "args": [
        "a",
        "100",
```



```

    "b",
    "200"
  ]
},
"secureContext": "test_user0"
},
"id": 1
}

```



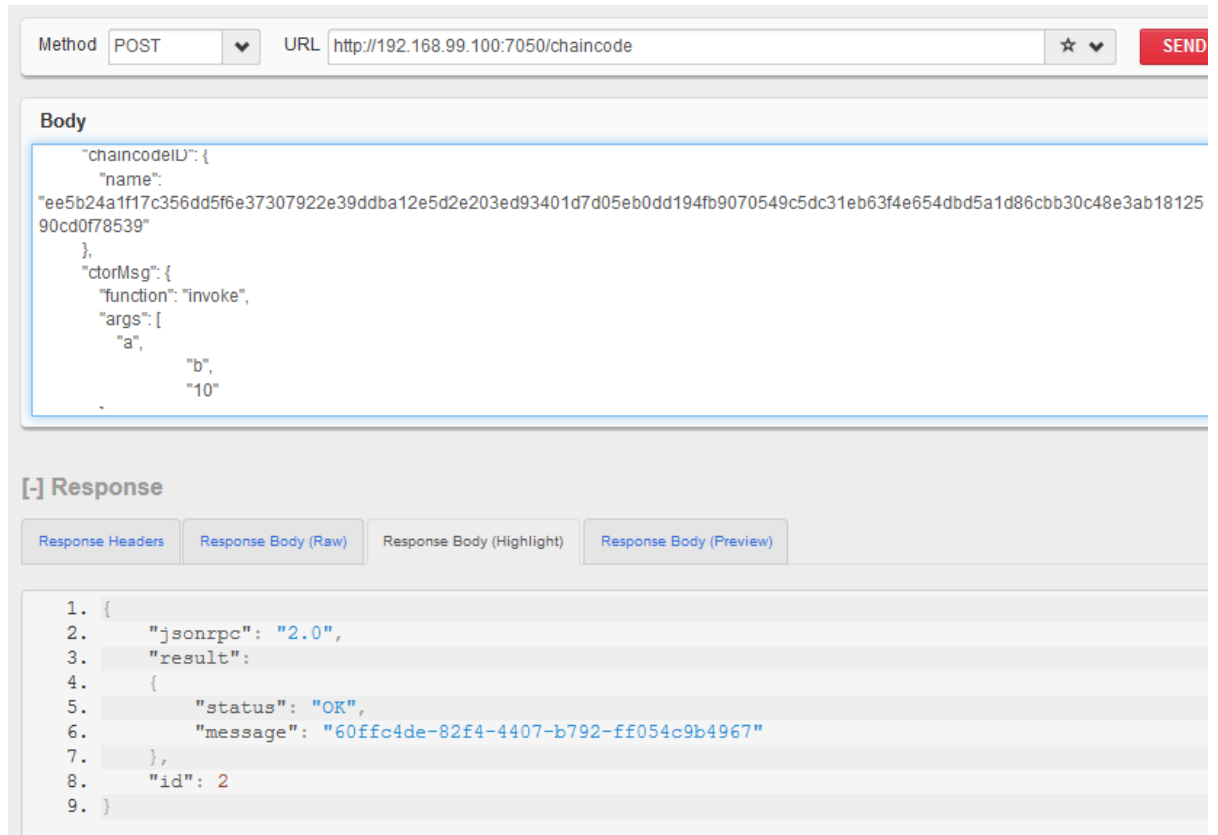
f. Invoke Chaincode with REST API

<http://<IP Address>:7050/chaincode>

With Body as :

```
{
```

```
"jsonrpc": "2.0",
"method": "invoke",
"params": {
  "type": 1,
  "chaincodeID": {
    "name":
"ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed93401d7d05eb0dd19
4fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539"
  },
  "ctorMsg": {
    "function": "invoke",
    "args": [
      "a",
      "b",
      "10"
    ]
  },
  "secureContext": "test_user0"
},
"id": 2
}
```



g. Query Chaincode with REST API

<http://192.168.99.100:7050/chaincode>

With Body as:

```
{
  "jsonrpc": "2.0",
  "method": "query",
  "params": {
    "type": 1,
    "chaincodeID": {
      "name": "ee5b24a1f17c356dd5f6e37307922e39ddba12e5d2e203ed93401d7d05eb0dd194fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539"
    },
    "ctorMsg": {
      "function": "read",

```

```
"args": [  
  "a"  
],  
,  
"secureContext": "test_user0"  
,  
"id": 2  
}
```

Method POST URL http://192.168.99.100:7050/chaincode

Body

```
{  
  "type": 1,  
  "chaincodeID": {  
    "name":  
    "ee5b24a1f17c356dd5f6e37307922e39ddbba12e5d2e203ed93401d7d05eb0dd194fb9070549c5dc31eb63f4e654dbd5a1d86cbb30c48e3ab1812590cd0f78539"  
  },  
  "ctorMsg": {  
    "function": "query",  
    "args": [  
      "a"  
    ]  
  }  
}
```

[-] Response

Response Headers Response Body (Raw) Response Body (Highlight) Response Body (Preview)

```
1. {  
2.   "jsonrpc": "2.0",  
3.   "result":  
4.     {  
5.       "status": "OK",  
6.       "message": "90"  
7.     },  
8.   "id": 2  
9. }
```