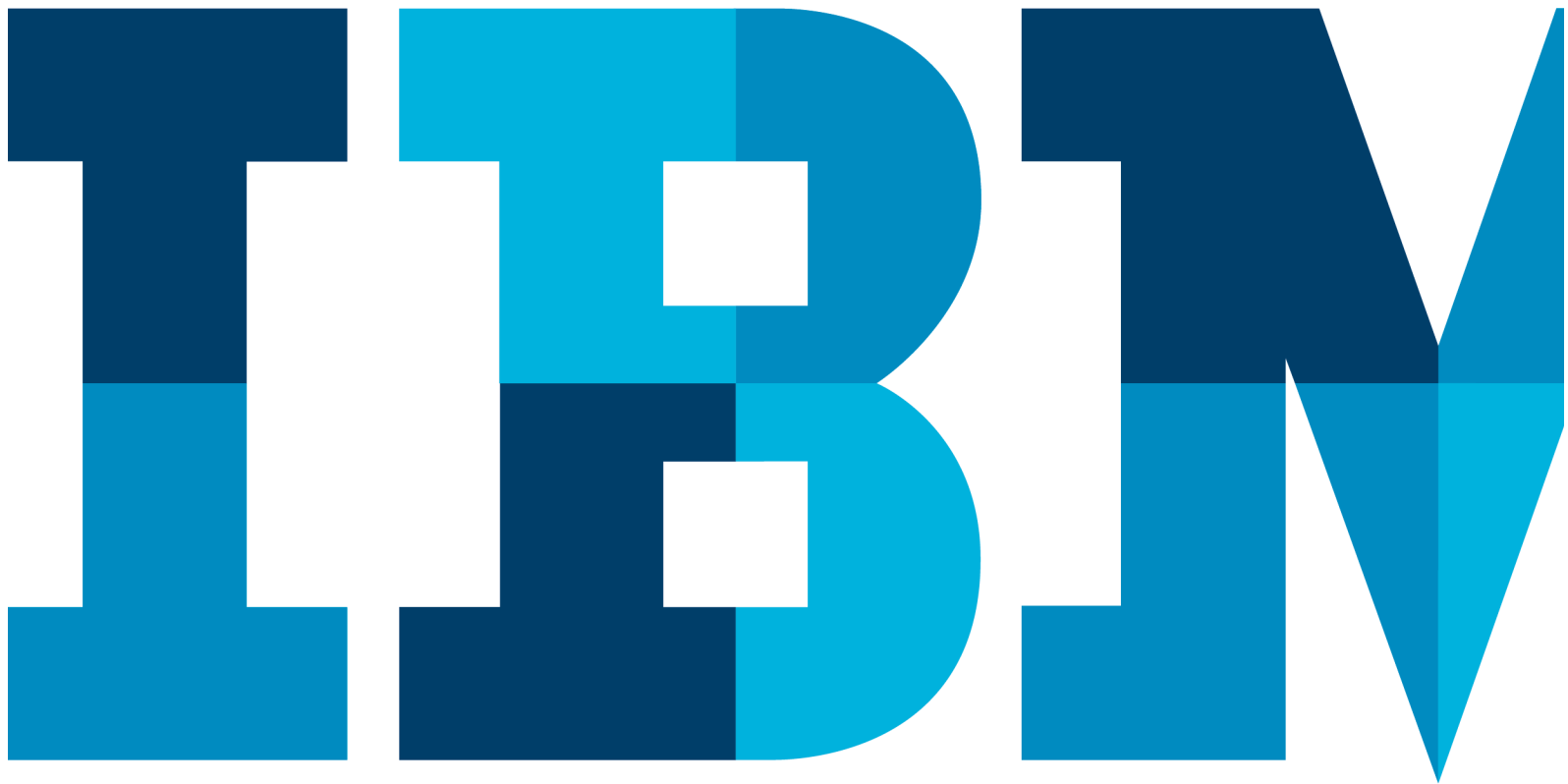


IBM Blockchain Hands-On Blockchain Unchained

Lab Three – VM – Exercises



Contents

SECTION 1.	REVIEWING THE CHAINCODE	4
SECTION 2.	ACCESSING THE VAGRANT ENVIRONMENT.....	10
SECTION 3.	DEPLOYING THE CHAINCODE.....	12
SECTION 4.	INVOKING THE CHAINCODE	17
SECTION 5.	QUERYING CHAINCODE.....	21
APPENDIX A.	NOTICES.....	22
APPENDIX B.	TRADEMARKS AND COPYRIGHTS.....	24

Overview

The aim of this lab is to introduce you to chaincode development by showing how to deploy your first chaincode to a new blockchain service in a virtual machine environment.

We will use a sample piece of chaincode (Smart Contract) called **example02** as the foundation for our lab. This sample is provided as part of the Hyperledger Fabric code.

Once deployed, the chaincode can be invoked and queried.

Introduction

Pre-requisites:

- 4 cores
- 4GB RAM
- VMWare V10+
- The lab virtual machine (IBM_Hyperledger_Car_Leasing_Demo_v0.8)

The virtual machine is based on Linux Ubuntu 14.04 and contains Hyperledger Fabric V0.6, Golang, Git, Vagrant, Visual Studio Code with the “Encode Decode” extension and Firefox.

A network needs to be visible to the virtual machine (even if the network is just to the host environment). If you do not see the up/down arrows in the status bar at the top of the screen, or if you receive errors about no network being available, please tell the lab leader. The virtual machine might need to be reconfigured in NAT mode.

There are no additional files or software that is proprietary to the lab in the virtual machine. This means that the lab may be run on a machine without the without a lab virtual machine if Hyperledger Fabric and the other pre-requisites have been installed.

It is recommended that students have previously completed the Blockchain Explained and Blockchain Explored labs.

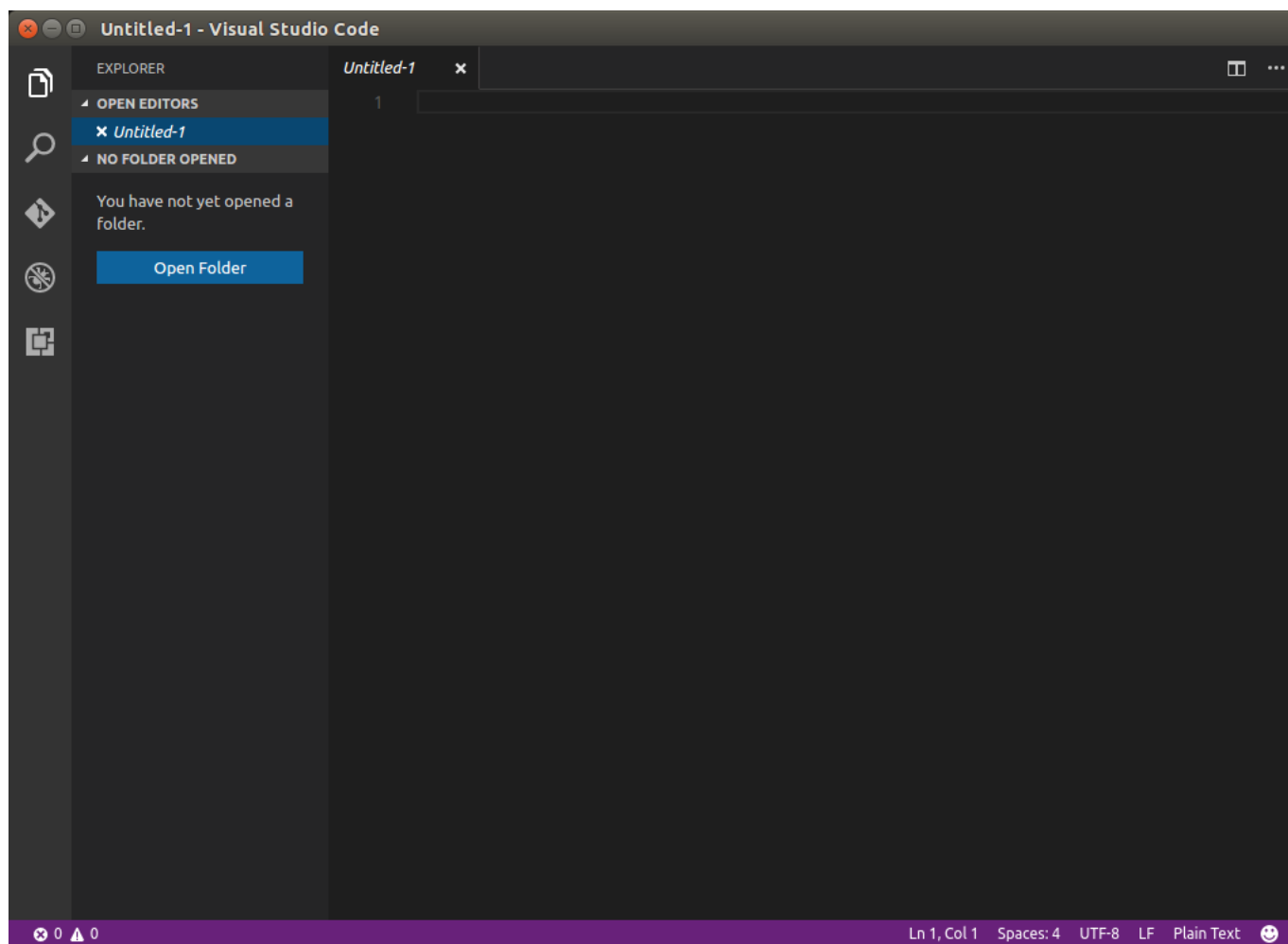
Section 1. Reviewing the chaincode

We will first locate the source of the example02 chaincode and review it.

__1. To start the Visual Studio Code IDE, **left-click** the icon:



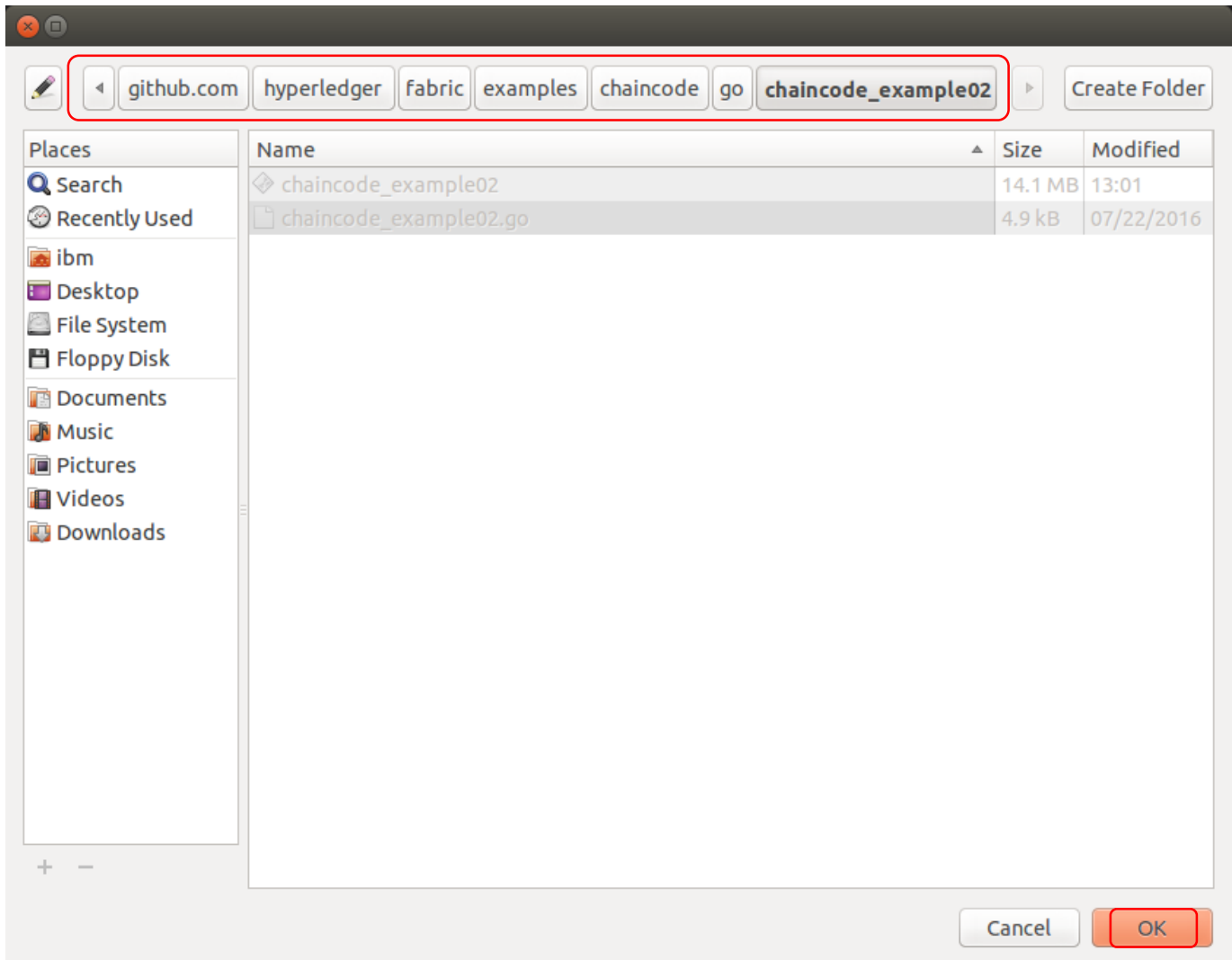
__2. The Visual Studio Code editor is loaded.



__3. The **example02** chaincode can be found in the following directory:

```
/home/ibm/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric  
/examples/chaincode/go/chaincode_example02
```

Within the editor, select **File > Open Folder** from the menu and navigate to directory above, and select **OK**.



- ___4. Using the Explorer pane to the left of the editor, **left-click** the file **chaincode_example02** (source file). This will open the file in the editor window to view.

Every chaincode deployed to Hyperledger Fabric must adhere to the shim (github.com/hyperledger/fabric/core/chaincode/shim) which defines the interface.

Upon initialization, example02 creates two key/value pairs in the world-state. Each key is an identifier string (e.g. "A") whose value is an associated integer balance (e.g. 100). Transactions that invoke this chaincode will increment the balance of one identifier while decrementing the other.

For example:

Initializing the chaincode with ["a", "100", "b", "200"] will set up "a" to be "100" and "b" to be "200".
Invoking a transaction with ["a", "b", "10"] will decrement the value of "a" by 10 and increment "b" by the same amount.
Querying the value of "a" and "b" at this point will yield "a" to have the value "90" and "b" to have "210".

```

1  /*
2  Copyright IBM Corp. 2016 All Rights Reserved.
3
4  Licensed under the Apache License, Version 2.0 (the "License");
5  you may not use this file except in compliance with the License.
6  You may obtain a copy of the License at
7
8      http://www.apache.org/licenses/LICENSE-2.0
9
10 Unless required by applicable law or agreed to in writing, software
11 distributed under the License is distributed on an "AS IS" BASIS,
12 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 See the License for the specific language governing permissions and
14 limitations under the License.
15 */
16
17 package main
18
19 //WARNING - this chaincode's ID is hard-coded in chaincode_example04 to illustrate
20 //calling chaincode from a chaincode. If this example is modified, chaincode_example
21 //to be modified as well with the new ID of chaincode_example02.
22 //chaincode_example05 show's how chaincode ID can be passed in as a parameter instea
23 //hard-coding.
24
25 import (
26     "errors"
27     "fmt"
28     "strconv"
29
30     "github.com/hyperledger/fabric/core/chaincode/shim"
31 )
32
33 // SimpleChaincode example simple Chaincode implementation
34 type SimpleChaincode struct {
35

```

- ___5. Review the `Init()` method of the chaincode. The `Init()` method is called once on deploying the chaincode. The deployment of the chaincode will be recorded as a transaction in the blockchain.

```

36
37 func (t *SimpleChaincode) Init(stub *shim.ChaincodeStub, function string, args []string) ([]byte, error) {
38     var A, B string // Entities
39     var Aval, Bval int // Asset holdings
40     var err error
41
42     if len(args) != 4 {
43         return nil, errors.New("Incorrect number of arguments. Expecting 4")
44     }
45
46     // Initialize the chaincode
47     A = args[0]
48     Aval, err = strconv.Atoi(args[1])
49     if err != nil {
50         return nil, errors.New("Expecting integer value for asset holding")
51     }
52     B = args[2]
53     Bval, err = strconv.Atoi(args[3])
54     if err != nil {
55         return nil, errors.New("Expecting integer value for asset holding")
56     }
57     fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)
58
59     // Write the state to the ledger
60     err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
61     if err != nil {
62         return nil, err
63     }
64
65     err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
66     if err != nil {
67         return nil, err
68     }
69
70     return nil, nil

```

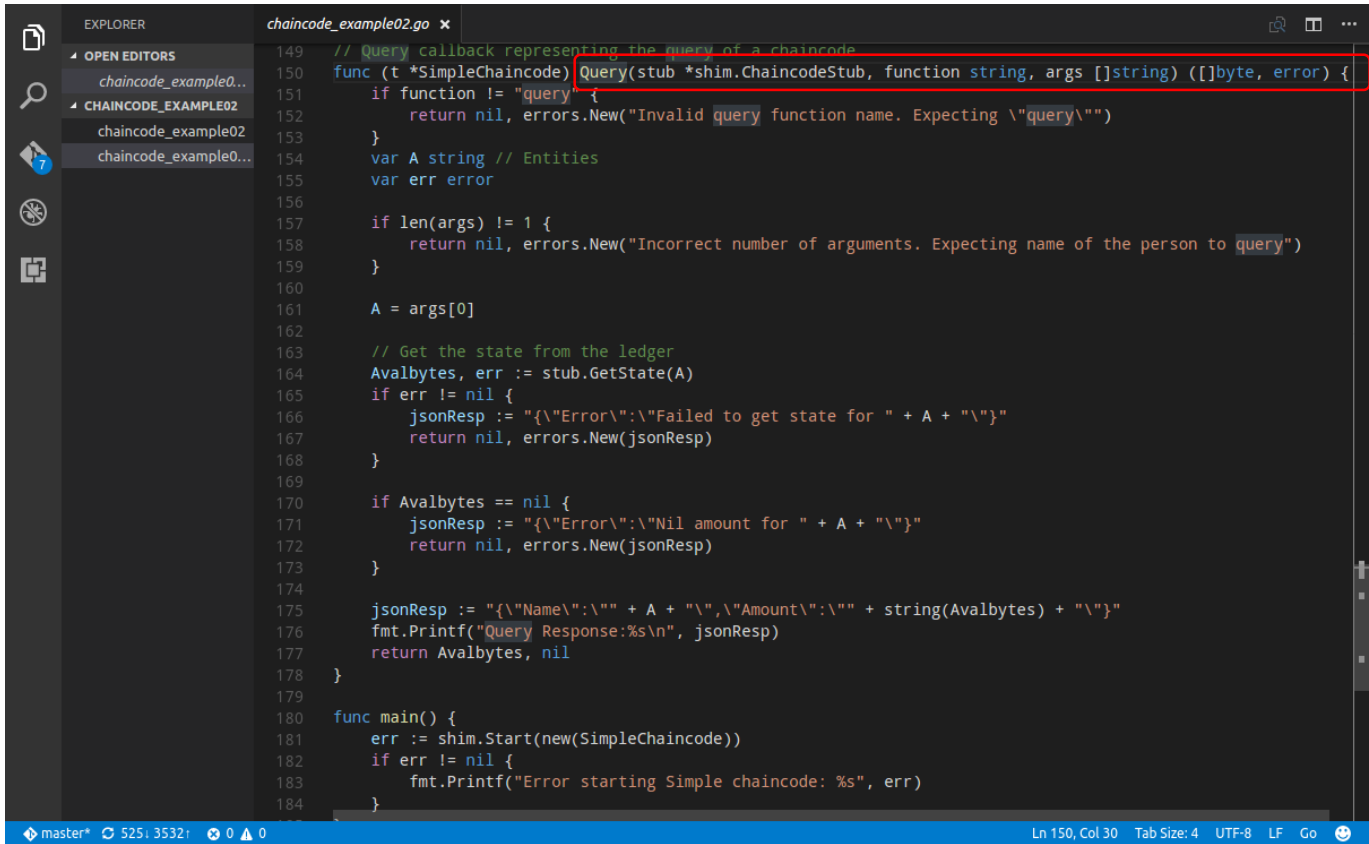
- ___6. Review the **Invoke()** method of the chaincode. This is called everytime the chaincode is “invoked”. The call to **Invoke()** will be recorded as a transaction in the blockchain. An invoke normally updates content of the world-state.

```

73 // Transaction makes payment of X units from A to B
74 func (t *SimpleChaincode) Invoke(stub *shim.ChaincodeStub, function string, args []string) ([]byte, error) {
75     if function == "delete" {
76         // Deletes an entity from its state
77         return t.delete(stub, args)
78     }
79
80     var A, B string // Entities
81     var Aval, Bval int // Asset holdings
82     var X int // Transaction value
83     var err error
84
85     if len(args) != 3 {
86         return nil, errors.New("Incorrect number of arguments. Expecting 3")
87     }
88
89     A = args[0]
90     B = args[1]
91
92     // Get the state from the ledger
93     // TODO: will be nice to have a GetAllState call to ledger
94     Avalbytes, err := stub.GetState(A)
95     if err != nil {
96         return nil, errors.New("Failed to get state")
97     }
98     if Avalbytes == nil {
99         return nil, errors.New("Entity not found")
100     }
101     Aval, _ = strconv.Atoi(string(Avalbytes))
102
103     Bvalbytes, err := stub.GetState(B)
104     if err != nil {
105         return nil, errors.New("Failed to get state")
106     }
107     if Bvalbytes == nil {
108         return nil, errors.New("Entity not found")

```

7. Review the **Query()** method of the chaincode. This is called everytime the chaincode is “queried”. The call to **Query()** will not be recorded as a transaction in the blockchain. A query is a read-only transaction and therefore cannot update content in the world-state.



```
149 // Query callback representing the query of a chaincode
150 func (t *SimpleChaincode) Query(stub *shim.ChaincodeStub, function string, args []string) ([]byte, error) {
151     if function != "query" {
152         return nil, errors.New("Invalid query function name. Expecting \"query\"")
153     }
154     var A string // Entities
155     var err error
156
157     if len(args) != 1 {
158         return nil, errors.New("Incorrect number of arguments. Expecting name of the person to query")
159     }
160
161     A = args[0]
162
163     // Get the state from the ledger
164     Avalbytes, err := stub.GetState(A)
165     if err != nil {
166         jsonResp := "{\"Error\":\"Failed to get state for " + A + "\"}"
167         return nil, errors.New(jsonResp)
168     }
169
170     if Avalbytes == nil {
171         jsonResp := "{\"Error\":\"Nil amount for " + A + "\"}"
172         return nil, errors.New(jsonResp)
173     }
174
175     jsonResp := "{\"Name\":\"" + A + "\",\"Amount\":\"" + string(Avalbytes) + "\"}"
176     fmt.Printf("Query Response:%s\n", jsonResp)
177     return Avalbytes, nil
178 }
179
180 func main() {
181     err := shim.Start(new(SimpleChaincode))
182     if err != nil {
183         fmt.Printf("Error starting Simple chaincode: %s", err)
184     }
185 }
```

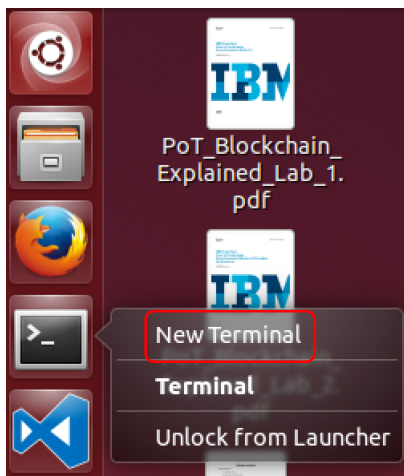
We will now deploy and run this chaincode.

Section 2. Accessing the Vagrant Environment

Vagrant is a virtual OS *inside* the lab virtual machine that is used to compile and run the Hyperledger Fabric. Having a nested virtualisation environment helps ensure consistency in the runtime environment, allowing you to see similar compilation and runtime results whether you are developing on MacOS, Windows, Linux or whatever. The Vagrant VM is based on Ubuntu Linux.

In this section you will create a shell command prompt in Vagrant which is where the Hyperledger Fabric peers are running. Once in this environment you will be able to issue **peer** commands to deploy chaincode (Smart Contracts).

- __1. Start a new terminal window. **Right-click** on the terminal icon, and select “**New Terminal**”.



- __2. As Hyperledger Fabric is running in a Vagrant environment, create a shell command prompt.

Vagrant has been started automatically in the Ubuntu VM. In order to run the fabric **peer** commands, a Vagrant shell command prompt needs to be created.

From the command line in the new terminal, change to the following directory directory which has the Vagrant image file, by entering:

```
cd Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv
```

- __3. Start the Vagrant shell by entering: **vagrant ssh**

```
ibm@ubuntu: ~/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv
ibm@ubuntu:~$ cd Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv/
ibm@ubuntu:~/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv$ vagrant ssh
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-86-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Tue Oct 25 11:49:50 UTC 2016

System load:  0.16           Processes:           95
Usage of /:   11.8% of 38.75GB Users logged in:        0
Memory usage: 6%           IP address for eth0: 10.0.2.15
Swap usage:   0%           IP address for docker0: 172.17.0.1

Graph this data and manage this system at:
https://landscape.canonical.com/

New release '16.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue Oct 25 11:49:51 2016 from 10.0.2.2
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:~$
```

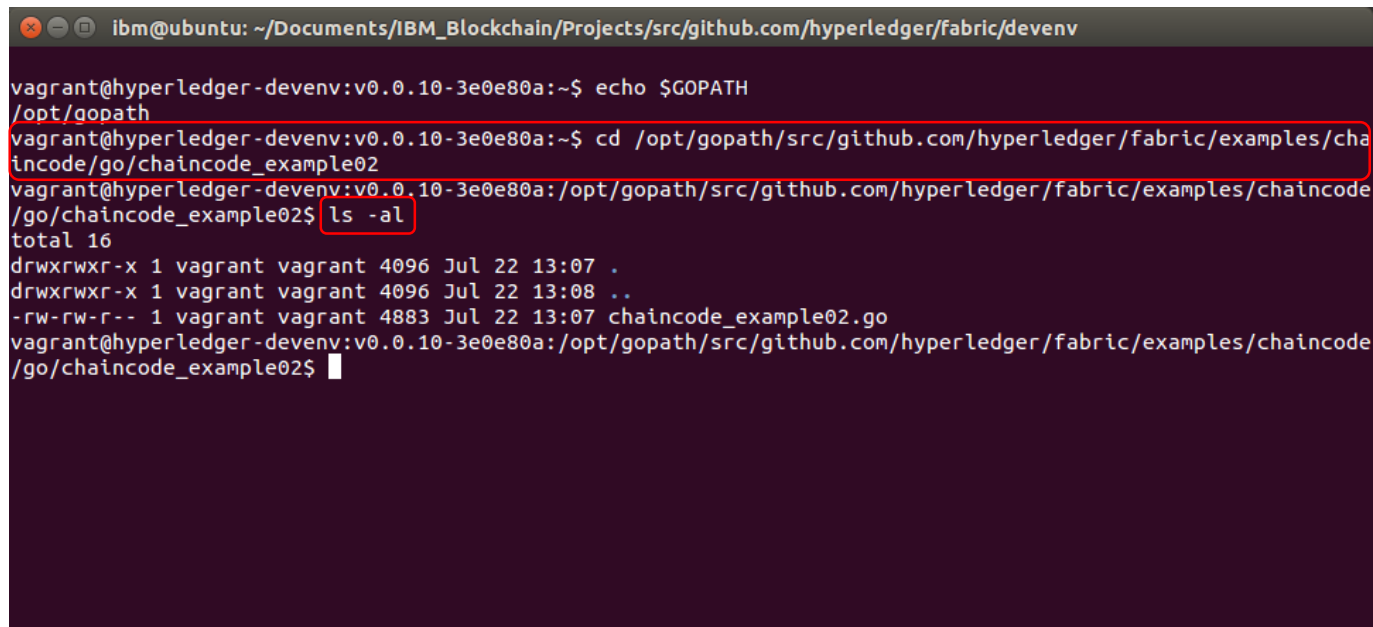
Section 3. Deploying the Chaincode

We will now deploy the chaincode from within the Vagrant environment previously started.

__1. Locate the example chaincode to deploy

The environment `$GOPATH` locates the root of the Hyperledger Fabric code, and chaincode must exist as a subdirectory of this root to deploy using the `peer` command. Change the directory to locate the **example02** chaincode, and list the contents of the directory by entering:

```
cd $GOPATH/src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
ls -al
```

A terminal window titled 'ibm@ubuntu: ~/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv'. The prompt is 'vagrant@hyperledger-devenv:v0.0.10-3e0e80a:~\$'. The user enters 'echo \$GOPATH' and the output is '/opt/gopath'. Then, the user enters 'cd /opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02' and the prompt changes to 'vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02\$'. Finally, the user enters 'ls -al' and the output is: 'total 16', 'drwxrwxr-x 1 vagrant vagrant 4096 Jul 22 13:07 .', 'drwxrwxr-x 1 vagrant vagrant 4096 Jul 22 13:08 ..', '-rw-rw-r-- 1 vagrant vagrant 4883 Jul 22 13:07 chaincode_example02.go'. The prompt returns to 'vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02\$'.

__2. Compile example02.

It is good practice to compile chaincode before deployment. This is not strictly necessary as the chaincode source is deployed to the blockchain, but doing so helps eliminate any errors as early as possible.

The environment is set up to include all pre-requisites for building chaincode (Hyperledger Fabric, shim, Golang compiler, RocksDB etc).

Build **example02** by entering the following commands (there should be no errors):

```
go build
ls -al
```

```

ibm@ubuntu: ~/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$ go build
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$ ls -al
total 13796
drwxrwxr-x 1 vagrant vagrant    4096 Oct 25 12:01 .
drwxrwxr-x 1 vagrant vagrant    4096 Jul 22 13:08 ..
-rwxrwxr-x 1 vagrant vagrant 14110112 Oct 25 12:01 chaincode_example02
-rw-rw-r-- 1 vagrant vagrant    4883 Jul 22 13:07 chaincode_example02.go
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$

```

Note: The output is the compiled file **chaincode_example02**.

Before the chaincode can be deployed, the user (Administrator) must first login to the Fabric peer process, which is already running.

A set of users are configured, and these can be found in the **membersrvcl.yaml** file.

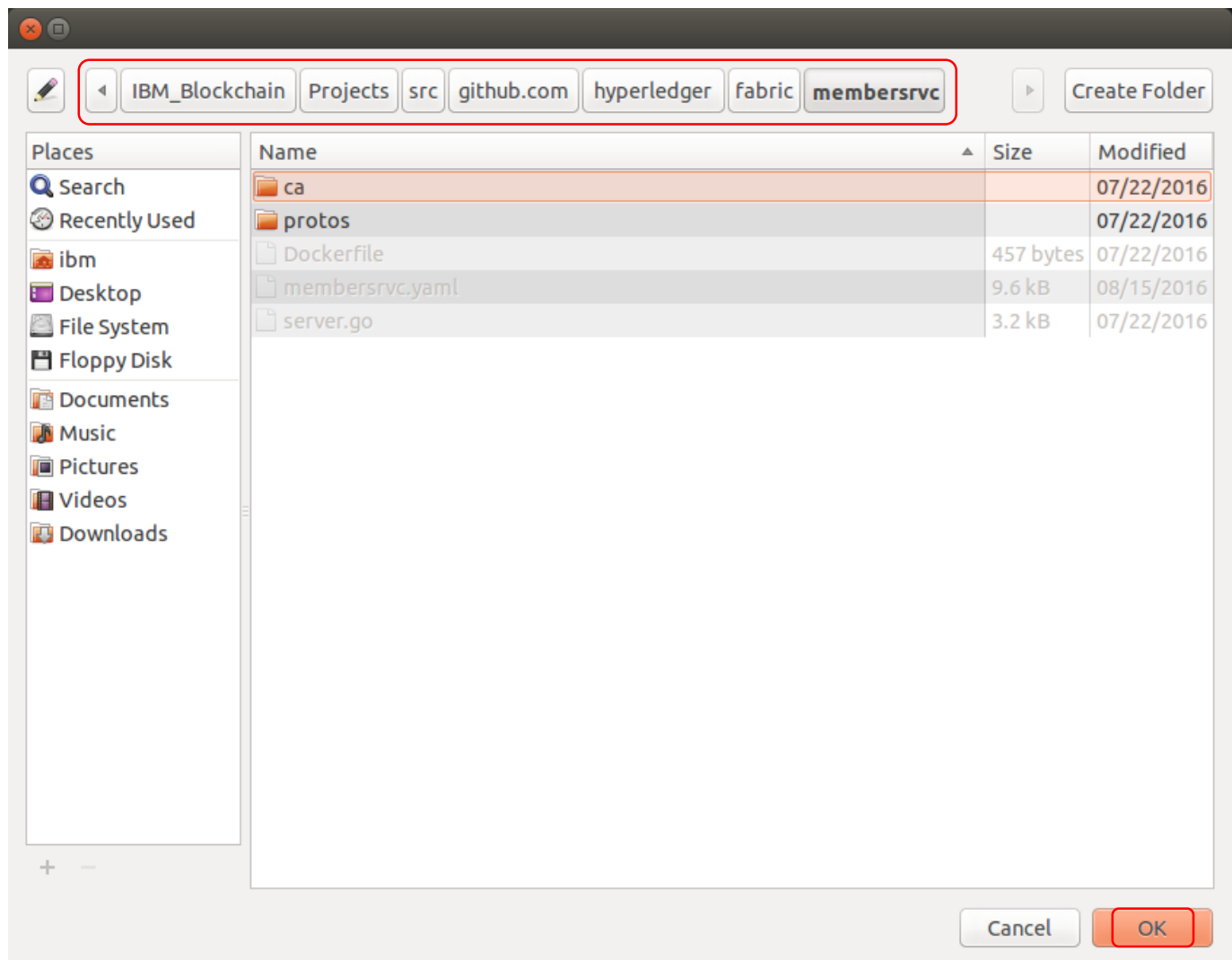
__3. Open the **membersrvcl.yaml** file in VSCode, located in:

```

/home/ibm/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric
/membersrvcl

```

Within the Visual Studio Code editor, select **File > Open Folder** and navigate to the above directory, and select **OK**.



4. Using the Explorer pane to the left of the editor, **left-click** the file **membersrv.yaml**. This will open the file in the editor window. Scroll down to review user information. For the purposes of this lab we will use **jim** with the password **6avZQLwcUe9b**.

```

72 # Note that this also controls who can register users via the
73 # client SDK.
74 #
75 # Only users with a 'registrar' section may be a registrar to
76 # register other users. In particular,
77 # 1) the "roles" field specifies which member roles may be
78 # registered by this user, and
79 # 2) the "delegateRoles" field specifies which member roles may
80 # become the "roles" field of registered users.
81 # The valid role names are "client", "peer", "validator", and
82 # "auditor".
83 #
84 # Example1:
85 #   The 'admin' user below can register clients, peers,
86 #   validators, or auditors; furthermore, the 'admin' user can
87 #   register other
88 #   users who can then register clients only.
89 #
90 # Example2:
91 #   The 'WebAppAdmin' user below can register clients only,
92 #   but none of the users registered by this user can register
93 #   other users.
94 #
95 #   admin: 1 Xurw3yU9zI0l institution_a 00001 '{"registrar":
96 #   {"roles":["client","peer","validator","auditor"],"delegateRoles":["client"]}}'
97 #   WebAppAdmin: 1 DJY27pEnl16d institution_a 00002 '{"registrar":
98 #   {"roles":["client"]}}'
99 #   lukas: 1 NPKYL39uKbkj bank_a 00001
100 #   system_chaincode_invoker: 1 DRJ20pEq115a institution_a 00002
101 #   diego: 1 DRJ23pEq116a institution_a 00003
102 #   jim: 1 6avZQLwcUe9b bank_a 00004
103 #   binm: 1 7avZQLwcUe9q institution_a 00005
104
105 # Users for asset transfer with roles test located at
106 # sdk/node/test/unit/asset-mgmt-with-roles.js
107 #   alice: 1 CMS10pE01B16 bank_a 00006

```

5. Login user **jim** on the peer. Switch back to the terminal window, and enter the following command (Note: The password **6avZQLwcUe9b** is entered separately):

```
peer network login jim
6avZQLwcUe9b
```

```

ibm@ubuntu: ~/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$ peer network login jim
2016/10/25 12:02:28 Load docker HostConfig: %v &{[] [] [] false map[] [] false [] [] [] [] host {0
} [] { map[] } false [] 0 0 0 false 0 0 0 0 []}
12:02:28.320 [crypto] main -> INFO 002 Log level recognized 'info', set to INFO
12:02:28.325 [main] networkLogin -> INFO 003 CLI client login...
12:02:28.325 [main] networkLogin -> INFO 004 Local data store for client loginToken: /var/hyperledger/produ
ction/client/
Enter password for user 'jim': *****
12:02:44.433 [main] networkLogin -> INFO 005 Logging in user 'jim' on CLI interface...
12:02:44.538 [main] networkLogin -> INFO 006 Storing login token for user 'jim'.
12:02:44.539 [main] networkLogin -> INFO 007 Login successful for user 'jim'.
12:02:44.539 [main] main -> INFO 008 Exiting....
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$

```

Note: The output from running this command should be “Login successful for user ‘jim’”.

Now we can deploy the chaincode **example02** to the peer, using the same terminal where we have just logged in as **jim**.

Deploys can be done for chaincode which is located on the **\$GOPATH** root directory, or in a public github repository. For the purposes of this lab we will continue to make use of the local **example02** sample. The full directory is not specified as it implies **\$GOPATH/src** as the root.

- ___6. Enter the following **peer** command in the vagrant shell. The parameters passed in are used to call the **Init()** function reviewed earlier:



This command is also located in the file '**FYI – Cheatsheet Commands handout.pdf**' which can be found on the Ubuntu desktop in the VM. The command can be copied and pasted to the terminal window.

```
peer chaincode deploy -p
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
-c '{"Function":"init", "Args": ["a","100", "b", "200"]}' -u jim
```

```
ibm@ubuntu: ~/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$ peer chaincode deploy -p github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02 -c '{"Function":"init", "Args": ["a","100", "b", "200"]}' -u jim
2016/10/25 12:03:43 Load docker HostConfig: %v &{[] [] [] [] false map[] [] false [] [] [] [] host { 0
} [] { map[] } false [] 0 0 0 false 0 0 0 0 []}
12:03:43.322 [crypto] main -> INFO 002 Log level recognized 'info', set to INFO
c126325b3483090998849ad29bda82c1af52a364f467bd8e53bf6ab920d3f8a786385d9a35b477bb650aaab8921b65a8777efbeb321
2355544362df80114a9e0
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$
```

Note: The output from running this command is the chaincode identifier:

```
c126325b3483090998849ad29bda82c1af52a364f467bd8e53bf6ab920d3f8a786385d9a35b47
7bb650aaab8921b65a8777efbeb3212355544362df80114a9e0
```

This will be different for each deploy, and uniquely identifies the chaincode deployed. Take a copy of the identifier from the deployment as it will be needed to invoke and query the chaincode.

Section 4. Invoking the chaincode

We will now run the chaincode that has been deployed. This means submitting transactions to the validating peers that call the `invoke` method. This will add a new transaction to the blockchain.

- ___1. Enter the following **peer** command. The parameters passed in are used to call the **Invoke()** function we reviewed earlier. You will need to substitute the chaincode identifier with the one returned from the deployment in the previous step:



This command is also located in the file '**FYI – Cheatsheet Commands handout.pdf**' which can be found on the Ubuntu desktop in the VM. The command can be copied and pasted to the terminal window.

```
peer chaincode invoke -n <chaincode identifier from deployment> -c
'{"Function": "Invoke", "Args": ["a", "b", "10"]}' -u jim
```

```
ibm@ubuntu: ~/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$ peer chaincode invoke -n c126325b3483090998849ad29bda82c1af52a364f467bd8e53bf6ab92
9d3f8a786385d9a35b477bb650aaab8921b65a8777efbeb3212355544362df80114a9e0 -c '{"Function": "Invoke", "Args":
["a", "b", "10"]}' -u jim
2016/10/25 12:05:17 Load docker HostConfig: %v &{[] [] [] [] false map[] [] false [] [] [] [] host { 0
} [] { map[] false [] 0 0 0 false 0 0 0 0 []}
12:05:17.254 [crypto] main -> INFO 002 Log level recognized 'info', set to INFO
1997c322-15b0-4bcb-b16f-ae8c0e44c5e2
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$
```

Note: The output from a successful **Invoke()** of the chaincode is a transaction UUID, for example:

1997c322-15b0-4bcb-b16f-ae8c0e44c5e2

- ___2. Review the transaction for UUID **1997c322-15b0-4bcb-b16f-ae8c0e44c5e2**, by querying the blockchain. To retrieve the transaction we must use the REST interface. Enter the following **curl** command, substituting the transaction UUID returned from the previous **Invoke()**.

```
curl http://localhost:5000/transactions/<transaction UUID>
```

```

vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/devenv
/go/chaincode_example02$ curl http://localhost:5000/transactions/1997c322-15b0-4bcb-b16f-ae8c0e44c5e2
{"type":2,"chaincodeID":"EoABYZEyNjMyNWIZNDgzMDkw0Tk4ODQ5YWQyOWJkYTgyYzFhZjUyYTM2NGY0NjdiZDhlNTNiZjZhYjkyMG
QzZjhhNzg2Mzg1ZDlhMzViNDc3YmI2NTBhYWFiODkyMWI2NWE4Nzc3ZWZiZWZmZjYyMTU1NTQ0MzYyZGY4MDExNGE5ZTA=", "payload":
"CqQBCAESgWESgAFjMTI2MzI1YjM0ODMwOTA5OTg4NDlhZDI5YmRhODJjMWFmNTJhMzY0ZjQ2N2JkOGU1M2JmNmFiOTIwZDNmOGE3ODYzODV
kOWEzNWlONzdiYjY1MGFhYWl0OTIxYjY1YTg3Nzd1ZmJlYjMyMTIzNTU1NDQzNjJkZjgwMTE0YTllMBoaCgZJbnZva2USBmludm9rZRIByR
IBYhICMTA=", "uuid":"1997c322-15b0-4bcb-b16f-ae8c0e44c5e2", "timestamp":{"seconds":1477397117,"nanos":3779972
36}, "nonce":"emSQVLWDj/s//AgpFLXYz0iprue6wewx", "cert":"MIICPzCCAeSgAwIBAgIRAKhMgtDPg036p43ciSRr7ywwCgYIKoZI
zj0EAwMwMTELMakGA1UEBhMCVVMxZDASBgNVBAoTC0h5cGVybGVkZ2ZyYmQwYDVR0BAYEBAECAwQwDwYDVR0jBAGwBoAEAQIDBDBNBgYqAwQFBgcBAF8EQEUW61RiZe6m
D6ElT+10T47RzC6hXW/+nITe6xZiOb9uiuw7Yy3vKLBkZcoppdo72lyk+fjBR2ZBCuCs9m6OKKEwSgYgKgmEBQYIBEA0NiBGYPd1/QMN+0U/FDbfpcamMe8QgHKxPDOW6II9oPlyMBNjN8vQ
ca2ZYpNcnvhgOb/yKqct35eXGpgXza0MAoGCCqGSM49BAMDA0kAMEYCIQC32f/fl/LYA+PoWc45LyzSvxx8vnoLFJG3zcp0uSjkaQIhAI
PhBmc5prh8GgWn+xt9hmsPgMPKbKZ0ZmdTyNNT//V", "signature":"MEQCIEZ4g/6coHQBHpfmScnRWSrYHd9cKrISHfyIdDaVnAYnA
iAXWprSxGVBV5WjKp0821R0UafbmfcvOQTsgvzna+fng==" }
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/devenv
/go/chaincode_example02$

```

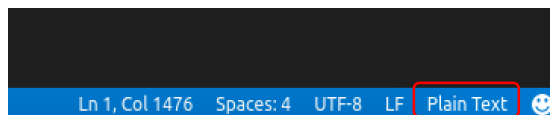
The output from this command is a JSON structure that includes all the information recorded on the blockchain for the transaction. Highlight and copy the JSON structure, so that we can review it in the Visual Studio Code editor:

```

{"type":2,"chaincodeID":"EoABYZEyNjMyNWIZNDgzMDkw0Tk4ODQ5YWQyOWJkYTgyYzFhZjUyYTM2NGY0NjdiZDhlNTNiZjZhYjkyMG
QzZjhhNzg2Mzg1ZDlhMzViNDc3YmI2NTBhYWFiODkyMWI2NWE4Nzc3ZWZiZWZmZjYyMTU1NTQ0MzYyZGY4MDExNGE5ZTA=", "payload":
"CqQBCAESgWESgAFjMTI2MzI1YjM0ODMwOTA5OTg4NDlhZDI5YmRhODJjMWFmNTJhMzY0ZjQ2N2JkOGU1M2JmNmFiOTIwZDNmOGE3ODYzODV
kOWEzNWlONzdiYjY1MGFhYWl0OTIxYjY1YTg3Nzd1ZmJlYjMyMTIzNTU1NDQzNjJkZjgwMTE0YTllMBoaCgZJbnZva2USBmludm9rZRIByR
IBYhICMTA=", "uuid":"1997c322-15b0-4bcb-b16f-ae8c0e44c5e2", "timestamp":{"seconds":1477397117,"nanos":377997236}, "nonce":"emSQVLWDj/s//AgpFLXYz0iprue6wewx", "cert":"MIICPzCCAeSgAwIBAgIRAKhMgtDPg036p43ciSRr7ywwCgYIKoZIzj0EAwMwMTELMakGA1UEBhMCVVMxZDASBgNVBAoTC0h5cGVybGVkZ2ZyYmQwYDVR0BAYEBAECAwQwDwYDVR0jBAGwBoAEAQIDBDBNBgYqAwQFBgcBAF8EQEUW61RiZe6mD6ElT+10T47RzC6hXW/+nITe6xZiOb9uiuw7Yy3vKLBkZcoppdo72lyk+fjBR2ZBCuCs9m6OKKEwSgYgKgmEBQYIBEA0NiBGYPd1/QMN+0U/FDbfpcamMe8QgHKxPDOW6II9oPlyMBNjN8vQca2ZYpNcnvhgOb/yKqct35eXGpgXza0MAoGCCqGSM49BAMDA0kAMEYCIQC32f/fl/LYA+PoWc45LyzSvxx8vnoLFJG3zcp0uSjkaQIhAIPhBmc5prh8GgWn+xt9hmsPgMPKbKZ0ZmdTyNNT//V", "signature":"MEQCIEZ4g/6coHQBHpfmScnRWSrYHd9cKrISHfyIdDaVnAYnAiaXWprSxGVBV5WjKp0821R0UafbmfcvOQTsgvzna+fng==" }

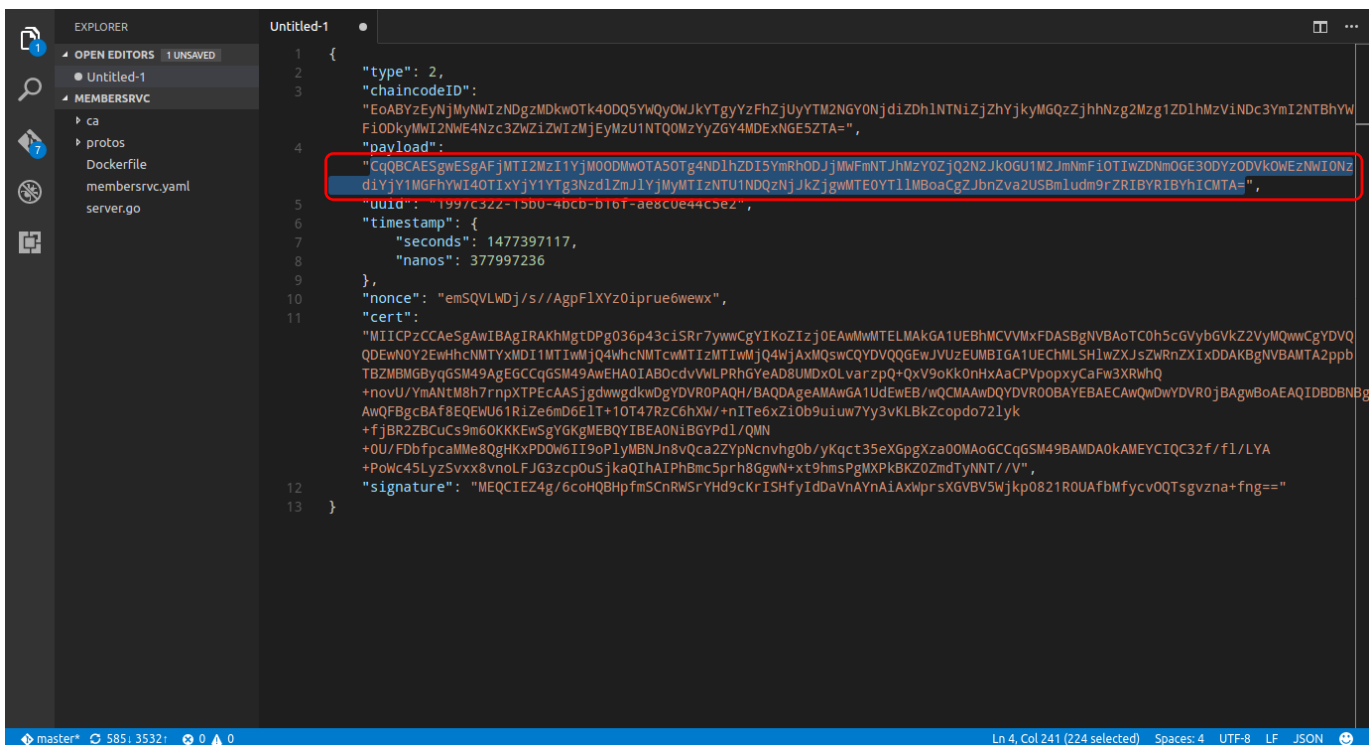
```

- ___3. Switch back to the editor, and choose **File > New File** from the menu or press **“ctrl+n”** to create a new file. Paste the JSON output from the previous **curl** command into the new file.
- ___4. In the bottom right of the editor window, click on “Plain Text” and from the list that appears choose “JSON” to set the editor format to show JSON data.

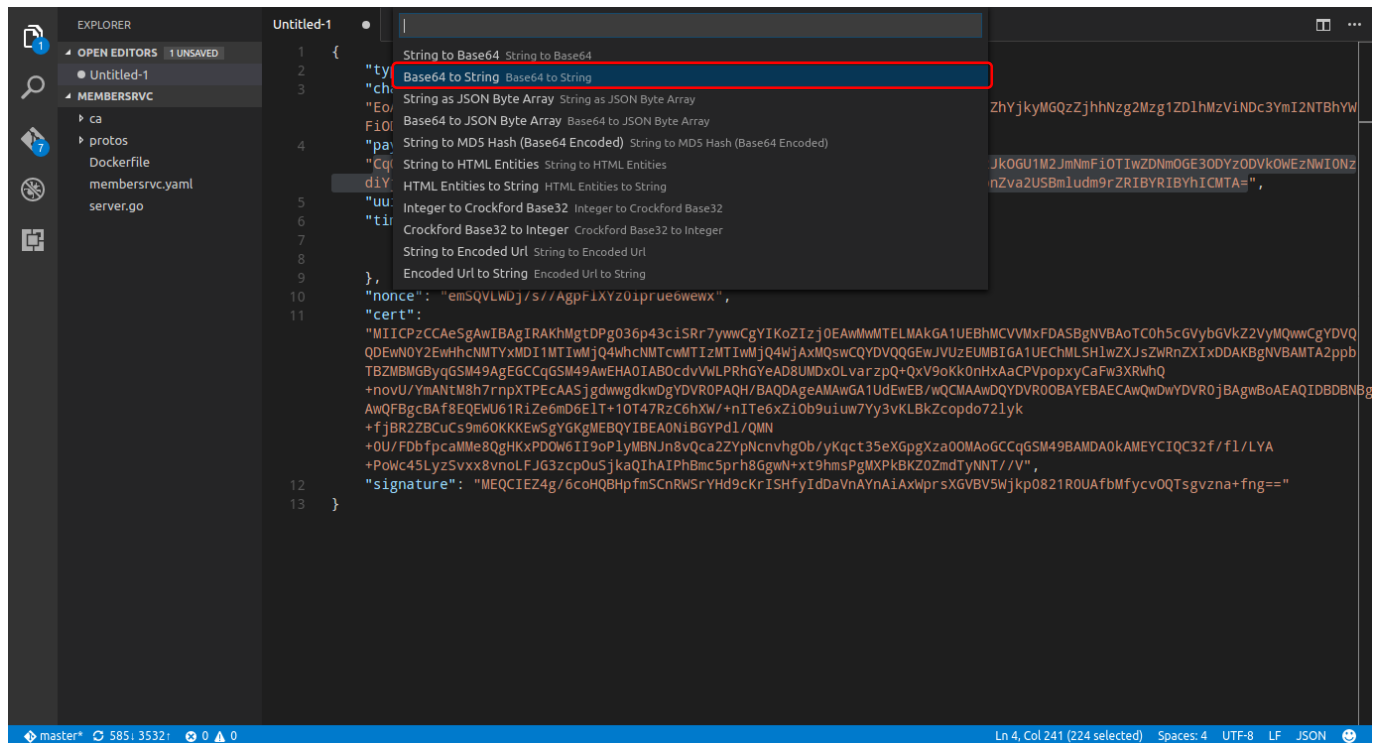


- ___5. You will see the text is colour coded, but not properly formatted yet. **Right-click** and choose **‘Format Code’**. The text will now look similar to the screen shot below. You should also choose **View > Toggle Word Wrap** to wrap the text onto the screen.

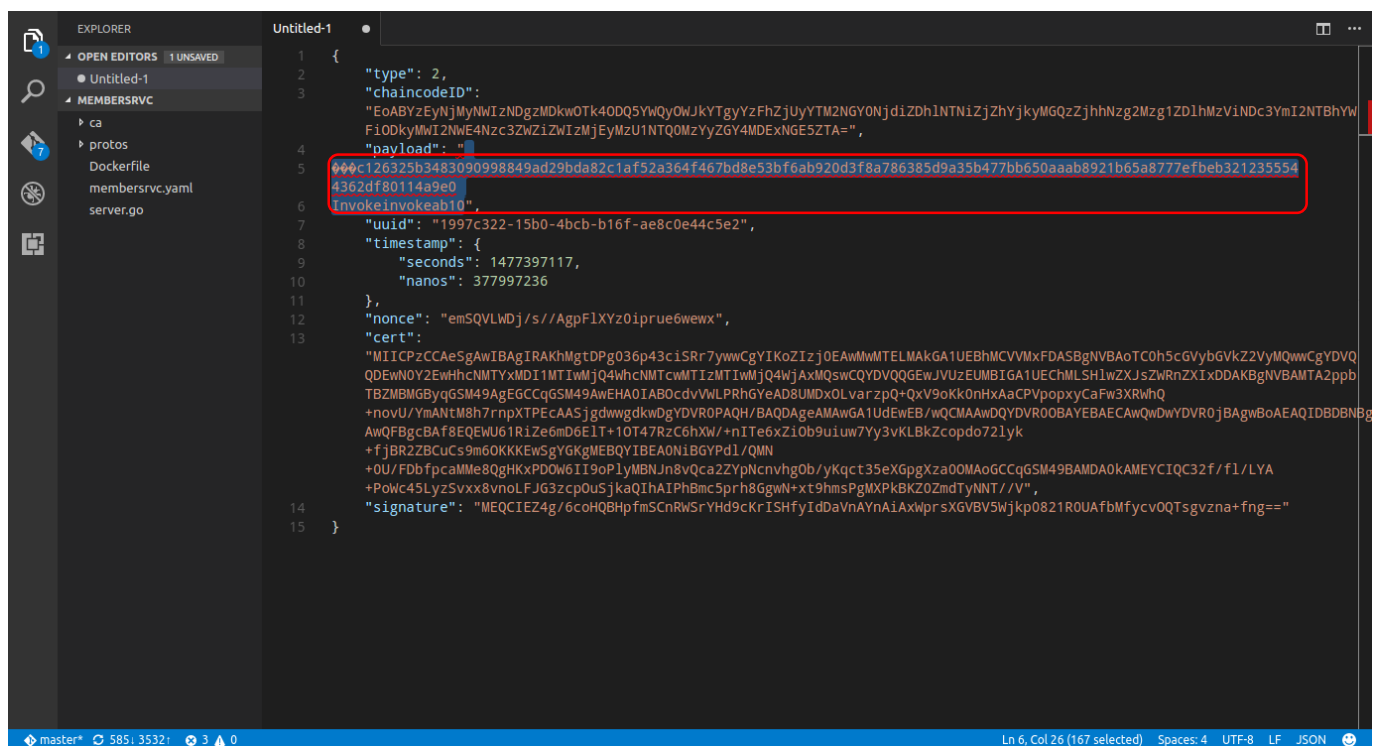
___6. Looking at the data you can see that much of it is not easily understood, because the data is also encoded using “base64” encoding. To see a little more of the contents we will decode the value of the “payload” field. First select all the data in the payload field inside the quotes, but do not select the quotes themselves.



7. Now press “**ctrl+alt+c**” and the conversion menu will appear.



- ___8. From the conversion menu select the second option “**Base64 to String**” and the selected data will be decoded.



Section 5. Querying chaincode

Having run the chaincode successfully, it can now be queried.

- __1. Enter the following **peer** command. The parameters passed in are used to call the **Query()** function we reviewed earlier. You will need to substitute the chaincode identifier with the one returned from the deployment in the previous step:



This command is also located in the file '**FYI – Cheatsheet Commands handout.pdf**' which can be found on the Ubuntu desktop in the VM. The command can be copied and pasted to the terminal window.

```
peer chaincode query -n <chaincode identifier from deployment> -c
'{"Function": "query", "Args": ["a"]}' -u jim
```

```
ibm@ubuntu: ~/Documents/IBM_Blockchain/Projects/src/github.com/hyperledger/fabric/devenv
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$ peer chaincode query -n c126325b3483090998849ad29bda82c1af52a364f467bd8e53bf6ab920
d3f8a786385d9a35b477bb650aaab8921b65a8777efbeb3212355544362df80114a9e0 -c '{"Function": "query", "Args": ["
a"]}' -u jim
2016/11/07 15:10:40 Load docker HostConfig: %v &{[] [] [] [] false map[] [] false [] [] [] [] host { 0
} [] { map[] false [] 0 0 0 false 0 0 0 0 []}
15:10:40.700 [crypto] main -> INFO 002 Log level recognized 'info', set to INFO
90
vagrant@hyperledger-devenv:v0.0.10-3e0e80a:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode
/go/chaincode_example02$
```

The output from running the query is the value **90** for the user **a**.

- __2. Repeat the query for **'b'**.
- __3. Rerun the **peer** invoke command, followed by the **peer** query command.

Congratulations on completing the Blockchain Unchained lab!

Appendix A. Notices

This information was developed for products and services offered in the U.S.A.
IBM may not offer the products, services, or features discussed in this document in other countries.

Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM

products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
<i>System z</i>	<i>Tivoli</i>	<i>WebSphere</i>	<i>Workplace</i>	<i>System p</i>	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

NOTES

[illegible]

NOTES

[illegible]



© Copyright IBM Corporation 2016.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.



Please Recycle
