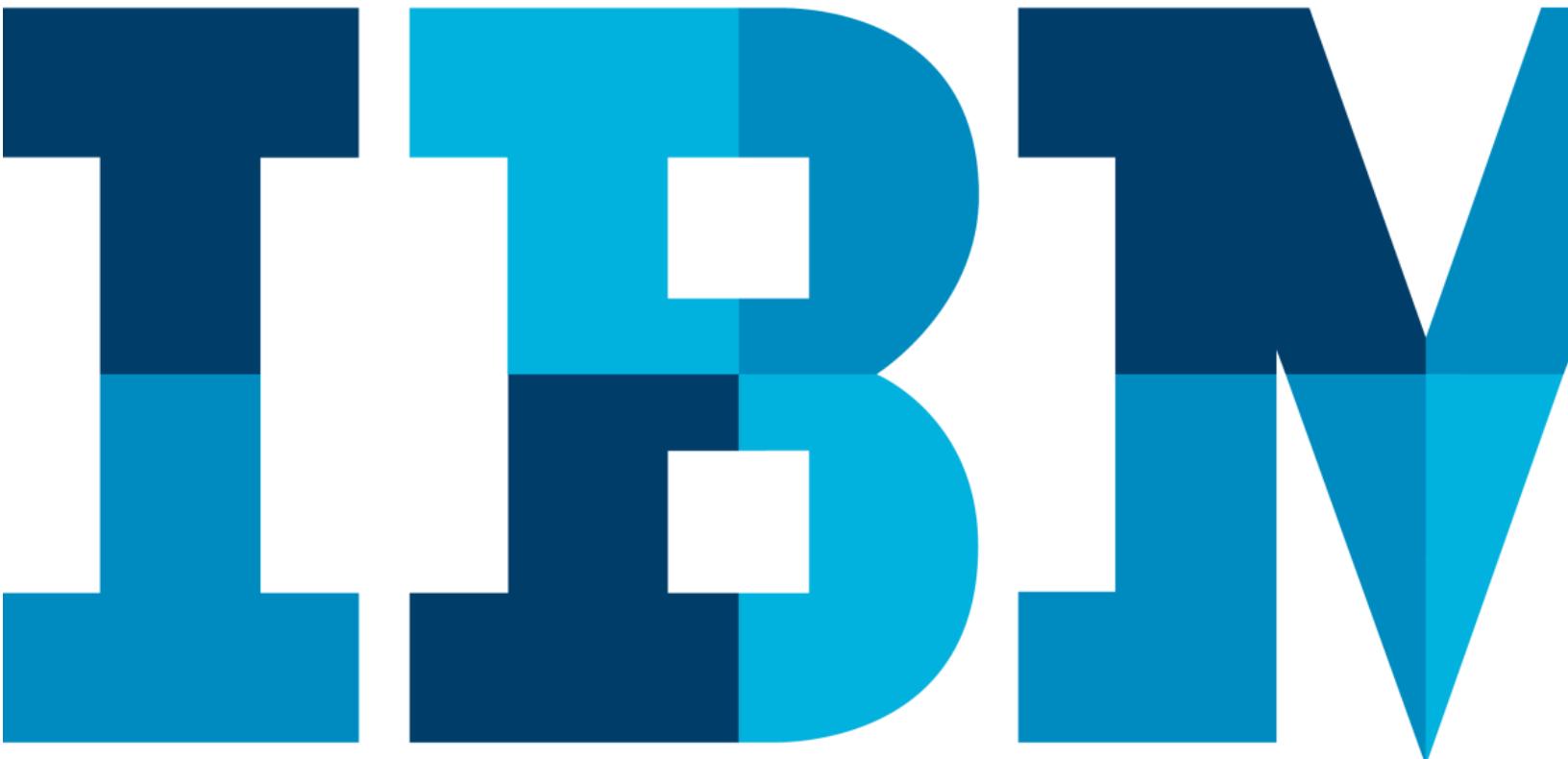


# IBM Blockchain Platform Hands-On

**Lab 4:**

*Verifiable Exchange with Private Data Collections*



# Table of Contents

<b>Disclaimer</b> .....	3
Overview of the lab environment and scenario .....	5
Simplified Commodity Trading Exchange .....	6
Trade Network .....	11
Lab Structure.....	12
1 Build and Start Fabric for the ‘Trade Network’ .....	13
2 Deploy the first version of the Smart Contract to test & prove the Trade Network .....	30
3 Private Data for the Offer and Accept Transactions .....	38
3.1 Invoking the Offer and Accept transactions.....	43
4 Work with the Cross Verify Functions.....	54
4.1 Understanding the Cross-Verify implementation.....	56
4.2 Performing the Cross Verification of Private Data.....	63
5 Add Demo Data and Private Query Transaction.....	70
5.1 Create some Demo Data for querying data .....	71
5.2 Executing Rich Queries against the Private Data Collections .....	74
<b>We Value Your Feedback!</b> .....	80
<b>Appendix 1 - transaction List Info for ‘exchangecontract’</b> .....	81
<b>Appendix 2 Access control considerations for Private data</b> .....	83
<b>Appendix 3 - Alternate Trade Network Fabric</b> .....	84

## Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts.

In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.

**Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.**

Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.

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 about this publication 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. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com and [names of other referenced IBM products and services used in the presentation] are trademarks of International Business Machines Corporation, 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](http://www.ibm.com/legal/copytrade.shtml).

© 2019 International Business Machines Corporation. No part of this document may be reproduced or transmitted in any form without written permission from IBM.

**U.S. Government Users Restricted Rights – use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.**

## Overview of the lab environment and scenario

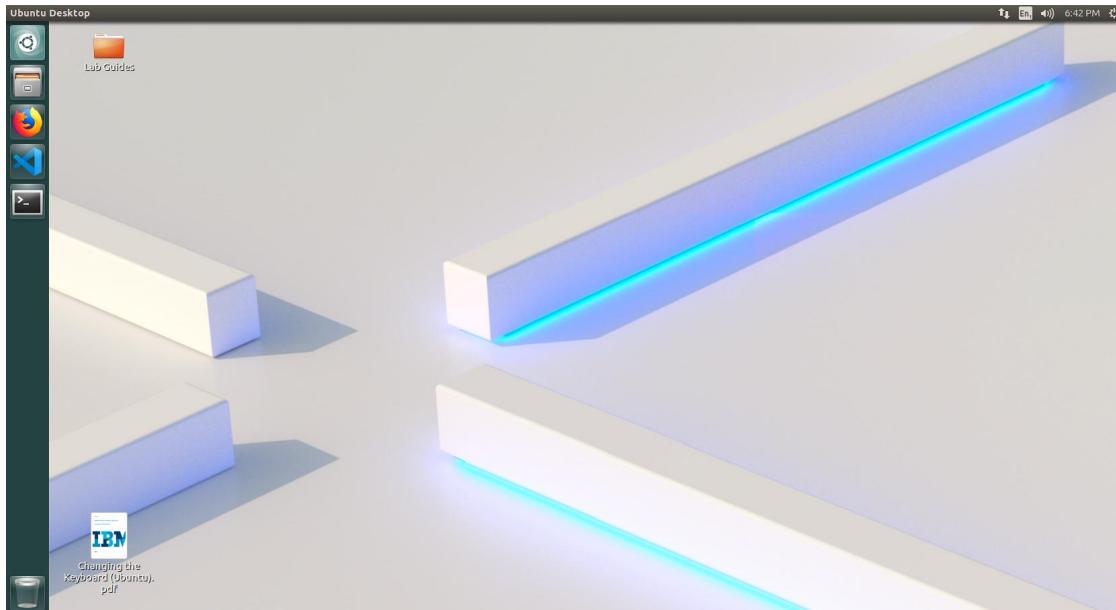
This lab is the 4<sup>th</sup> lab in the series and shows how the IBM Blockchain Platform Extension for VS Code can be used to work with Private Data Collections. The lab uses a local custom fabric.

**Note:** The screenshots in this lab guide were taken using version **1.39.2 of VS Code**, and version **1.0.12 of the IBM Blockchain Platform extension**. If you use different versions, you may see differences from those shown in this guide.

**Start here. Instructions are always shown on numbered lines like this one:**

- \_\_ 1.** If it is not already running, start the virtual machine for the lab. Your instructor will tell you how to do this if you are unsure.
- \_\_ 2.** Wait for the image to boot and for the associated services to start. This happens automatically but might take several minutes. The image is ready to use when the desktop is visible as per the screenshot below.

**Note:** If it asks you to login, the userid and password are both “**blockchain**”.



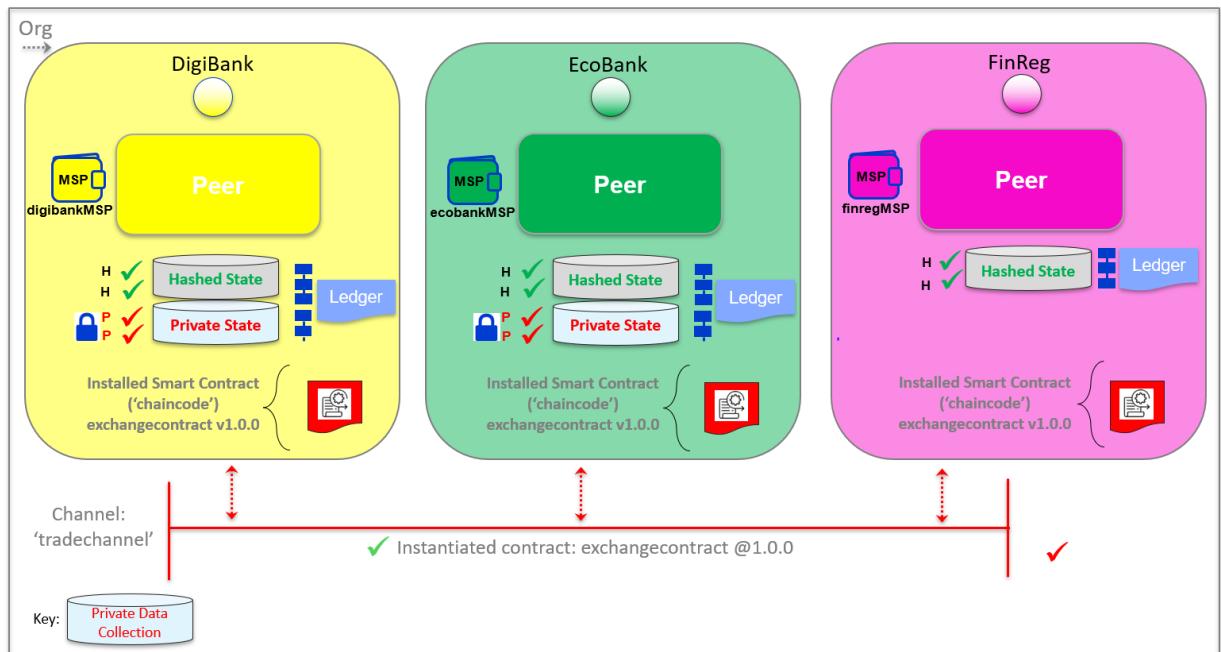
## Simplified Commodity Trading Exchange

**DigiBank** and **EcoBank** participate (as investing banks) in a Commodity Exchange network – the buying and selling **Commodity Contracts** (legal contracts) for investment purposes). A commodity exchange is a regulated marketplace that facilitates the purchase and sale of such **Commodity Contracts** whose values are tied to the price of commodities (e.g., oil, corn, silver, gold). These contracts represent the total value or price of tradeable commodities. Each advertised commodity has a Contract ID in this lab. So in summary: a Contract is put up for sale – i.e. it is advertised at a ‘market’ price, then offers (and counter offers perhaps) are made, before finally an offer is accepted by the seller and a deal is made. It then remains to cross-verify the offer details against the original offer and write that status to the ledger.

In this lab example we see that Ecobank and Digibank have private data collections when it is likely that both banks know all the data! But imagine a scenario where there are many more banks trading. Digibank would not be the only bank making an offer, and these multiple offers would be kept private and confidential so "Bigbank" for instance would not be aware of the offer that Digibank made.

An overview of the exchange network on the blockchain and organisations involved, is shown below:

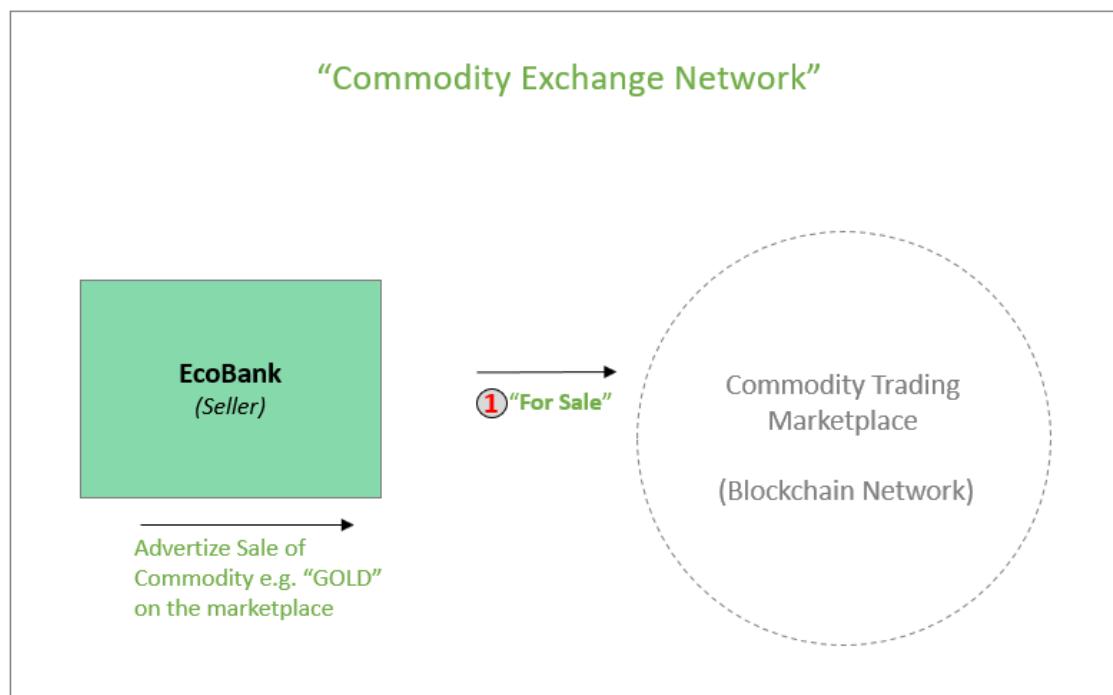
### Commodity Trading Exchange: Overview



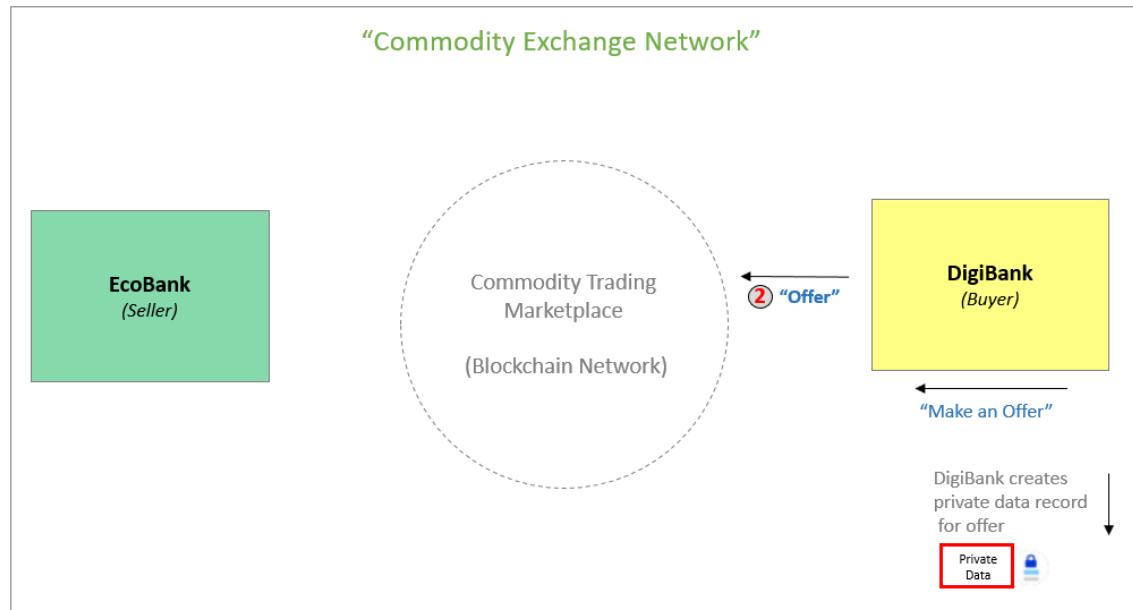
Notice in the above diagram, the regulator **FinReg** does not store any private data of its own – it does however have a copy of the ledger including the hashed state ledger. Its role is to audit records – e.g. third-party verification or dispute resolution of private data, shared with it. The hash state of the private data on the ledger, serves as evidence of the original private data transaction; the third party can compute the hash of the private data and see if it matches the state.

The smart contract reflects the transactions in this network – lets examine the transaction in the lab:

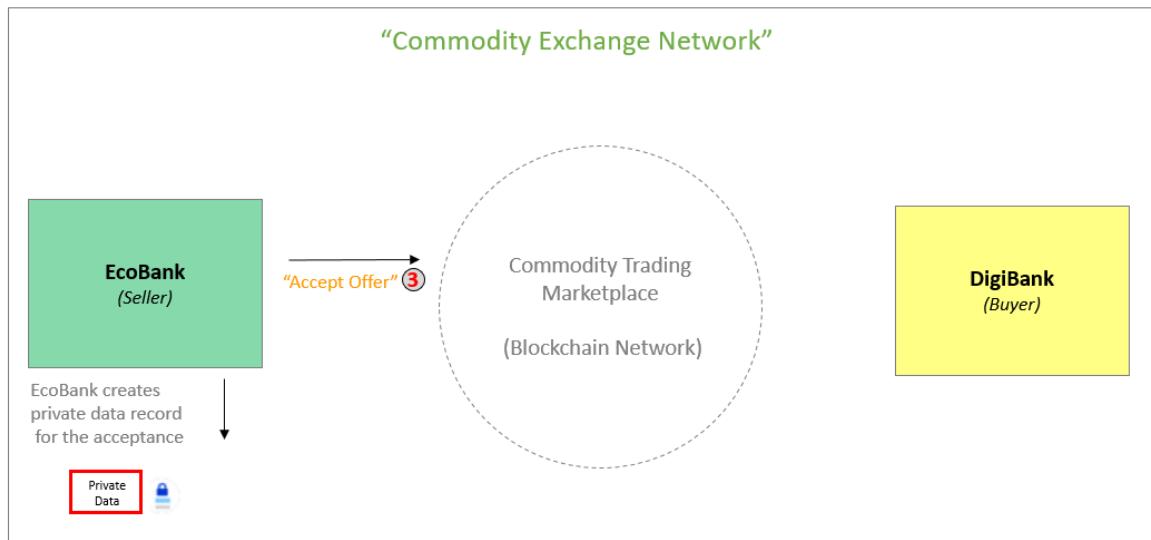
1. **EcoBank** advertises or ‘broadcasts’ a commodity contract for sale on the marketplace



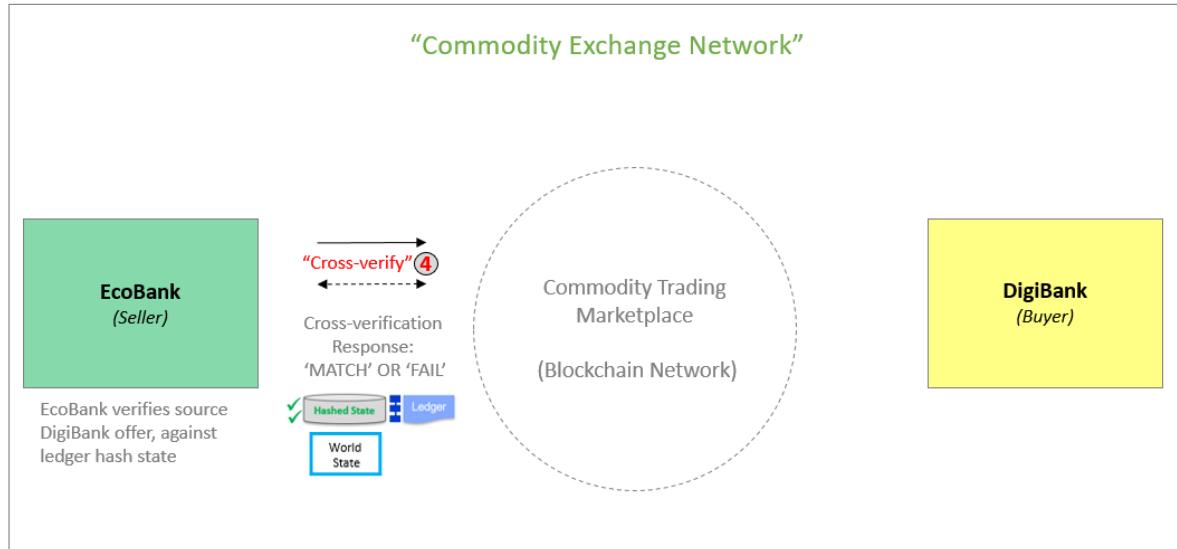
2. **DigiBank** makes an offer to EcoBank. Note that DigiBank writes a record of this offer, to its own private data store. A cryptographic ‘hash’ of the record is also recorded on the blockchain (channel world state).



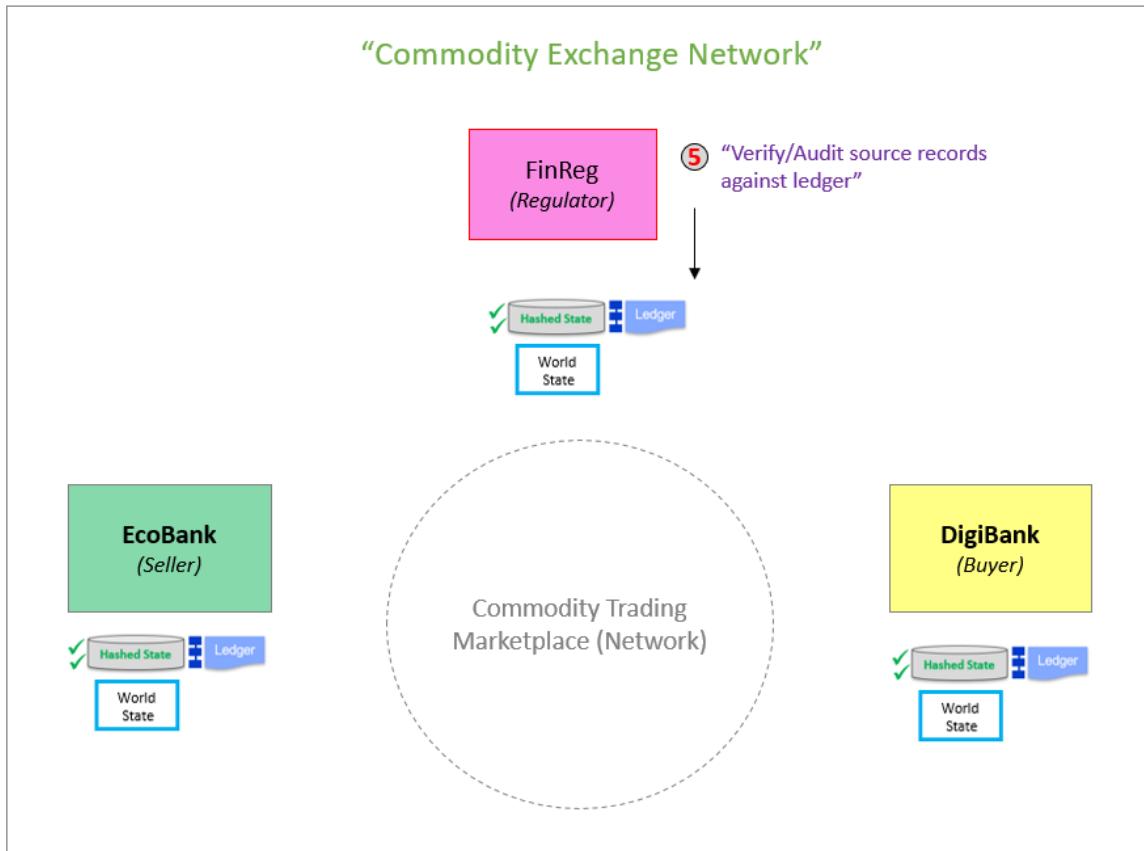
- EcoBank receives the offer and informs DigiBank that it accepts the offer; EcoBank writes its own 'accept with offer' record to its own private data store – the offer on the ledger is as yet UNVERIFIED – meaning, it needs to be cross-verified.



- EcoBank calculates a hash of the source offer private data and verifies this hash, against the private data hash that was written by DigiBank in step 2. The smart contract compares the hashes and if they match, will update the accept/offer to 'CROSSVERIFIED' – this status attribute is not actually private – it is shared as a general status on the ledger that the organisations share.

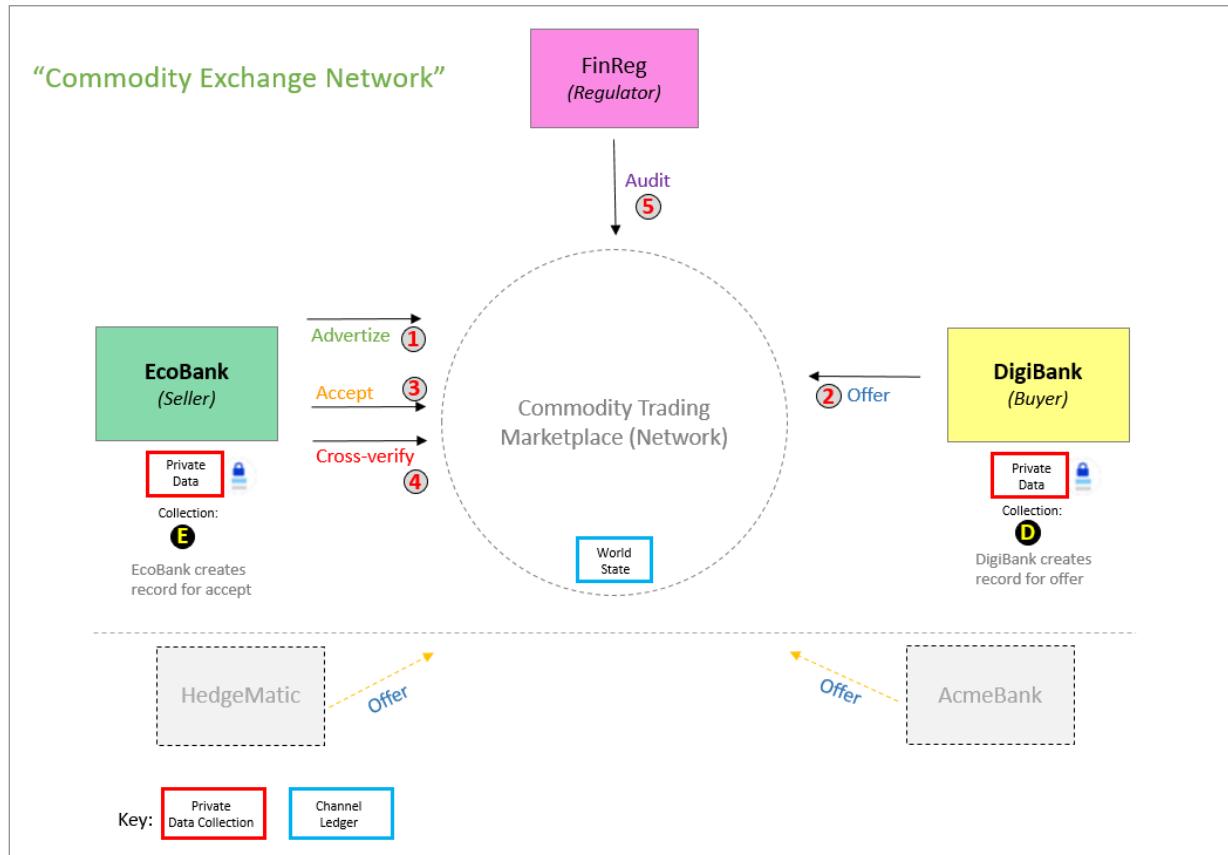


5. **FinReg**, a 3<sup>rd</sup> party regulator, sometime later, carries out audit checks and takes source offer or accept data, and cross-verifies the records, against the hashes originally written to the ledger.



The following diagram contains a consolidated transaction summary:

### Private Data Lab: Transaction Summary



Note that we focus on 3 organisations in this lab exercise; but remember, there could eventually be more (reference the grey boxes in the above diagram) already joined the network and be on the same channel.

**FinReg**, the regulator – will, later in the lab, require to validate historical offers or accepts, to see they are the genuine, source data that each organisation (DigiBank, EcoBank) recorded, independently.

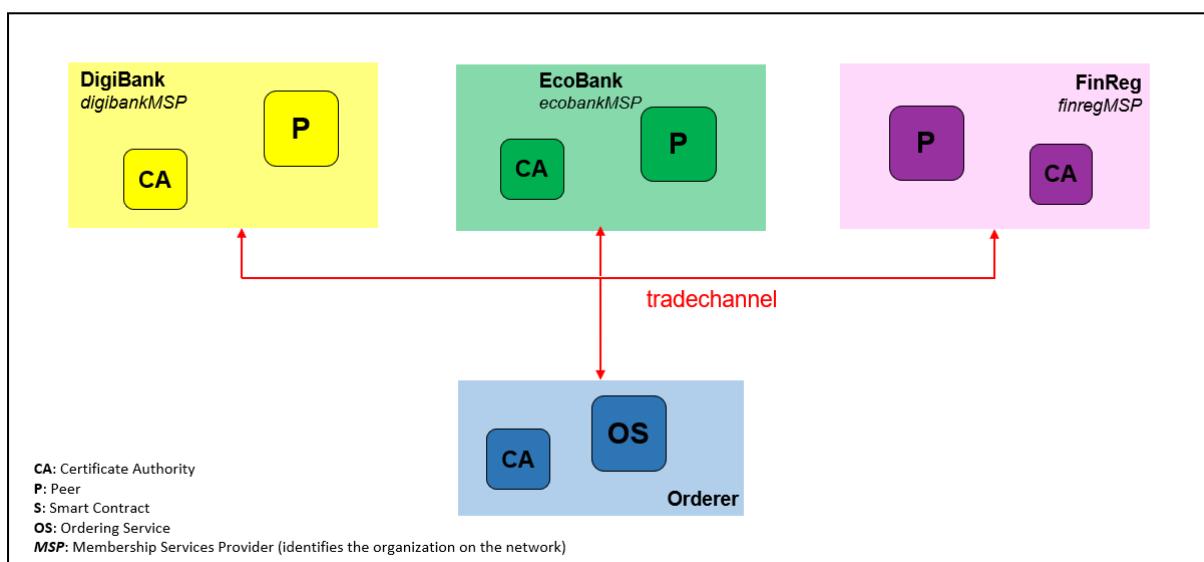
## Trade Network

The lab uses a custom Hyperledger Fabric network with 3 Organisations to illustrate the Private Data features of Hyperledger Fabric. It comprises a set of Docker Containers built with Ansible scripts.

The Ansible tool is an open-source configuration and deployment tool. The VM contains Ansible scripts to build the custom network for the lab – these scripts run inside a Docker container. The Lab requires no knowledge of Ansible, just an awareness that it is being used.

The scripts and the build process are currently an evaluation example, but a similar feature may be integrated with the VS Code extension in the future to allow more realistic topologies to be available to developers.

## Custom Network



The diagram illustrates the Custom Network showing the channel used is **tradechannel**. Each Peer will be using CouchDB as the State Database to enable rich queries to be run in the Lab.

## Lab Structure

This lab is structured into 6 parts (one is optional, to complete if time allows).

**Part 1** of this lab will take you through Building and Starting the Fabric for the Trade Network.

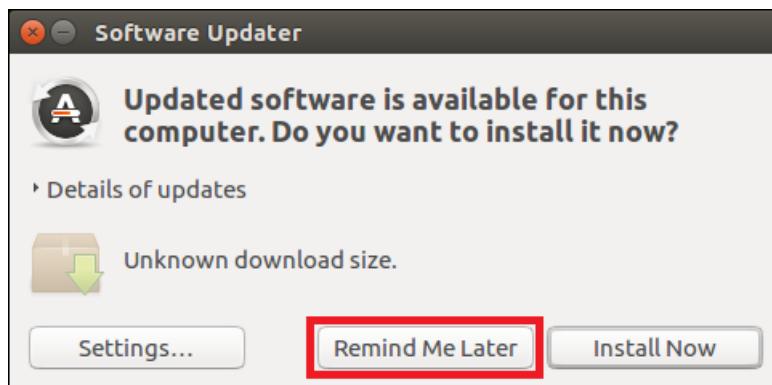
**Part 2** of this lab will deploy the first version of the Smart Contract that is used to test and prove the Trade Network Fabric

**Part 3** of this lab will focus on the Offer and Accept transactions and the writing of the initial Private Data.

**Part 4** of this lab will show the Cross-Verify transactions allowing parties to verify the hashes of data of Private Data collections.

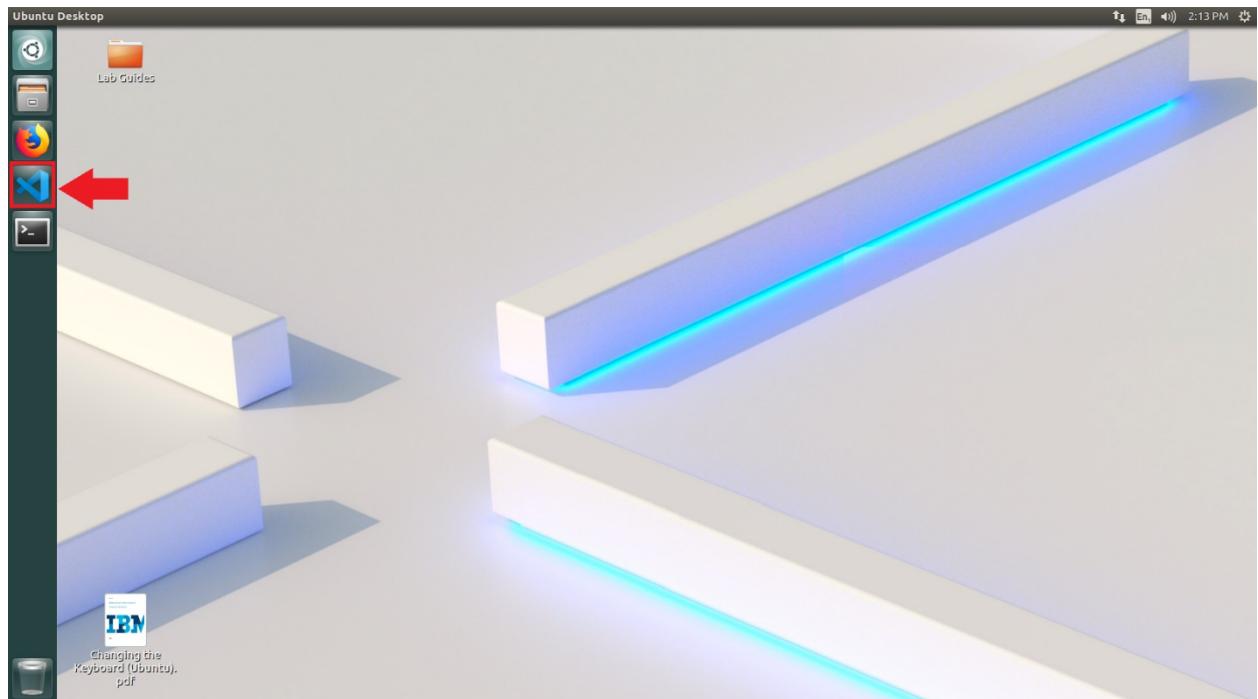
**Part 5** adds additional sample private data for queries, and works with rich query transactions, running against the Private Data Collections

**Note** that if you get an “Software Updater” pop-up at any point during the lab, please click “**Remind Me Later**”:

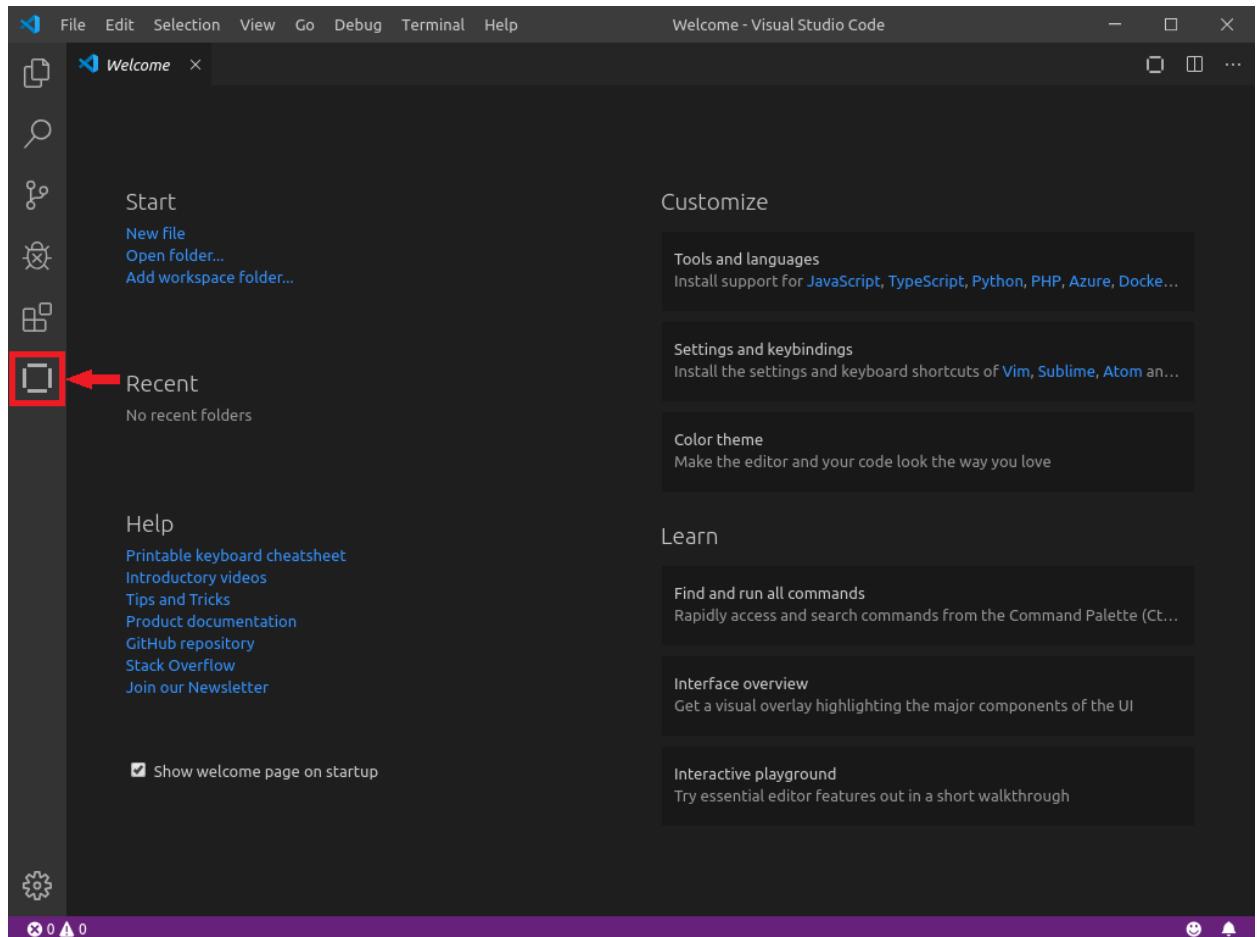


## 1 Build and Start Fabric for the ‘Trade Network’

-- 3. VS Code may already be running from a previous lab exercise, but if not, launch VS Code by clicking on the VS Code icon in the toolbar.



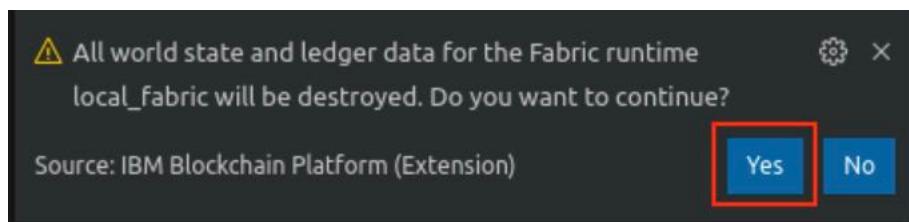
- 4. When VS Code opens, click on the IBM Blockchain Platform icon in the Activity Bar in VS Code as shown below.



- 5. Right click on the **Local Fabric** environment and click the option the **Teardown Fabric Runtime**:



Click **Yes** in the bottom right corner to confirm the teardown.



We are now going to start the custom Hyperledger Fabric network. This VM includes a script to automate the process.

- 6. On the Ubuntu Dock **click** the **Terminal** icon to open a new terminal window:



- 7. Using the following command in the terminal, change to the folder containing the ansible build scripts:

```
cd workspace/hlf-ansible-master/
```

```
blockchain@ubuntu:~$ cd workspace/hlf-ansible-master/
blockchain@ubuntu:~/workspace/hlf-ansible-master$ █
```

The Custom Fabric will be built based on a “playbook” file in yaml format. The playbook describes the network we will use in the lab and is consumed by the Ansible scripts described in the introduction.

Briefly examine the playbook in the terminal window.

- 8. Type the following command:

```
more site.yml
```

```
blockchain@ubuntu:~/workspace/hlf-ansible-master$ more site.yml
---
- name: Deploy blockchain infrastructure NO smart contracts
  hosts: localhost
  vars:
    infrastructure:
```

(When using the “more” command, Press the spacebar for “Next Screen”.)

Note the definition of **ecobankMSP** and the **ids** associated with Ecobank – you will use them later in this section.

Issue the following command to run the deploy script which runs the docker container to build the Trade Network fabric. (The command takes about 2 minutes to complete and writes many lines of output in the terminal.)

```
./deploy.sh
```

```
blockchain@ubuntu:~/workspace/hlf-ansible-master$ ./deploy.sh
PLAY [Deploy blockchain infrastructure NO smart contracts] *****
TASK [Gathering Facts] *****
ok: [localhost]
```

Successful completion of this command will look similar to the following:

```
TASK [ibp : Join peer to channel] *****
changed: [localhost]

TASK [ibp : Manage all contracts] *****
fatal: [localhost]: FAILED! => {"msg": "'contracts' is undefined"}

PLAY RECAP *****
localhost          : ok=324  changed=152  unreachable=0    failed=1    skipped=40   rescued=0
ignored=0

blockchain@ubuntu:~/workspace/hlf-ansible-master$ █
```

Don't worry about the red Fatal message – the script can optionally deploy a Smart Contract but in this lab the Smart Contract will be manually deployed later.

#### Note

If you see a red error ending with “Bind for 0.0.0.0:17054 failed port is already allocated” then the Local Fabric in VS Code is still present and needs a teardown as described in Step 5.

- 9. Run the following command to verify that the 11 expected containers are running.  
 “docker ps” is the command to display running containers, and the format parameter is just used to simplify the output.

```
docker ps --format 'table {{.Names}}:\t {{.Ports}}'
```

```
blockchain@ubuntu:~/workspace/hlf-ansible-master$ docker ps --format 'table {{.Names}}:\t {{.Ports}}'
NAMES                                PORTS
orderer.example.com:                  7050/tcp, 0.0.0.0:17050->17050/tcp
ca.orderer.example.com:                7054/tcp, 0.0.0.0:16054->16054/tcp
peer0.finreg.example.com:              0.0.0.0:19051-19052->19051-19052/tcp
couchdb0.finreg.example.com:           4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp
ca.finreg.example.com:                 7054/tcp, 0.0.0.0:19054->19054/tcp
peer0.digibank.example.com:            0.0.0.0:18051-18052->18051-18052/tcp
couchdb0.digibank.example.com:          4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp
ca.digibank.example.com:               7054/tcp, 0.0.0.0:18054->18054/tcp
peer0.ecobank.example.com:              0.0.0.0:17051-17052->17051-17052/tcp
couchdb0.ecobank.example.com:           4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp
ca.ecobank.example.com:                7054/tcp, 0.0.0.0:17054->17054/tcp
```

- 10. In order to have full access to some configuration files, file protections need to be modified. Run the following 2 commands to modify the file protection – if requested for the password enter **blockchain**

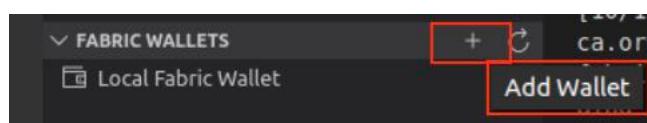
```
sudo chown -R blockchain:blockchain wallet/
sudo chown -R blockchain:blockchain nodes/
```

```
blockchain@ubuntu:~/workspace/hlf-ansible-master$ sudo chown -R blockchain:blockchain wallet/
blockchain@ubuntu:~/workspace/hlf-ansible-master$ sudo chown -R blockchain:blockchain nodes/
blockchain@ubuntu:~/workspace/hlf-ansible-master$
```

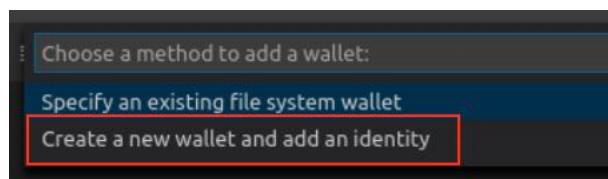
The network is now created and running, it is time to create wallets and import identities into them. This will allow us to access the network from within VS Code.

We will first create the wallet and import identities for **EcoBank**

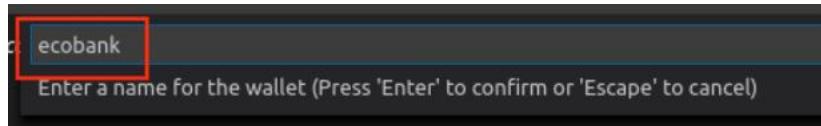
- 11. Back in VS Code, hover the mouse over the ‘+’ sign in the Fabric Wallets panel and click **Add Wallet**.



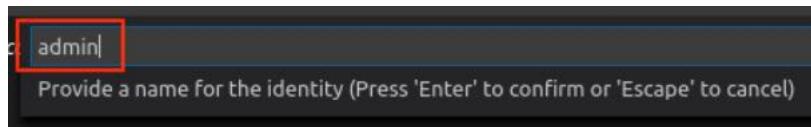
- 12. At the top of the screen click the option to **Create a new wallet and add an identity**



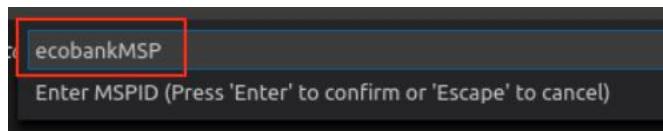
- **13.** Specify **ecobank** as the name of the wallet (it is important to name this exactly with all lowercase letters)



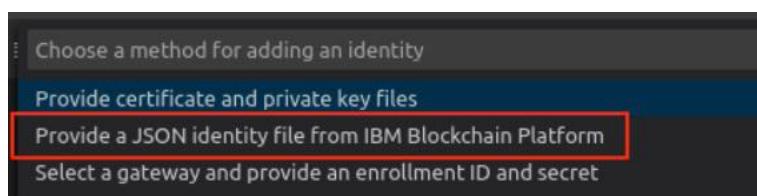
- **14.** Specify **admin** as the name of the identity



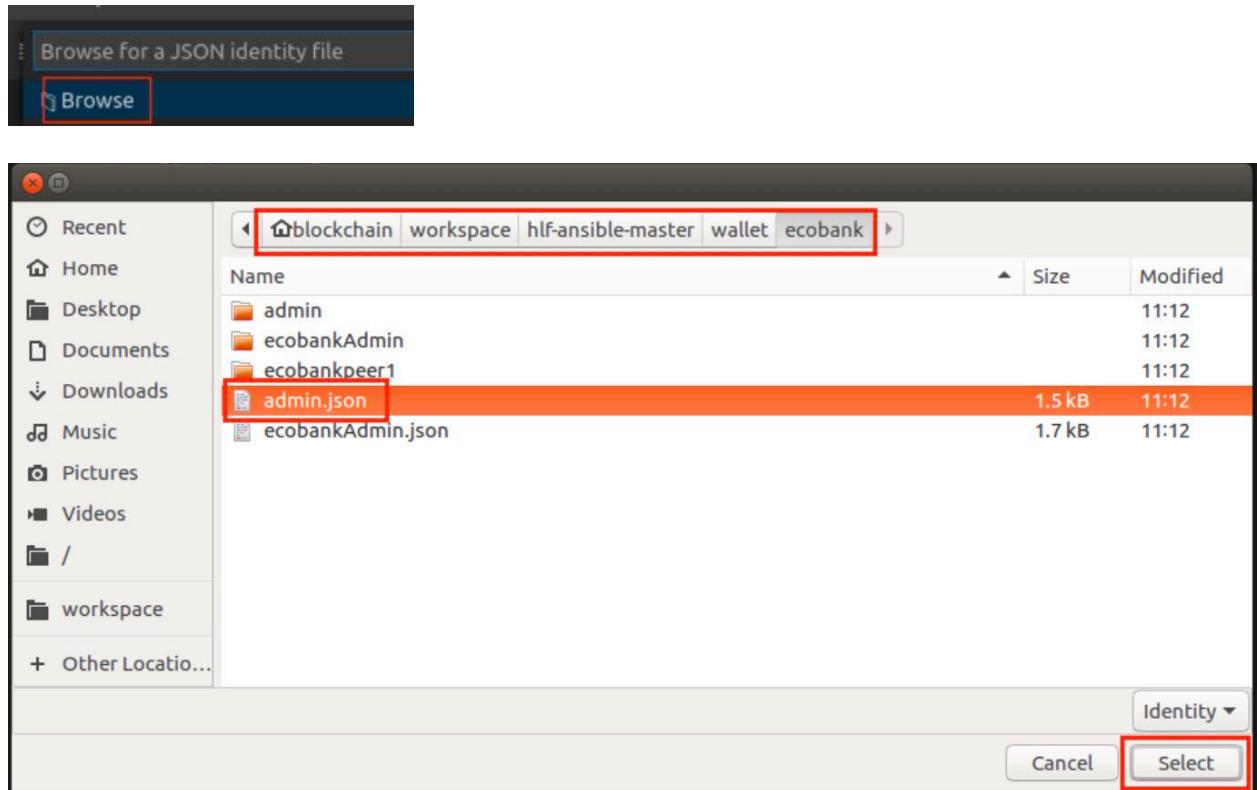
- **15.** Specify **ecobankMSP** as the MSPID



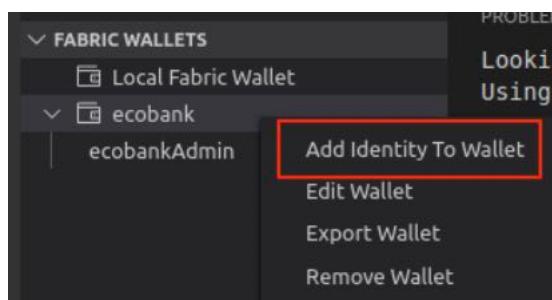
- **16.** Select the option to Provide a JSON identity from IBM Blockchain Platform



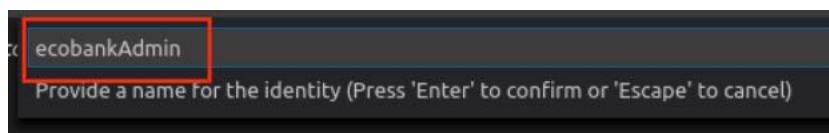
- 17. Click **Browse** and then navigate to the location:  
“Home” > workspace > hlf-ansible-master > wallet > ecobank  
Click the file **admin.json** to highlight it, then click **Select**



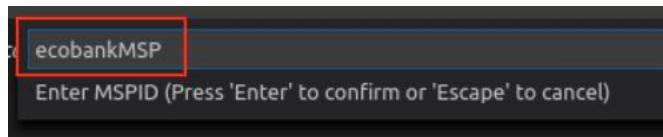
- 18. Right click the ecobank wallet icon on the **ecobank** wallet to add another identity



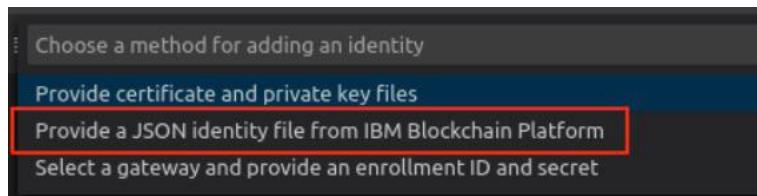
- 19. Specify **ecobankAdmin** as the identity name and press enter



-- 20. Specify **ecobankMSP** as the MSPID



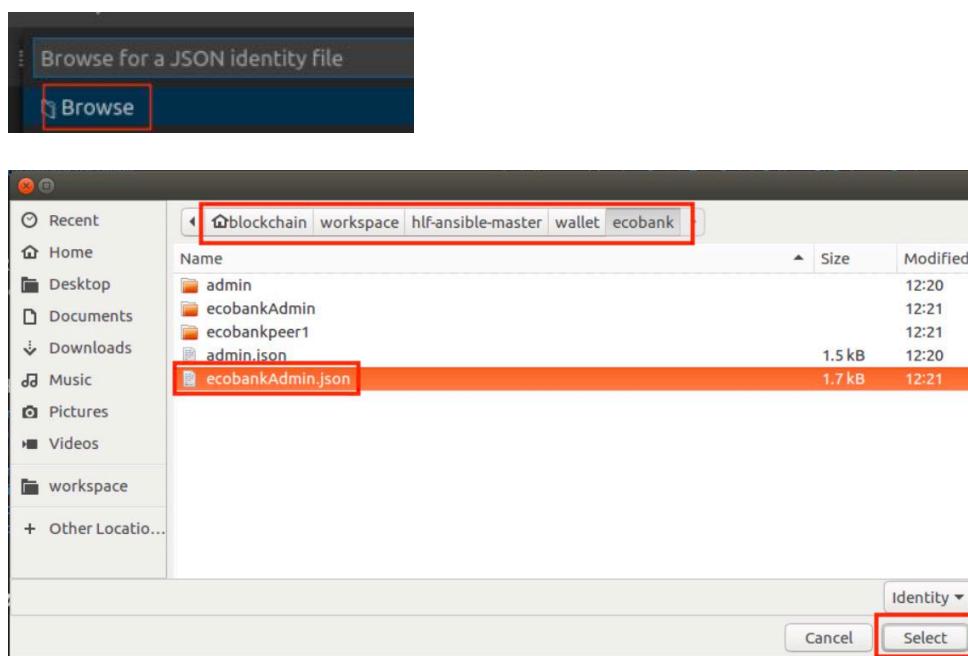
-- 21. Select the option to Provide a JSON identity from IBM Blockchain Platform



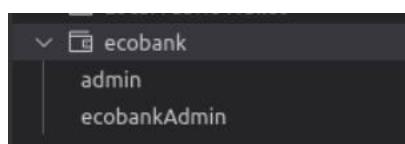
-- 22. Click **Browse** and then navigate to the location:

"Home" > workspace > hlf-ansible-master > wallet > ecobank

Click the file **ecobankAdmin.json** to highlight it, then click **select**



The newly created wallet and two imported identities will be shown in the Fabric Wallets panel:



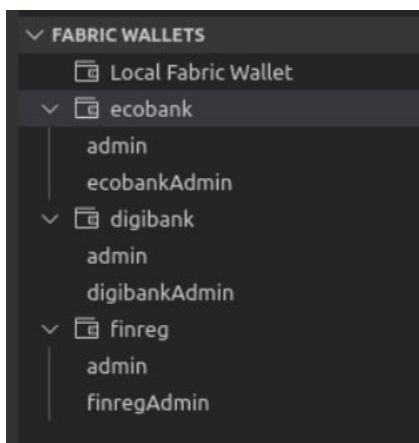
- **23.** Using Steps 12 – 23 as a guide create a wallet called **digibank** and import the IDs **admin** and **digibankAdmin**. The MSPID will be **digibankMSP**. Similar to the other wallet, remember to be exact with the names used. The folder to browse for the JSON file will be:

**“Home” > workspace > hlf-ansible-master > wallet > digibank**

- **24.** Using Steps 12 – 23 as a guide create a third wallet called **finreg** and import the IDs **admin** and **finregAdmin**. The MSPID will be **finregMSP**. Similar to the other wallets, remember to be exact with the names used. The folder to browse for the JSON file will be:

**“Home” > workspace > hlf-ansible-master > wallet > finreg**

When all the wallets are created and the identities imported, they will display as follows:



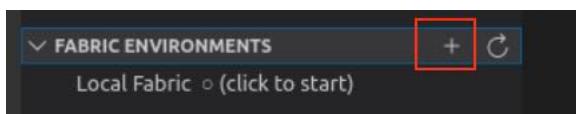
The wallets are now created, and the identities imported.

In order to install and instantiate smart contracts it is necessary to create **Fabric Environments** for each of the 3 main organisations in the consortium.

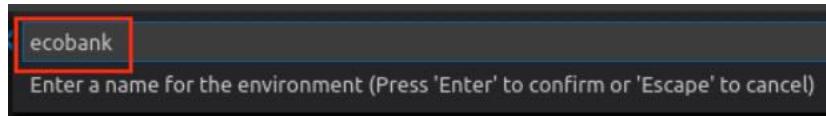
We will first create the Fabric Environment for EcoBank.

- **25.** Import “Nodes” file to create a Fabric Environment for EcoBank.

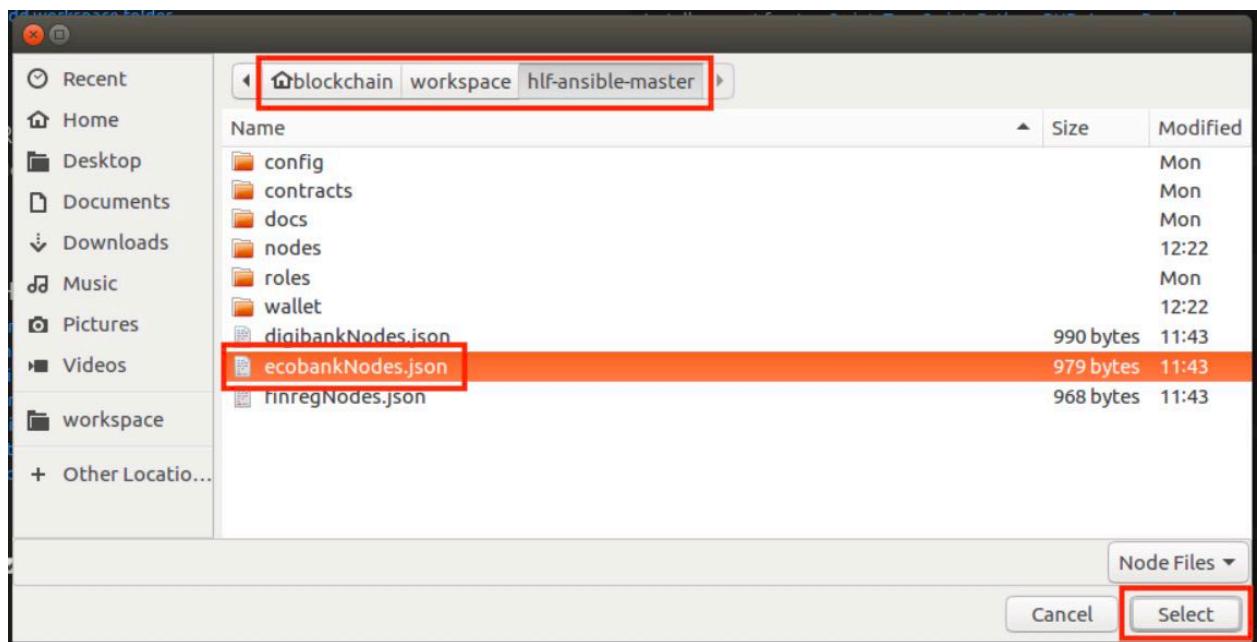
On the Fabric Environments Panel click the ‘+’ icon to add a new environment.



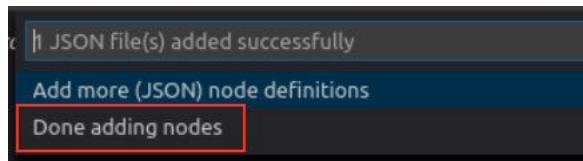
- 26. At the top of the screen type **ecobank** as the name of the new environment and press enter.



- 27. Click **Browse** and then navigate to the location:  
“Home” > workspace > hlf-ansible-master  
Click to highlight **ecobankNodes.json** click **Select**

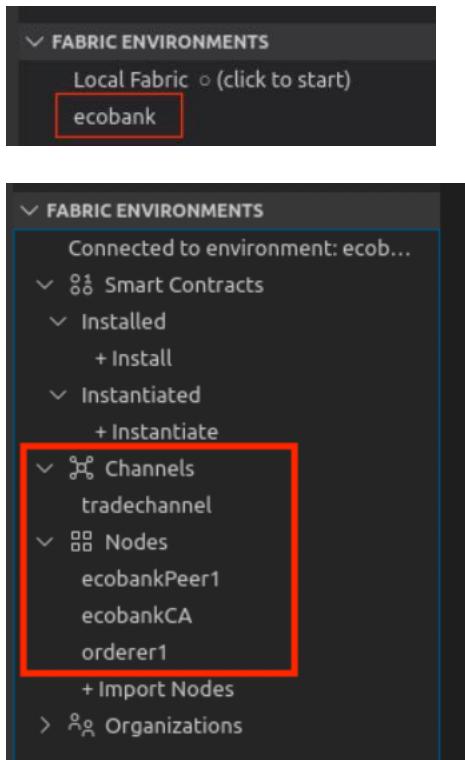


- 28. The JSON file you selected contains definitions for Peer, Certificate Authority and Orderer, so there are no more files to import for EcoBank. At the top of the screen 1 file is shown as imported – click **Done adding nodes**

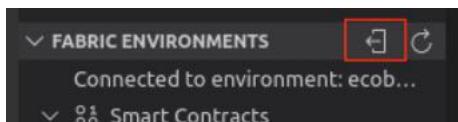


**Note:** The imported Node file included the name of the wallet and the identity to use, so when clicking this environment it will connect immediately.

- **29.** Click on the newly created **ecobank** environment to check that it connects as expected, and expand the **Channels** and **Nodes** twisties.



- **30.** Hover over the Fabric Environments panel, and click the icon to disconnect from the Ecobank environment



- **31.** Using steps 25 – 30 as a guide, create a Fabric Environment for **digibank** using the file “Home” > workspace > hlf-ansible-master > **digibankNodes.json**.

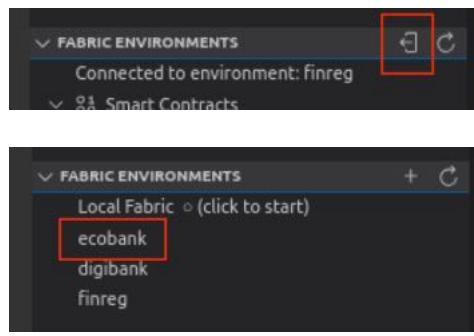
- **32.** Again, using steps 25 – 30 as a guide, create a Fabric Environment for **finreg** using the file “Home” > workspace > hlf-ansible-master > **finregNodes.json**.

These environments will now be used to create additional (non-administrator) IDs to allow more realistic testing of the smart contract. These IDs are

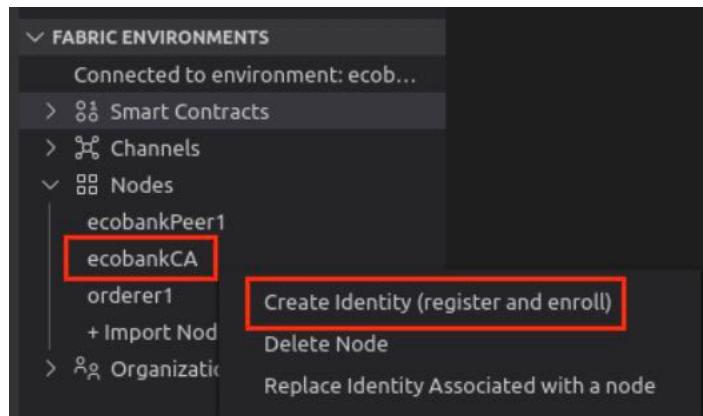
Eva for EcoBank  
David for DigiBank and  
Fran for FinReg

We will start by creating the ID for Eva.

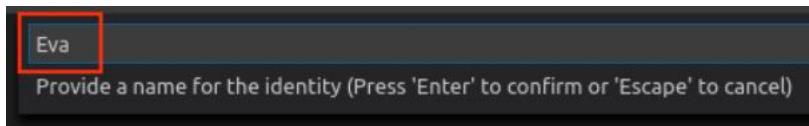
- **33.** Disconnect from any connected Fabric Environment and click on **ecobank** to connect to the EcoBank environment.



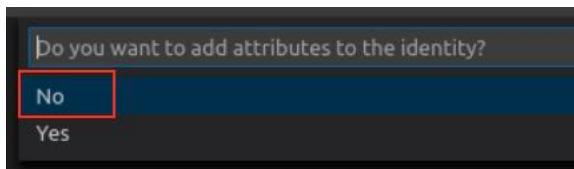
- **34.** Expand the **Nodes**, right-click on the **ecobankCA** and then click **Create Identity (register and enrol)**



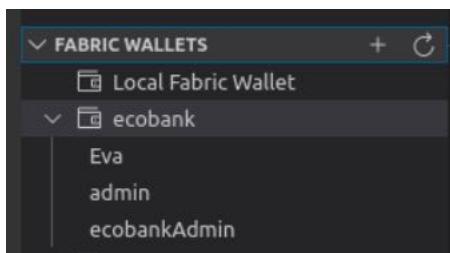
-- 35. Specify **Eva** as the identity



-- 36. There are no attributes required for this lab – click **No**



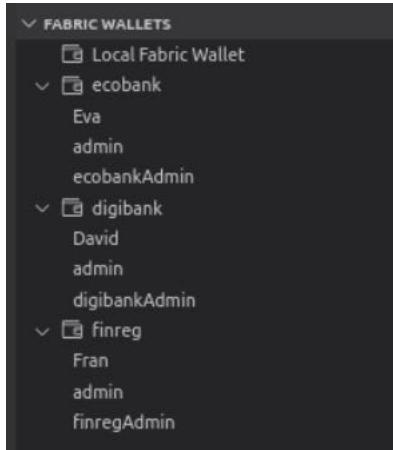
The Identity **Eva** has been added to the **ecobank** wallet



-- 37. Using steps 33 – 36 as a guide, connect to the **digibank** environment and Create and Identity **David**

-- **38.** Using steps 33 – 36 as a guide, connect to the **finreg** environment and Create and Identity **Fran**

The complete list of wallets and identities for the 3 organisations is shown below

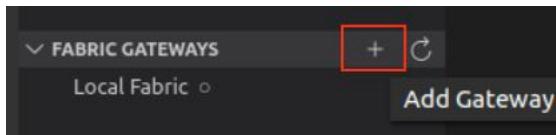


The identities we have just created will connect to the network using **Gateways**.

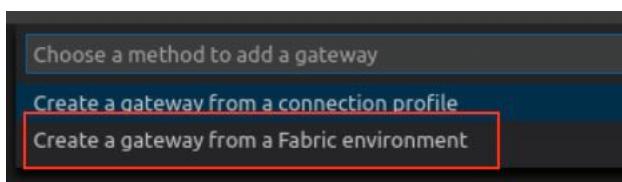
We'll first add the gateway for Ecobank.

-- **39.** Add the Gateway for **ecobank**

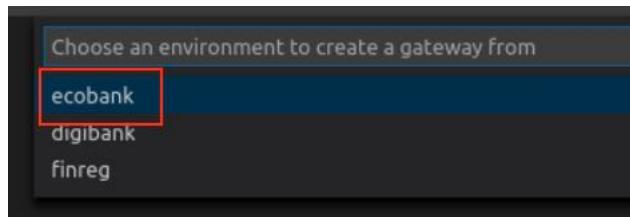
Hover over Fabric Gateways and click the '+' to add a gateway



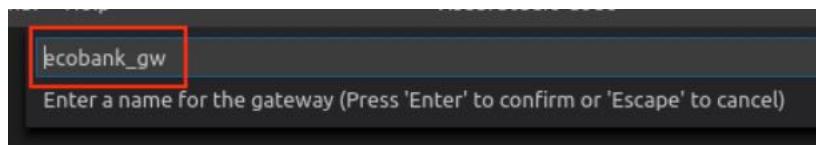
-- **40.** The simplest way to create a Gateway is to base the definition on an existing Environment. Click **Create a gateway from a Fabric environment**



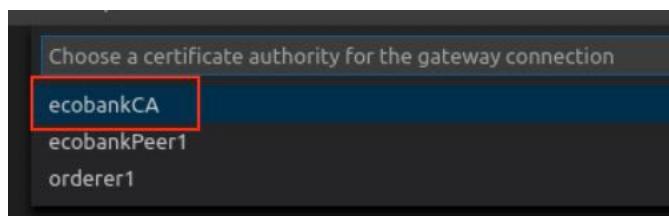
-- 41. Click **ecobank** as the environment to copy from



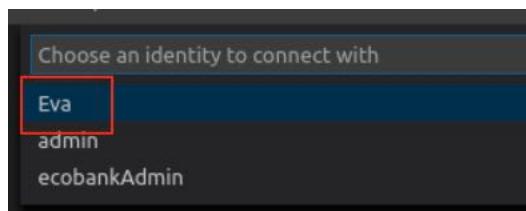
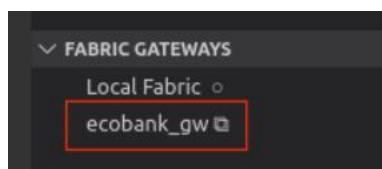
-- 42. Use the suggested name **ecobank\_gw** and press **enter**



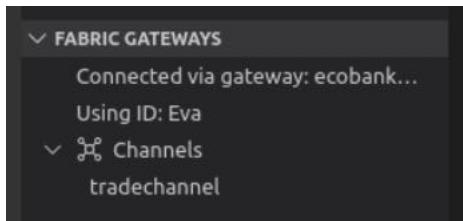
-- 43. Click the **ecobankCA** as the certificate authority



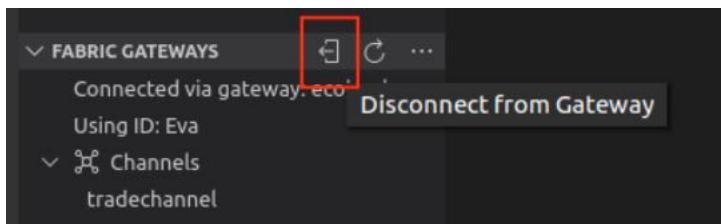
-- 44. Click on the new gateway **ecobank\_gw** and choose **Eva** as the identity to connect with



-- **45.** Verify that Eva can see the **tradechannel**



-- **46.** Hover over the Fabric Gateways panel and **click** the **Disconnect** icon to disconnect from the gateway



-- **47.** Using steps 39 – 46 as a guide, add a **digibank** gateway called **digibank\_gw** and connect with the identity **David**. Verify that David can see the tradechannel channel.

-- **48.** Using steps 39 – 46 as a guide, add a **finreg** gateway called **finreg\_gw** and connect with the identity **Fran**. Verify that Fran can see the tradechannel channel.

## Review

In this section you have:

- Used a playbook (site.yml) to create a network
- Created wallets and imported administrative users
- Created environment definitions that allowed the administrators to operate the network
- Created additional non-administrative users for more realistic testing
- Created gateway definitions that allowed the non-administrative user IDs to be used to test transactions when the smart contract was deployed.

## 2 Deploy the first version of the Smart Contract to test & prove the Trade Network

The network has been created and a first version of the smart contract can be deployed and tested. The process is to package the Smart Contract, then install it on peers in the network, and finally instantiate it once on the channel. The instantiation step takes a few moments to build the new chaincode container.

The smart contract contains code that handles private data, something which was introduced to Hyperledger Fabric as far back as version 1.2. It was introduced to enable organisations to keep data private from other organisations on a channel, whether as one organisation or in a private arrangement with another organisation. This functionality was designed, among other things, to prevent the need to create a separate channel to achieve the same end.

Let's briefly describe two key entities essential to understanding Fabric Private Data:

The first is a **Private Data collection** – the second, is a **Private Data collection definition file**. **Private Data collections** allow an organisation – or a subset of organizations on a channel - the ability to endorse, commit, or query private data without having to create a separate channel. Collections consist of **private data** sent peer-to-peer to only the organisations entitled to see it (by policy, defined in a collection definition file) – and – a **hash of the data** is also endorsed, ordered, and written to the ledgers of every peer on the channel. The hash serves as evidence of the transaction, i.e. for state validation or audit/compliance purposes.

The second is **Private Data collection definition files**. These are policy-based definitions that describe which organisation(s) belong to a Private Data collection – it could be one, two or more organisations (as policy dictates), that govern who can access private data in a given collection, on a channel. It is commonplace for the definition file to contain one or more collections (each has a name and the participating organisations in that stanza), as well as properties used to control dissemination of private data at endorsement time and, optionally, whether the data will be purged / its longevity. The collection definition then gets deployed to the channel, supplied as a policy file when instantiating/upgrading the chaincode package.

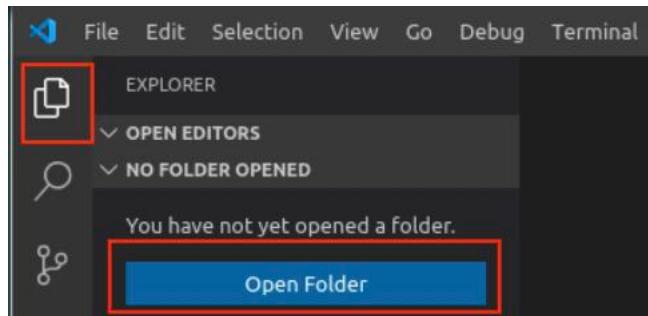
When instantiating this smart contract you will be supplying the definition of the Private Data Collections.

The definition file for this lab is shown below:

```
[{"name": "CollectionD", "policy": {"identities": [{"role": {"name": "member", "mspId": "digibankMSP"}}], "policy": {"1-of": [{"signed-by": 0}]}}, {"requiredPeerCount": 1, "maxPeerCount": 1, "blockToLive": 0, "memberOnlyRead": true}, {"name": "CollectionE", "policy": {"identities": [{"role": {"name": "member", "mspId": "ecobankMSP"}}], "policy": {"1-of": [{"signed-by": 0}]}}, {"requiredPeerCount": 1, "maxPeerCount": 1, "blockToLive": 0, "memberOnlyRead": true}]
```

Complete details of the syntax and meaning of all the fields is available in the Fabric Documentation, but note that the two Collections defined in this lab exercise have collection names of **CollectionD** for Digibank and **CollectionE** for Ecobank. (The “Policy” section in the JSON file is the same format as that used for Endorsement Policies)

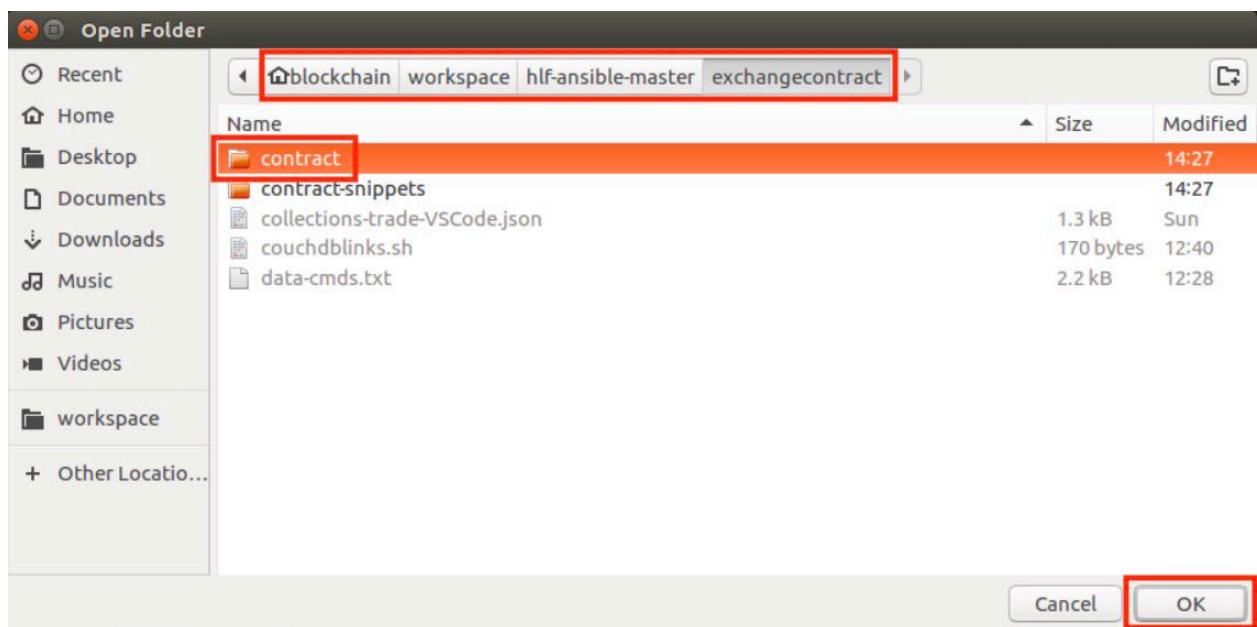
-- **49.** Click the Explorer icon in VS Code and **Open Folder**



Navigate to the location:

“Home” > workspace > hlf-ansible-master > exchangecontract

**Click** to highlight the **contract** folder and click **OK**



-- **50.** VS Code will display the **Contract folder** tree. Expand the twisty for the **lib** folder and click **trading-contract.js** to view it.

**Note 1:** Notice the location of the source ‘trading-contract.js’ base contract under ‘lib’ in Explorer - you will need it later to add additional transactions.

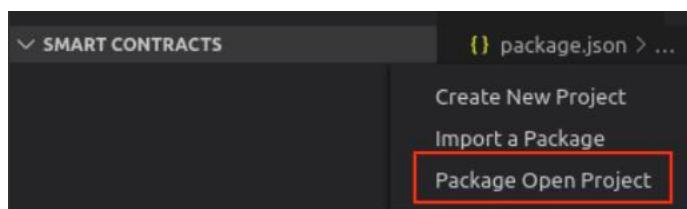
**Note 2:** This Smart Contract includes the Fabric new programming model Contract API. (`fabric-contract-api`)

```

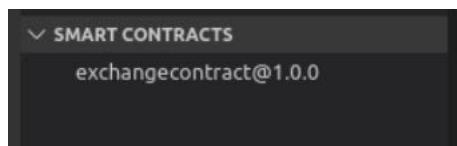
File Edit Selection View Go Debug Terminal Help
trading-contract.js - Visual Studio Code
EXPLORER
OPEN EDITORS
trading-contract.js lib
CONTRACT
> .vscode
lib
trading-contract.js
test
.editorconfig
.eslintignore
.eslintrc.js
.gitignore
.npmignore
index.js
package-lock.json
package.json
JS trading-contract.js
1  /*
2   * SPDX-License-Identifier: Apache-2.0
3   */
4
5 'use strict';
6
7 const { Contract, Context } = require('fabric-contract-api');
8
9 // GLOBALS - as this is meant to be a simple lab exercise, we'll leave them here for now - easy to reference
10 const Namespace = 'org.example.marketplace';
11 const Assetspace = Namespace + '.Trade';
12
13 class TradingContract extends Contract {
14
15     // MAIN CONTRACT starts here, after all the 'preliminaries' ....
16
17     /**
18      * Instantiate the contract ('start' - short/easy name to type for the instantiation function call in the lab :-)
19      * @param {Context} ctx the transaction context
20     */
21
22     async start(ctx) {
23         console.log('Instantiating the contract ...');
24         let invokingMSpid = await this._getInvokingMSpid(ctx);
25         console.log(`instantiated by an id from MSP ${invokingMSpid}`); // written to the container
26         return 'DONE as MSP ' + invokingMSpid; // will see this in VS Code output pane
27     }
28
29     // UTILITY functions that may prove useful at some point

```

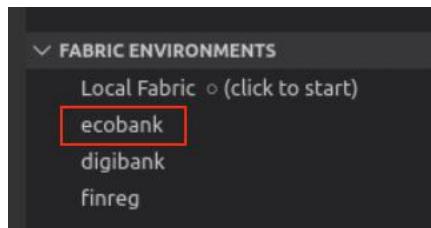
- 51. Click the icon to return to the IBM Blockchain Platform extension, hover over the and under **Smart Contracts** panel, click the Ellipses icon to select “More actions” and then select **Package Open Project**



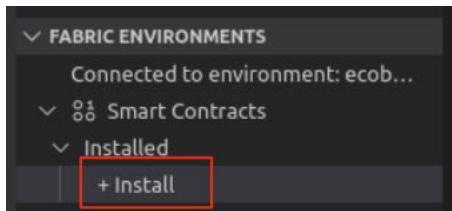
This will cause the smart contract you just imported to be packaged into a CDS file that can be installed on a peer. The package will then appear in the Smart Contracts list.



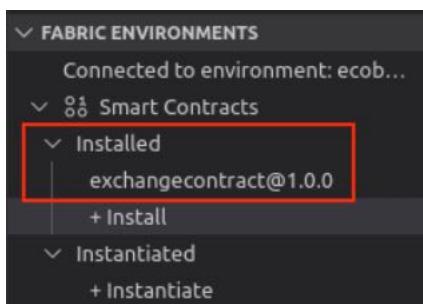
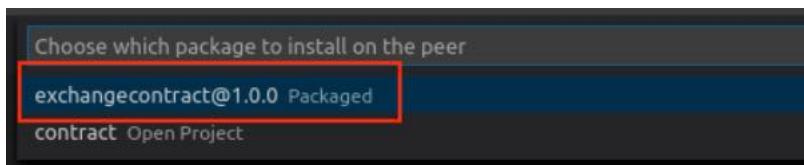
- 52. On the Fabric Environments panel click on **ecobank** to connect to the Ecobank environment:



- 53. Click **+Install** to install a contract on the peer



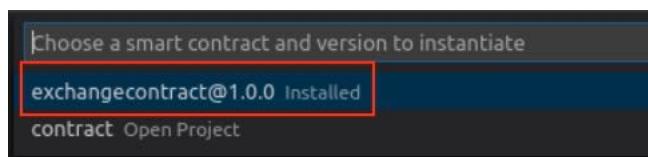
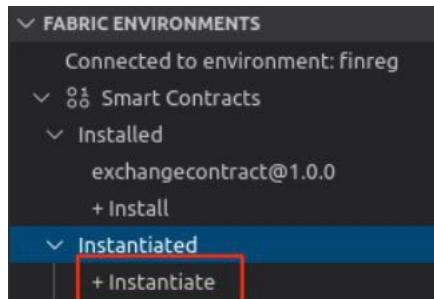
Select **exchangecontract@1.0.0** The contract will appear under the installed list



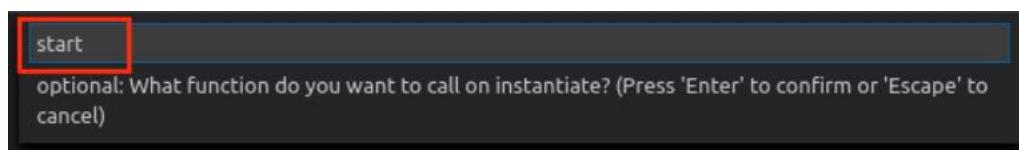
- 54. Disconnect from the Ecobank Fabric Environment, and using Steps 52 -53 as a guide, install the same smart contract in the **digibank** environment.

- 55. Disconnect from the Digibank Fabric Environment, and using Steps 52 -53 as a guide, install the same smart contract in the **finreg** environment.

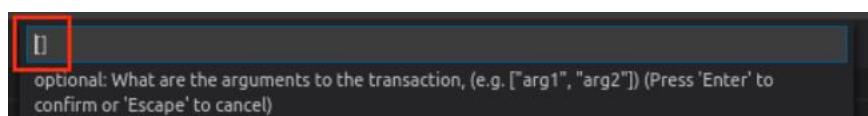
- 56. While still connected to the finreg environment, click **+Instantiate** to instantiate the contract on the finreg peer, and select **exchangecontract@1.0.0**



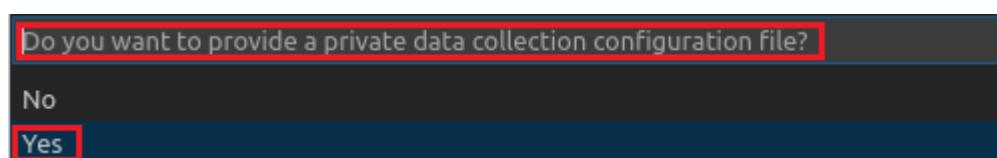
- 57. Specify **start** as the function to be used with instantiate



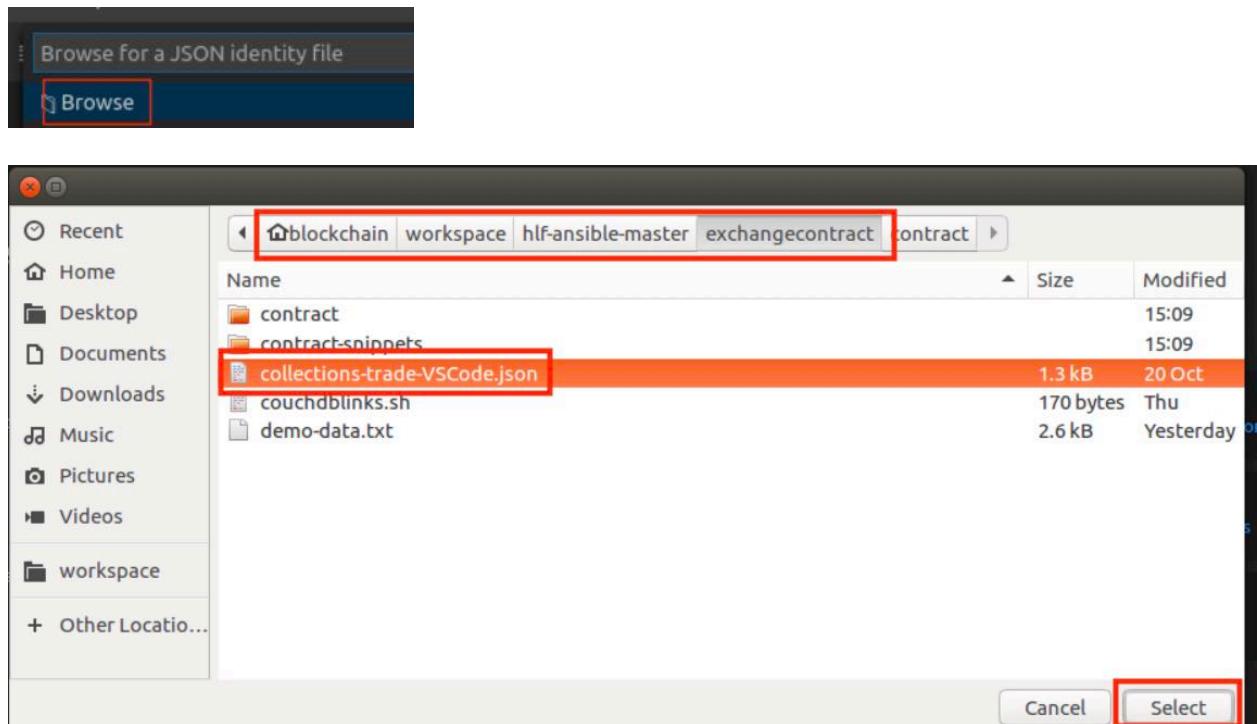
- 58. There are no parameters to pass to the start function – just press **enter** with the default empty array **[]**



- 59. When asked if you ‘want to provide a private data collection configuration file?’ – select **Yes**

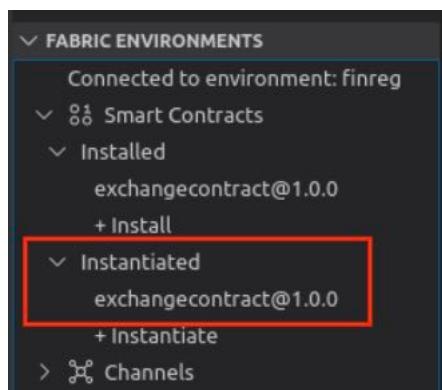


- 60. Click **Browse** and navigate to the folder:  
“Home” > workspace > hlf-ansible-master > exchangecontract  
and select the file **collections-trade-VSCode.json**.



This will take a minute or so to complete.

When the contract is instantiated it will show as follows



The smart contract has been instantiated on the **tradechannel** channel.

This is the end of Part 2 of the Lab.

**Review**

In this part of the Lab you have:

- Packaged the base contract
- Installed the base contract on peers for the 3 organisations
- Instantiated the Smart Contract on the FinReg peer (starting a chaincode container)
- Instantiated the Smart Contract with the definition of the Private Data Collections

## 3 Private Data for the Offer and Accept Transactions

### INTRODUCTION

In this section, we will focus on the logic for the **offer** and **accept** transactions in our private data scenario. We will additionally us an **advertise** transaction which will allow a commodities contract to be offered for sale on the blockchain network.

Part 3 of the lab is to execute these transactions so that we create records in both the channel ledger and most importantly in the private data collections. After an initial **advertize** function is executed, the next sequence is to invoke the new **offer / accept** functions – these are executed by identities from:

- DigiBank (the **buyer**, making the offer) and
- Ecobank (the **seller**, who chooses what offer to accept)

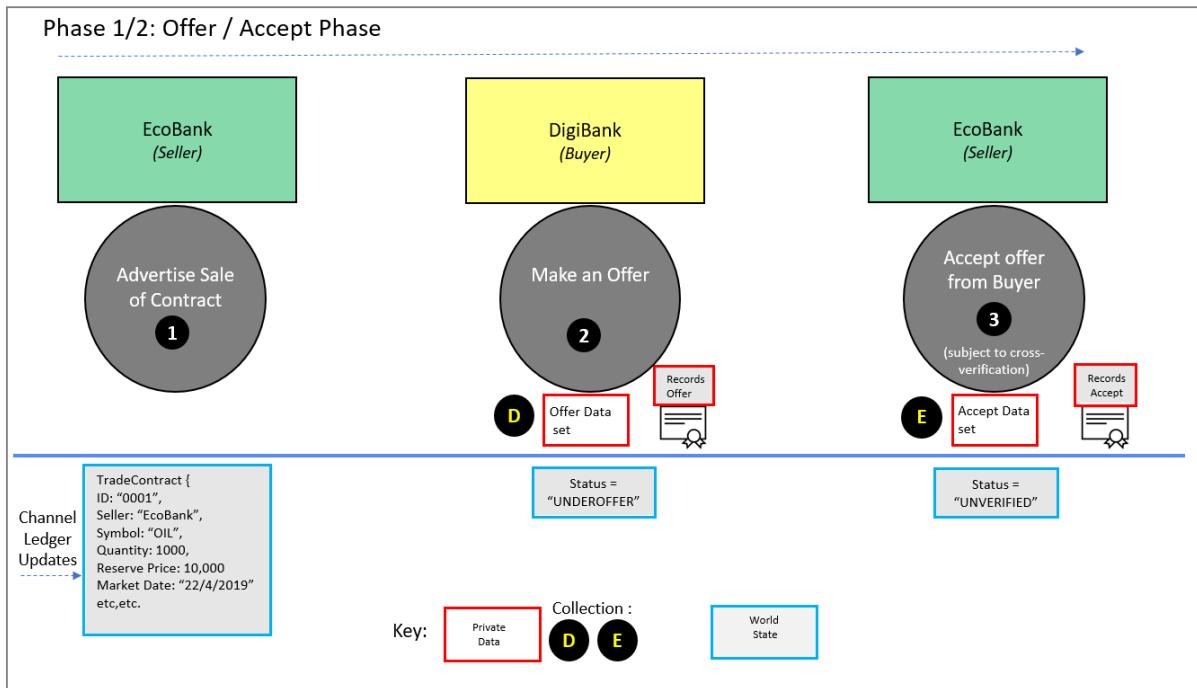
Both the offer and the accept transactions generate a request and create a private data record for the invoking identities' organisation. Note that the functions will also update the channel ledger with 'public' data like 'status' or 'date'. Later on, the offer / accept pattern will need to undergo a 'cross-verification' process – that will be done in Part 4.

For a complete list of the function names, their objectives and a brief description of each, please see the Appendix.

In the offer/accept phase of the scenario the following transactions occur:

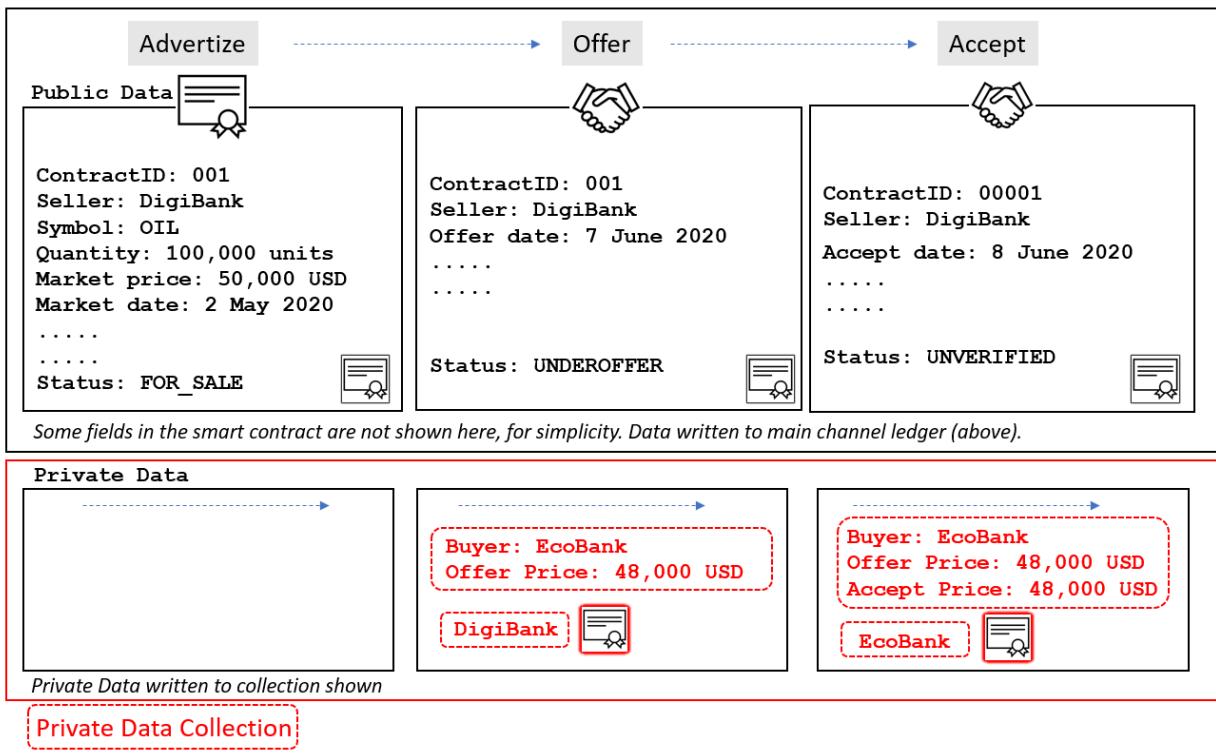
1. Seller **EcoBank** advertises the nominal sale price / info on the marketplace and it is recorded on the blockchain - this is written to the ledger and all would-be buyers get notified (via applications, out-of-band).
2. Buyer **DigiBank**, makes an *offer* to buy the contract – it formalises this offer by generating an offer request. The offer price is confidential, and so is written to its (DigiBank's) own private data store (i.e. one organisation in the collection) and shared only with Ecobank via an out-of-band application. Its status is 'UNDEROFFER' on the ledger (world state).
3. **EcoBank** who received the offer price out-of band – *accepts* the offer. It generates an accept approval request and it stores this in its own Private Data collection (one organisation). Its status (written to the channel ledger state) at this time is now set to 'UNVERIFIED'.

## Verify Pattern



The private data – and the ‘public’ (channel ledger) data being recorded, is best captured in the diagram below:

## Commodity Contract data: what's public, what's private ?



### Offer / Accept Phase – Introduction

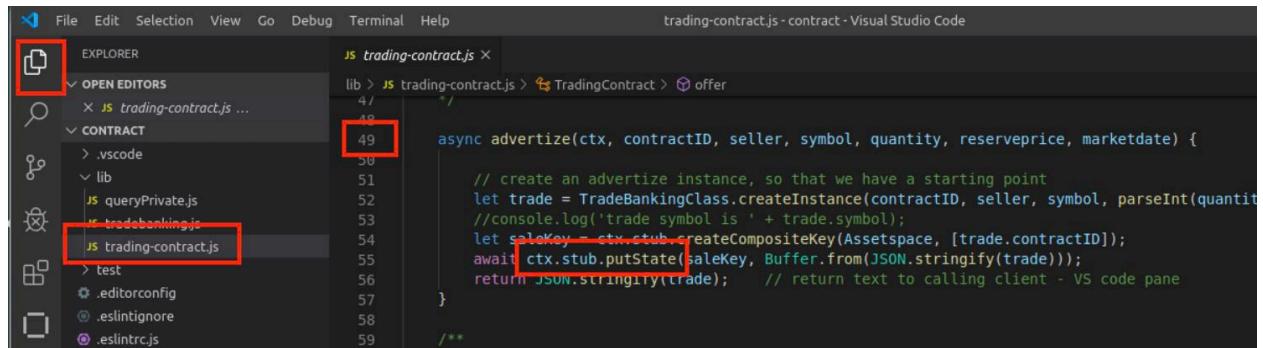
The first task is to examine the offer and accept transactions mentioned earlier. These are the main transactions that were added in the base contract, along with two helper methods called **readOffer** and **readAccept** which can be used to verify data written to private data collections (in the lab).

Input to the **offer** and **accept** transactions are passed as ‘transient data’; in the VS Code extension you’re prompted to provide this. Transient data is not persisted in the transaction that is stored on the channel ledger; this keeps the data confidential. Transient data is passed as binary data and then decoded and written to the local private data store.

**Note:** It is good and common practise to ‘salt’ this transient data with a random element, especially if it contains more predictable private data. See the appendix ‘Access Control and Salting of Private Data’ for more information.

Let’s briefly explore the functions starting with **advertize**.

- 61. Click the **Explorer** icon, and open the **trading-contract.js** from the folder view and find the **advertize** function at **line 49**.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar is open, displaying a tree view of files. A red box highlights the 'EXPLORER' icon. Another red box highlights the 'trading-contract.js' file under the 'CONTRACT' section. On the right, the main editor window shows the 'trading-contract.js' file. A red box highlights line 49 of the code, which contains the 'advertize' function definition.

```
JS trading-contract.js x
lib > JS trading-contract.js > TradingContract > offer
4/
49
async advertize(ctx, contractID, seller, symbol, quantity, reserveprice, marketdate) {
50
    // create an advertize instance, so that we have a starting point
51    let trade = TradeBankingClass.createInstance(contractID, seller, symbol, parseInt(quantity));
52    //console.log('trade symbol is ' + trade.symbol);
53    let saleKey = ctx.stub.createCompositeKey(AssetSpace, [trade.contractID]);
54    await ctx.stub.putState(saleKey, Buffer.from(JSON.stringify(trade)));
55    return JSON.stringify(trade); // return text to calling client - VS code pane
56
57
58
59
/**
```

The **advertize** transaction creates a saleable Commodity contract (identified by a contract ID) on the channel ledger which is traded on the marketplace. This uses the Fabric **putState() API** that writes state information to the ledger. This transaction function is called by someone from EcoBank (the seller).

**62.** Scroll to the **offer** transaction at **line 65**.

You can ignore the “DEMO” clause between lines 72 to 87, which can be used to create sample data.

The offer transaction takes a contract ID as a parameter and reads in the ‘transient data’ as private data - supplied when the transaction is invoked from the VS Code extension or a client app. Transient data is passed as binary data and then decoded in the function starting with the `getTransient` method.

**Lines 88-97** show the private data elements being committed to DigiBank's private collection using the chaincode API **PutPrivateData()** in **line 124**

**Line 120** causes the non-confidential data to be written to the main ledger using `putState()` – data like ‘Status’ or an ‘Offer Date’ – which is intentionally visible to others on the channel.

```
js trading-contract.js
lib > js trading-contract.js > TradingContract > offer
63
64
65 async offer(ctx, contractID) {
66
67     console.log('retrieving the transient data (1st) and then (2nd) the advertised contract (fn: offer)' + contractID);
68
69     let offerdate = '';
70     let priv_data = {};
71
72     if (contractID === "DEMO") {
73     }
74     const tranMap = ctx.stub.getTransient();
75     if (tranMap.size === 0) return 'Error: you didnt provide any transient data?? Try calling again, with transient data in the {} provided';
76
77     // get transient data supplied and set up private data write set
78     tranMap.forEach((value, key) => {
79         let dataval = new Buffer(value.toArrayBuffer()).toString();
80         if (key === 'privatebuyer') priv_data[key] = dataval;
81         if (key === 'privateprice') priv_data[key] = parseInt(dataval);
82         if (key === 'offerdate') offerdate = dataval; // its not going to private data - supplied to transient for convenience
83     });
84
85     let offerKey = ctx.stub.createCompositeKey(Assetspace, [contractID]);
86     let offerBytes = await ctx.stub.getState(offerKey); // getState goes to 'public' ledger pls note
87     if (offerBytes.length > 0) {
88         var offer = JSON.parse(offerBytes);
89         console.log('retrieved contract');
90         console.log(offer);
91     }
92     else {
93         console.log('Nothing advertised with that Key: ', contractID);
94         var offer = "EMPTY CONTRACT (offer function)";
95         return offer;
96     }
97
98     // private data is supplied as a param to VS Code - eg equivalent to ..
99     // let priv_data = { privatebuyer: buyer, privateprice: parseInt(offerprice) };
100    // first, write offerdate, and a hash of priv_data, to channel ledger using putState
101    offer.offerdate = offerdate; // the offer date is actually 'ledger-public' - let's write that first ...
102    offer.status = "UNDEROFFER";
103    // OPTIONAL LAB:
104    // Uncomment this single line - if you're completing the OPTIONAL lab (see lab exercise instructions)
105    offer.offerhash = crypto.createHash('sha256').update(JSON.stringify(priv_data)).digest("hex");
106
107    await ctx.stub.putState(offerKey, Buffer.from(JSON.stringify({ offer })));
108
109    // second..write the actual private_data (priv_data) to the private data collection !
110
111    await ctx.stub.putPrivateData(CollectionD, offerKey, Buffer.from(JSON.stringify(priv_data)));
112    //return Buffer.from(JSON.stringify(priv_data)); // return buffer to calling client
113    return JSON.stringify(priv_data); // return the data in the output pane/terminal, for lab invoker's reading..
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129 }
```

-- **63.** Scroll to the implementation of the **accept** transaction at **Line 157**

The accept method has the same format as the offer function, decoding the transient data (**lines 179-190**) then writing Private Data at **line 212** and writing main channel data at **line 208**.

The **accept** transaction represents an accepting party (EcoBank, the seller) completing the sale, by accepting an offer – in this scenario from DigiBank. Similar to the **offer** transaction, it also writes private data – but this time to EcoBank’s own private collection. It writes an accept price privately in addition to the, as well as the offer data set that was agreed with DigiBank. It also makes use of **putPrivateData()** to write confidential data and **putState()** to update the main channel ledger with non-confidential data as before.

-- **64.** Review the additional helper functions in the contract:

The read-only transaction **readOffer** is a helper function, to enable you to verify what was written to private data earlier.

The read-only transaction **readAccept** is a helper function, to enable the seller identity, to verify what was written in the private collection, when created in the ‘accept’ call earlier.

The remaining functions in the smart contract - like **\_getInvokingMSP** are utility functions, used invariably by the main transaction functions.

### 3.1 Invoking the Offer and Accept transactions

At this point we can now go and create some private data on each of the collections we are using.

However, we will first launch the CouchDB web interface for each organisation into a separate Firefox tab which will enable us to view the data in the tradechannel world state, and the private collections.

Note also that some of the parameter lists that you need to ‘paste’ into VS Code (to complete a task), can be copied and pasted from the file **demo-data.txt** which is located under the **workspace/hlf-ansible-master/exchangecontract** folder.

-- **65.** From a terminal window, use the following two commands to navigate to the **exchangecontract** folder and launch CouchDB sessions in Firefox using a bash script:

```
cd ~/workspace/hlf-ansible-master/exchangecontract  
./couchdblinks.sh
```

```
blockchain@ubuntu:~/workspace/hlf-ansible-master$  
blockchain@ubuntu:~/workspace/hlf-ansible-master$ cd ~/workspace/hlf-ansible-master/exchangecontract  
blockchain@ubuntu:~/workspace/hlf-ansible-master/exchangecontract$ ./couchdblinks.sh  
starting Firefox with tabs for CouchDB utils  
blockchain@ubuntu:~/workspace/hlf-ansible-master/exchangecontract$
```

When the script is complete you should see a Firefox browser window with three tabs displayed. Note that the script may take a couple of minutes to complete, and there will be a pause between tabs opening.

**Note:** If the tabs have not opened after three minutes close the firefox window and re-run the command.

Each tab represents the view for a different organization:

- Digibank is displayed on localhost:6984/\_utils
- EcoBank is displayed on localhost:5984/\_utils
- Finreg is displayed on localhost:7984/\_utils

**Note:** In a production scenario direct access to the CouchDB Web interface would be blocked for security, but it is useful during development and testing.

The screenshot shows a Mozilla Firefox window with three tabs open, all titled "Project Fauxton". The tabs are labeled "DigiBank", "EcoBank", and "FinReg". The "DigiBank" tab is active. On the left side of the screen, there is a sidebar for "Project Fauxton" which includes a "DB List Icon" (highlighted with a red box). Below the icon, there is a table listing databases. The table has columns for "Name", "Size", "# of Docs", and "Actions". The listed databases are: "\_replicator" (2.3 KB, 1 doc), "\_users" (2.3 KB, 1 doc), and "tradechannel\_" (20.2 KB, 2 docs). Each database entry has a set of icons under the "Actions" column.

Name	Size	# of Docs	Actions
_replicator	2.3 KB	1	[icons]
_users	2.3 KB	1	[icons]
tradechannel_	20.2 KB	2	[icons]

In the steps in this section of the lab, some of the parameters are quite long. There is a text file available to make the copy and paste of the parameters easier.

66. From the terminal window issue the command to open the text file in gedit

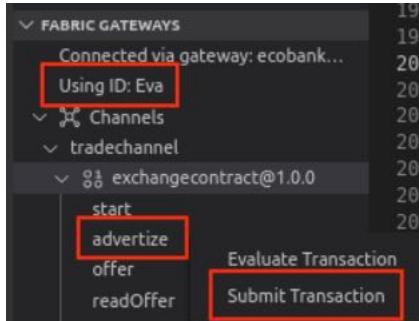
```
gedit demo-data.txt &
```

```
blockchain@ubuntu:~/workspace/hlf-ansible-master/exchangecontract$  
blockchain@ubuntu:~/workspace/hlf-ansible-master/exchangecontract$ gedit demo-data.txt &
```

You can copy and paste from this text file in the steps that follow.

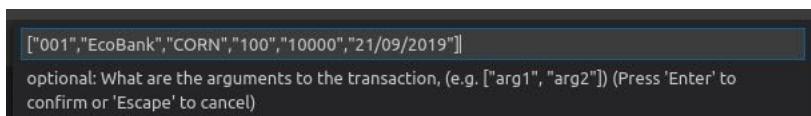
-- **67.** Switch back to the IBM Blockchain Platform VS Code extension, and in the Fabric Gateways panel click **ecobank\_gw** and connect it using the identity **Eva**

-- **68.** Expand Channels and tradechannel and then right click the **advertize** transaction and click **Submit transaction**.



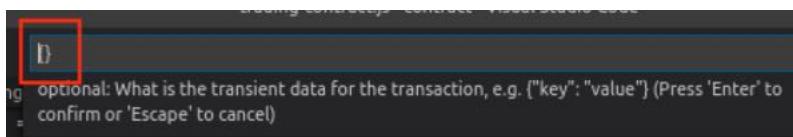
-- **69.** Next, enter the following parameters for arguments when prompted – overwrite the '[]' brackets offered in the prompt

```
[{"001","EcoBank","CORN","100","10000","21/09/2019"]
```



As you will recall from the `advertise()` implementation, the parameters represent the contract ID, seller, symbol, quantity, reserve price and market date respectively.

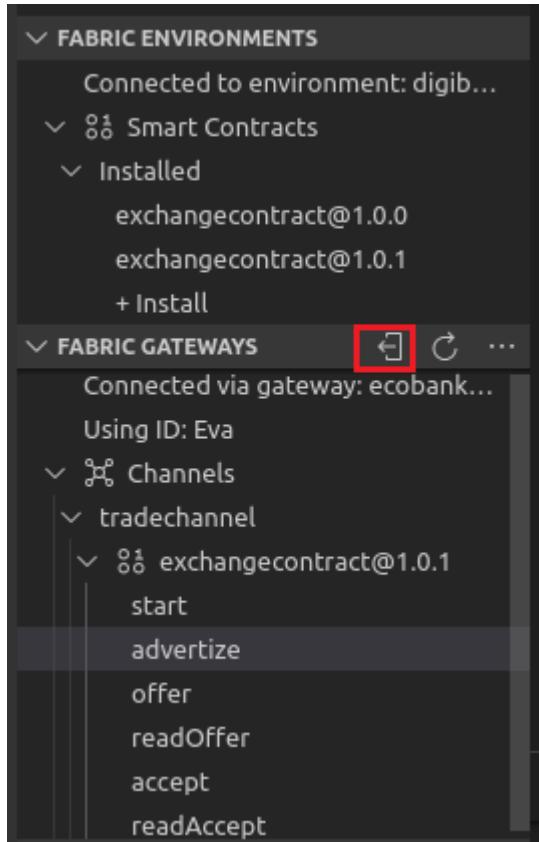
-- **70.** There is no transient data for this function, just press **enter** leaving the empty curly braces **{}**



the transaction will now be submitted, and should return with a SUCCESS message

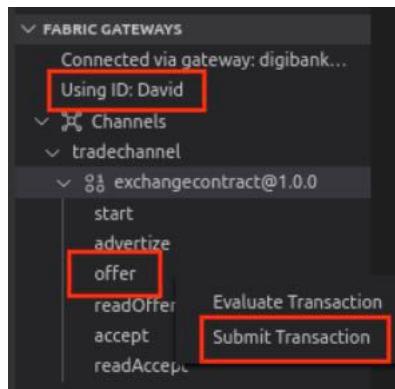
```
[10/23/2019 6:15:27 PM] [INFO] submitTransaction
[10/23/2019 6:16:13 PM] [INFO] submitting transaction advertize with args 001,EcoBank,CORN,100,10000,21/09/2019 on channel tradechannel
[10/23/2019 6:16:44 PM] [SUCCESS] Returned value from advertize: {"contractID":"001","seller":"EcoBank","symbol":"CORN","quantity":100,"reserveprice":10000,"marketdate":"21/09/2019"}
```

-- **71.** Next, we will make an offer for the advertised commodity with a Contract ID of "001" – to do this, we must first disconnect from the EcoBank gateway – **click** on the **disconnect** icon



-- 72. Next, click on the DigiBank gateway (**digibank\_gw**) – when prompted, select **David** as the identity

-- 73. Right-click on the **offer** transaction and click **Submit Transaction**



-- 74. When prompted for arguments, enter the following parameter (overwriting any existing '[]' data) – and press **enter**

["001"]

```
["001"]
optional: What are the arguments to the transaction, (e.g. ["arg1", "arg2"]) (Press 'Enter' to confirm or 'Escape' to cancel)
```

- **75.** Next, you will be asked to supply ‘transient data’. Our private offer is confidential and so we want to use this field to supply this information so that it is not captured on the ledger. Enter or paste the following overwriting the ‘{}’ text presented – **notice that it is wrapped in curly brackets this time!**

```
{"privatebuyer":"DigiBank","privateprice":"9999","offerdate":"23/09/2019"}
```

```
{"privatebuyer":"DigiBank","privateprice":"9999","offerdate":"23/09/2019"}
```

```
optional: What is the transient data for the transaction, e.g. {"key": "value"} (Press 'Enter' to confirm or 'Escape' to cancel)
```

- **76.** Check the output pane at the bottom for a SUCCESS message confirming that the offer data was submitted successfully.

**Note:** We are logging the private information to the console for debug and demo purposes – you would not do this in a live application!

```
[10/23/2019 6:39:17 PM] [INFO] submitting transaction offer with args 001 on channel tradechannel
[10/23/2019 6:39:20 PM] [SUCCESS] Returned value from offer: {"privatebuyer":"DigiBank","privateprice":9999}
```

To verify what was written to the private data collection, we have a helper function called **readOffer** which we can call, and verify the collection returns us the data that was written earlier.

- **77.** Right-click on transaction **readOffer** and click **Evaluate Transaction**

- **78.** When prompted for parameters, enter the following:

["001"]

```
["001"]
optional: What are the arguments to the transaction, (e.g. ["arg1", "arg2"]) (Press 'Enter' to confirm or 'Escape' to cancel)
```

- **79.** There is no transient data for this function, just press **enter** leaving the empty curly braces {}

The transaction will now be submitted, and should return with a SUCCESS message

```
[10/31/2019 9:42:46 AM] [INFO] evaluateTransaction
[10/31/2019 9:43:04 AM] [INFO] evaluating transaction readOffer with args 0@1 on channel tradechannel
[10/31/2019 9:43:04 AM] [SUCCESS] Returned value from readOffer: {"privatebuyer":"DigiBank", "privateprice":9999}
```

Notice that David can view this confidential information.

We can also review the ‘private data footprints’ in CouchDB – earlier, we launched 3 Firefox browser tabbed sessions, which will list currently databases and can click through to the individual records we’ve created at this time.

- **80.** Switch to the **firefox** window

- **81.** Click on the DigiBank CouchDB tab in Firefox – this is the URL running on **port 6984** and is the **first tab**. Refresh the page to see the update3d information.

- **82.** Review the databases created:

**tradechannel\_exchangecontract** the world state database for the exchangecontract on the tradechannel

**tradechannel\_exchangecontract\$\$h\$collection\$d** the private collection hashstore for CollectionD

**tradechannel\_exchangecontract\$\$p\$collection\$d** the private collection for Collection D (which is only available in Digibank’s CouchDB)

The screenshot shows a Mozilla Firefox browser window titled "Project Fauxton - Mozilla Firefox". The address bar displays "localhost:6984/\_utils/". The main content area is titled "Databases". On the left is a sidebar with various icons: a double arrow (back/forward), a wrench (replicator), a grid (users), a gear (tradechannel\_), a double square (exchangecontract), a book (collection), a checkmark (exchangecontract\$), and a person (lscc). The "tradechannel\_" icon is highlighted with a red box. The main table lists databases with their names and sizes:

Name	Size
_replicator	3.8 KB
_users	3.8 KB
tradechannel_	21.7 KB
tradechannel_exchangecontract	4.7 KB
tradechannel_exchangecontract\$\$h\$c ollection\$d	2.9 KB
tradechannel_exchangecontract\$\$p\$c ollection\$d	2.1 KB
tradechannel_lscc	3.0 KB

The last three rows of the table, which represent the private collection "tradechannel\_exchangecontract\$\$p\$collection\$d", are also highlighted with a red box.

-- 83. Click on the private collection tradechannel\_exchangecontract\$\$p\$collection\$d

-- 84. Click on the **{ } JSON** view and you can see a record of the private data that was written for trade “001”.

When you have reviewed the data, click the **All DBs** icon to return.

```

id "org.example.marketplace.Trade001"
{
  "id": "\u0000org.example.marketplace.Trade\u0000001\u0000",
  "key": "\u0000org.example.marketplace.Trade\u0000001\u0000",
  "value": {
    "rev": "1-dc86ad0724ec7b801fc2a5ba1553629f"
  },
  "doc": {
    "_id": "\u0000org.example.marketplace.Trade\u0000001\u0000",
    "_rev": "1-dc86ad0724ec7b801fc2a5ba1553629f",
    "privatebuyer": "DigiBank",
    "privateprice": 9999,
    "-version": "\u0000CgMBAA="
  }
}

```

-- 85. Back in the main database list – click on the world state database **tradechannel\_exchangecontract**

-- 86. Click on the **{ } JSON** view and you can see the record for “contractID”: “001”

Notice that the offer date is shown, but the private data is not.

Again, when you have reviewed the data, click the **All DBs** icon to return.

```

id "org.example.marketplace.Trade001"
{
  "id": "\u0000org.example.marketplace.Trade\u0000001\u0000",
  "key": "\u0000org.example.marketplace.Trade\u0000001\u0000",
  "value": {
    "rev": "5-990f02ecdcd74e96ec3314b5fdd7547b"
  },
  "doc": {
    "_id": "\u0000org.example.marketplace.Trade\u0000001\u0000",
    "_rev": "5-990f02ecdcd74e96ec3314b5fdd7547b",
    "accept": {
      "acceptdate": "24/09/2019",
      "offer": {
        "contractID": "001",
        "marketdate": "21/09/2019",
        "offerdate": "23/09/2019"
      }
    }
  }
}

```

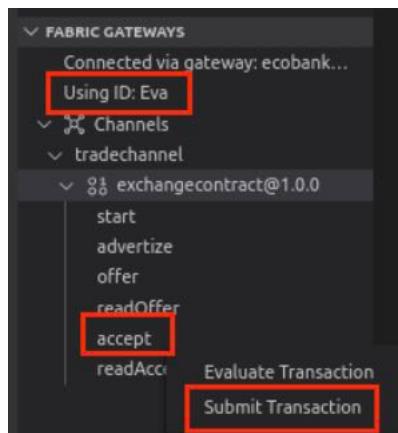
- **87.** Still using the **first tab** for the **Digibank** CouchDB open the collection hash database **tradechannel\_exchangecontract\$\$h\$collection\$d** and verify that the key and value is unreadable.
- **88.** Using the **second tab** for the **Ecobank** CouchDB, look at the database list and notice that Ecobank does not have the private database **(tradechannel\_exchangecontract\$\$p\$collection\$d)**

This is expected because the data is confidential to Digibank.

Having verified the databases in CouchDB we can proceed with the transaction flow with Ecobank (Eva) accepting the offer from Digibank.

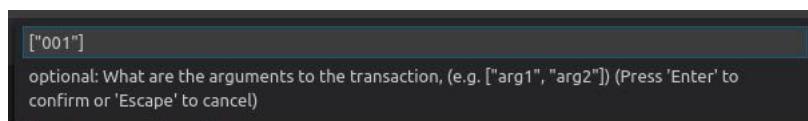
- **89.** Return to the VS Code editor and in the Fabric Gateways panel, click the Disconnect icon to disconnect from DigiBank's gateway.
- **90.** In the Fabric Gateways panel connect to **ecobank\_gw** and choose **Eva** as the identity
- **91.** Expand the transaction list as before, and **right click** the **accept** transaction and click **Submit Transaction**.

We will now perform an accept transaction, as Eva. This will create some private data, for an approved accept transaction in EcoBank's private data collection.



When prompted to enter arguments enter the following:

```
[ "001" ]
```



- \_\_ **92.** Enter or paste the following transient data for this transaction, replacing the '{}' provided with the following one-liner. Remember that you can copy and paste from the text file in gedit if necessary:

```
{"acceptprice":"9999","privatebuyer":"DigiBank","privateprice":"9999","acceptdate":"24/09/2019"}
```

```
{"acceptprice":"9999","privatebuyer":"DigiBank","privateprice":"9999","acceptdate":"24/09/2019"}
```

optional: What is the transient data for the transaction, e.g. {"key": "value"} (Press 'Enter' to confirm or 'Escape' to cancel)

The transaction is now submitted.

- \_\_ **93.** Review the output pane at the bottom for results – you should see a SUCCESS message similar to this:

```
[10/23/2019 7:16:33 PM] [INFO] submitTransaction
[10/23/2019 7:20:13 PM] [INFO] submitting transaction accept with args 001 on channel tradechannel
[10/23/2019 7:20:15 PM] [SUCCESS] Returned value from accept: {"acceptprice":9999,"privatebuyer":"DigiBank","privateprice":9999}
(Confirm or escape to cancel) banking.js ① package.json
```

Notice that the 'acceptdate' value is not part of this SUCCESS message. This is because we've made the 'acceptdate' a 'public' field, so that gets written to the channel ledger. The data set shown above (first 3 JSON values) is private data only viewable by EcoBank.

- \_\_ **94.** Return to Firefox and click on the **second tab** for the **Ecobank CouchDB** running on **Port 5984**

- \_\_ **95.** Click the **All DBs** icon and open the database for the private collection **CollectionE tradechannel\_exchangecontract\$\$p\$collection\$e**

- \_\_ **96.** Click on the **{ } JSON** format icon

Notice that the record for Trade001 has the private data for the accept transaction.

We are now going to try and read Ecobank private data with a Digibank ID which should fail, proving that the data is confidential to Ecobank.

- \_\_ **97.** Return to VS Code and in the Fabric Gateways panel click the **Disconnect** icon to disconnect from the current gateway.

- \_\_ **98.** Connect to **digibank\_gw** with the identity **David**.

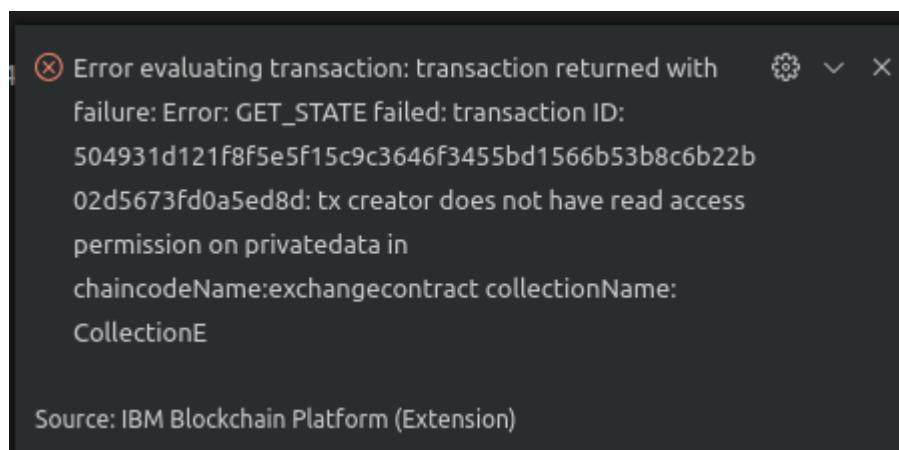
\_\_ **99.** Right-click on the **readAccept** transaction and click **Evaluate Transaction**. When prompted enter the following in the parameters

```
[ "001"]
```

\_\_ **100.** There is no transient data for this function, just press **enter** leaving the empty curly braces {}

The evaluate should return an error.

\_\_ **101.** Expand the popup message on the bottom right in VS Code and you'll see a message that the transaction failed. This is because the Digibank Identity used to invoke the transaction cannot access the private collection.



\_\_ **102.** Now **disconnect** from the DigiBank Gateway in the VS Code extension.

## Review

- In this part of the Lab you have understood how to add private data transaction functions to a smart contract.
- You've instantiated the smart contract with a Collection Policy describing two private data collections - one each for EcoBank and DigiBank,
- You have used the new functions to create private data in those collections.
- You have seen that the transactions executed simultaneously create a private data hash store, containing the public hash, corresponding to the original private data stored off-chain.
- You have seen that Private Data is indeed Private – David from Digibank could not see the Ecobank collection.

## 4 Work with the Cross Verify Functions

In this section we will upgrade our smart contract to include a crossVerify transaction implementation, which will allow the smart contract to verify a calculated hash of source data against the private data hash record.

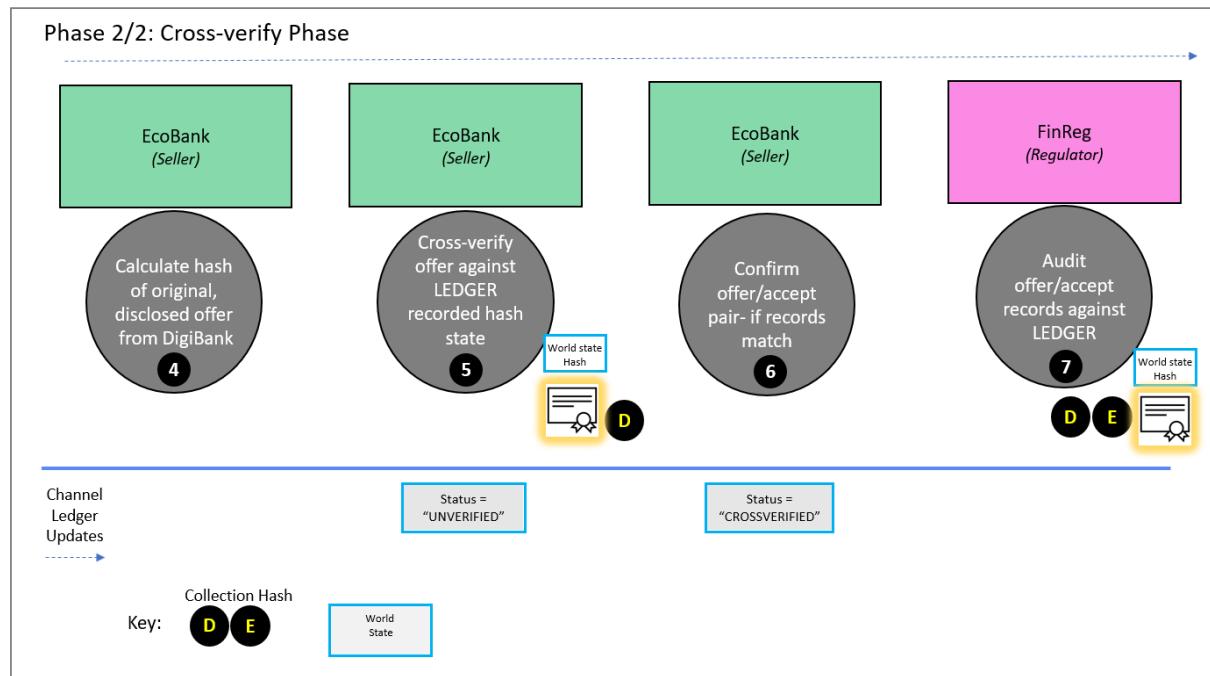
An identity from a proving organisation (for example, a Regulator) will invoke this cross-verify function and return a MATCH / NO MATCH response.

It's important to note that the verifying party only has access to the data hash store and not the private collection containing the confidential information.

Here are the steps to the cross-verify process:

1. **EcoBank** calculates a hash of the accepted offer and cross-verifies with the written hash of the offer for the contract in question, and which was written to the 'public' channel ledger at the same time the original private data transaction took place.
2. **EcoBank** cross-verifies they match.
3. If there is a match, the offer/accept pair is verified and the channel ledger record for the contract, is now set to 'CROSSVERIFIED' - At this point, the exchange can be completed, and the resultant sale transfer can be initiated
4. Later, the regulator **FinReg** wishes to cross-check the original offer and accept transactions recorded to each organisation's private data store, matches what is stored in the channel ledger.

## Verify Pattern



Note that as we upgrade the smart contract to include the crossVerify method, we will also implement a query transaction type we will use in the subsequent section of the lab.

## 4.1 Understanding the Cross-Verify implementation

-- 103. Click the Explorer icon in the VS Code editor, and return to the **trading-contract.js** file.

-- 104. Scroll down to **line 241**, where you find the code comment

```
// UTILITY functions that may prove useful at some point
```

```
233     console.log('No private data with that Key: ', readKey);
234     return 'No private data with that Key: ' + readKey;
235   }
236   return pdString;
237 }
238
239
240
241 // UTILITY functions that may prove useful at some point
242
243 /**
244  * Get invoking MSP id - or any other CID related values
245  * @param {Context} ctx the transaction context
246  */
247 async _getInvokingMSP(ctx) {
```

This is where you will paste in the remaining functions of the contract.

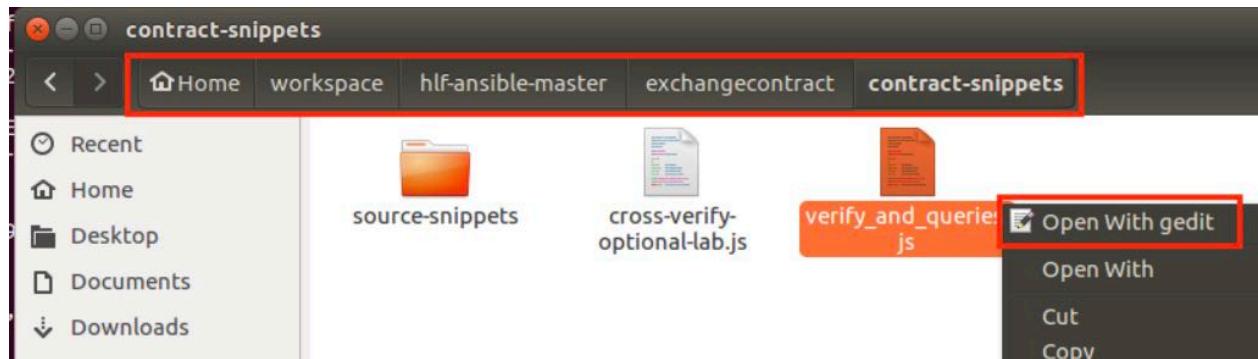
-- 105. Open the Ubuntu File Explorer



-- **106.** Navigate to the following folder:

Home > workspace > hlf-ansible-master > exchangecontract > contract-snippets

**Right click** on `verify_and_queries.js` and select **Open with Gedit**



-- **107.** In the gedit session, highlight all the code by pressing **CTRL+A** to select, then press **CTRL+C** to copy to the clipboard

-- **108.** Back in the VS Code Explorer session, position the cursor at **line 239** and paste in the contents you copied to the clipboard using **CTRL+V**. Scroll back to Line 239 to check it was pasted OK.

```

236     return pdString;
237 }
238
239
240 /**
241 * FABRIC-BASED Cross verify function - as non-member of Collection, can verify the hash, against what the seller's bank s
242 * @param {Context} ctx the transaction context
243 * @param {String} collection the collection name - this function can be called by a regulator, calling different collect
244 * @param {String} contractID contractID
245 * @param {String} hashvalue the SHA256 hash string calculated from the source private data to compare against the hash s
246 */
247 async crossVerify(ctx, collection, contractID, hashvalue) {
248
249     console.log('retrieving the hash from the PDC hash store of the buy transaction (fn: crossVerify)' + contractID);
250
251     let readKey = ctx.stub.createCompositeKey(AssetSpace, [contractID]);
252     let pdHashBytes = await ctx.stub.getPrivateDataHash(collection, readKey);
253     //console.log('~~ PDHASH raw is ', pdHashBytes.toString('hex'));
254
255     if (pdHashBytes.length > 0) {

```

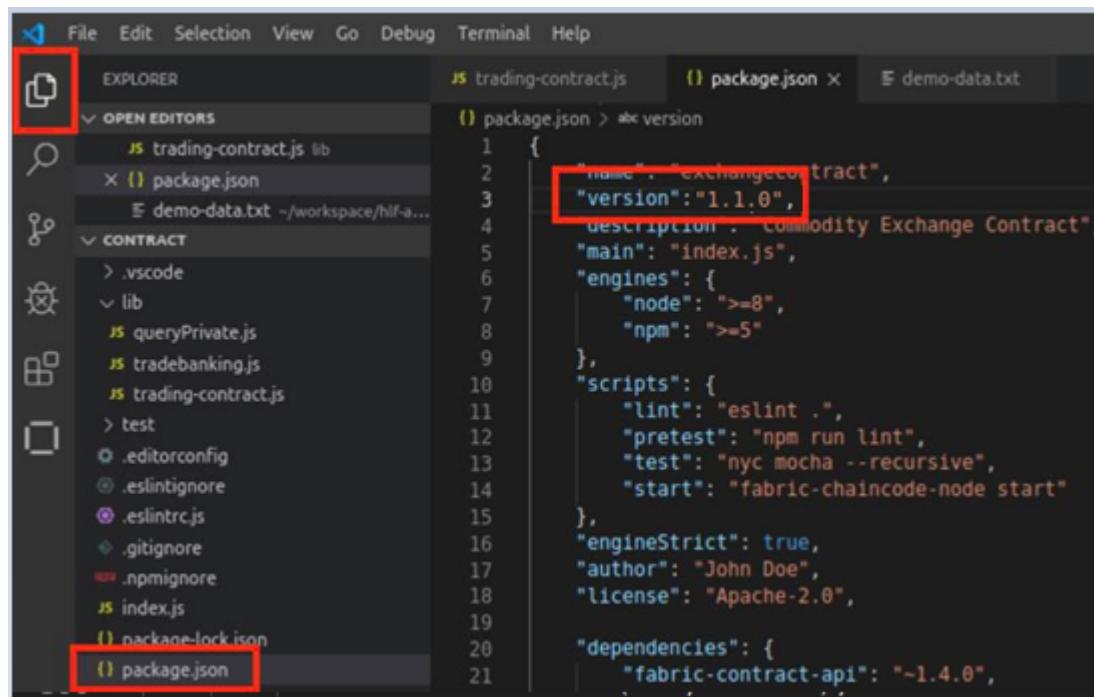
-- **109.** Select all the code in the window with **CTRL+A**.

Right-click and select **Format Selection** to format the code to fix the indentation.

The cross-verify code functionality and logic has now been added to the **trading-contract.js** edit session.

-- **110.** Save the file by pressing **CTRL+S**.

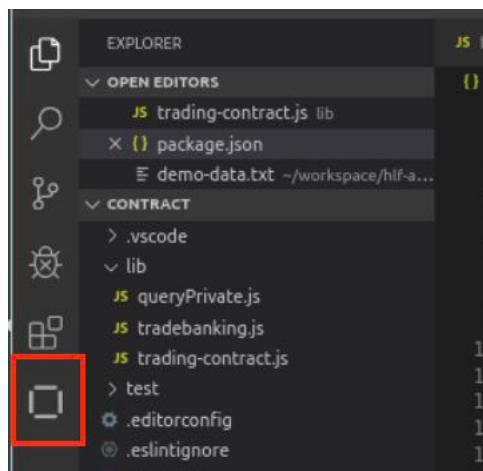
-- **111.** Still in the Explorer view, click on the **package.json** file and change the version number to “1.1.0”. Press **CTRL+S** to save the file.



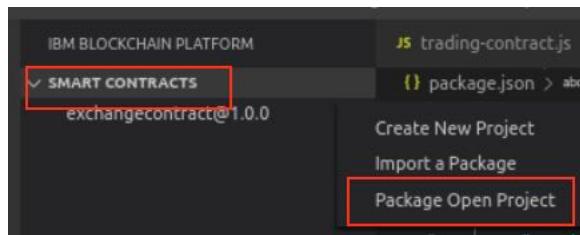
```
1  {
2    "name": "exchangecontract",
3    "version": "1.1.0",
4    "description": "commodity Exchange Contract",
5    "main": "index.js",
6    "engines": {
7      "node": ">=8",
8      "npm": ">=5"
9    },
10   "scripts": {
11     "lint": "eslint .",
12     "pretest": "npm run lint",
13     "test": "nyc mocha --recursive",
14     "start": "fabric-chaincode-node start"
15   },
16   "engineStrict": true,
17   "author": "John Doe",
18   "license": "Apache-2.0",
19
20   "dependencies": {
21     "fabric-contract-api": "~1.4.0",
22   }
23 }
```

This will allow us to create a smart contract package with a new version number, and therefore allow us to upgrade our instantiated contract on the channel.

- **112.** Click on the IBM Blockchain Platform extension icon.



- **113.** Click on the ellipsis to the right of the Smart Contracts panel and select **Package Open Project**.



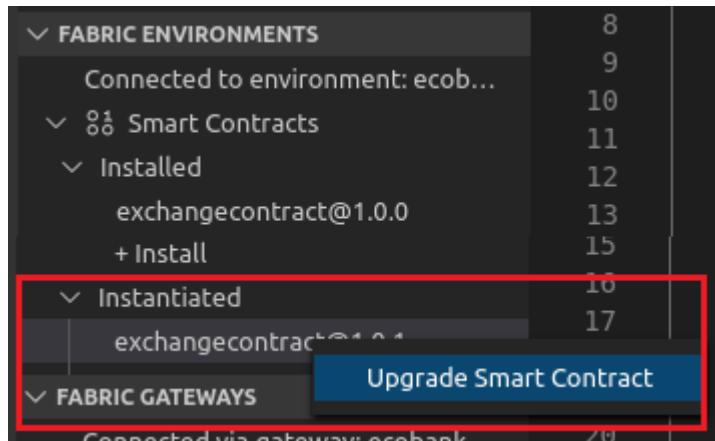
\_\_ 114. In the Fabric Environments view, click the Disconnect icon to disconnect from any connected environment, then connect to the **digibank** environment.

Click **+Install** and install the **exchangecontract@1.1.0** smart contract. Once completed, you'll get a confirmation it was performed successfully.

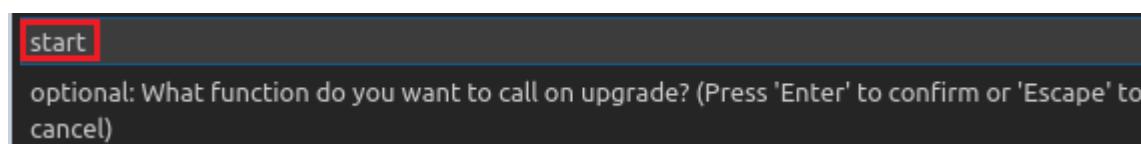
\_\_ 115. Repeat step 114 and install the new contract version onto the **finreg** environment.

\_\_ 116. Repeat step 114 and install the new contract version onto the **ecobank** environment.

\_\_ 117. Still connected to EcoBank's Fabric Environment **right click** the instantiated **exchangecontract@1.0.0** contract and click **Upgrade Smart Contract**. Select **exchangecontract@1.1.0** as the smart contract to upgrade to.

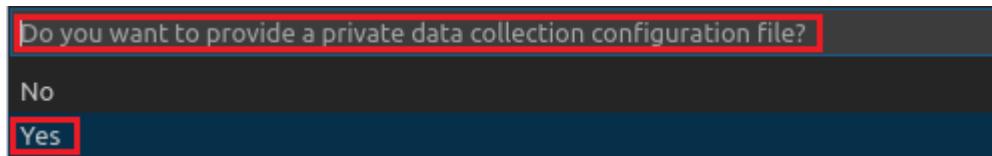


\_\_ 118. Enter **start** as the function to call on upgrade.

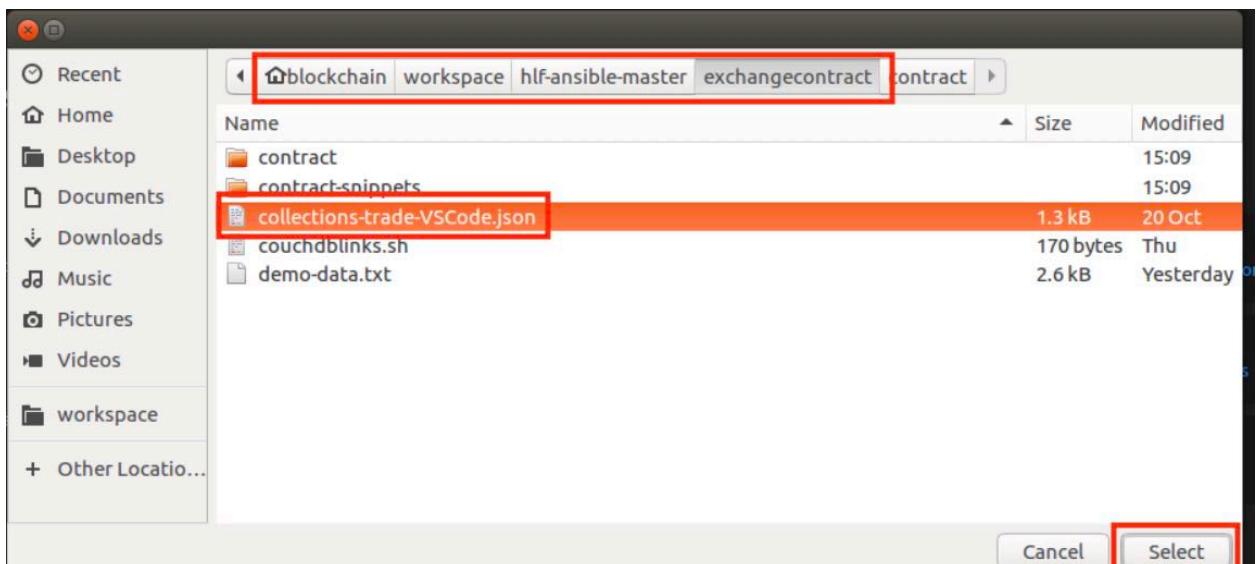


\_\_ 119. Press Enter to accept the default arguments to the function as there are none.

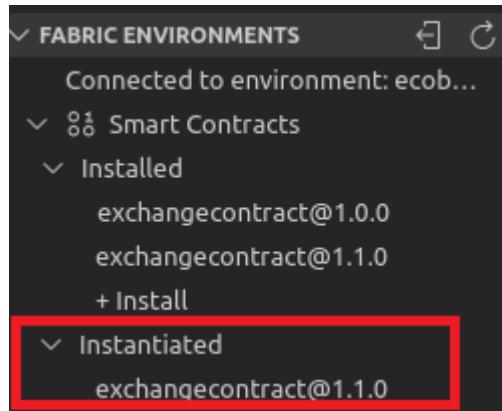
- **120.** When asked if you ‘want to provide a private data collection configuration file’ – select ‘Yes’



- **121.** Click **Browse** and navigate to the folder  
Home > workspace > hlf-ansible-master > exchangecontract  
and select the file **collections-trade-VSCode.json**.



Upgrading may take a minute or so. When the process has completed it will show in the Fabric Environments view as follows:



The smart contract has been instantiated on the tradechannel channel.

Before we proceed to invoke the cross-verification transactions, it helps to have a bit more explanation of what transaction functions were added by the copy/paste actions, and subsequent upgrade of the smart contract.

**122.** In the trading-contract.js editor scroll to line 247.

Let's look at the first transaction – **crossVerify()**.

```
JS trading-contract.js X
lib > JS trading-contract.js > TradingContract
246
247  */
248
249
250
251
252  let readKey = ctx.stub.createCompositeKey(AssetSpace, [contractID]);
253  let pdHashBytes = await ctx.stub.getPrivateDataHash(collection, readKey);
254  //console.log('~~ PDHASH raw is ', pdHashBytes.toString('hex'));
255
256  if (pdHashBytes.length > 0) {
257    // gets back the hash from the hash store
258    console.log('retrieved private data hash from collection');
259  }
260  else {
261    console.log('No private data hash with that Key: ', readKey);
262    return 'No private data hash with that Key: ' + readKey;
263  }
264
265  // retrieve SHA256 hash of the converted Byte array -> string from private data collection's hash store (DB)
266  let actual_hash = pdHashBytes.toString('hex');
267
268  //update the main ledger with status
269  // Get the 'channel hash' written in the 'offer' function (CUSTOM) - from the world state
270  let acceptKey = ctx.stub.createCompositeKey(AssetSpace, [contractID]);
271  let acceptBytes = await ctx.stub.getState(acceptKey);
272  if (acceptBytes.length > 0) {
273    var verify = JSON.parse(acceptBytes);
274    console.log('retrieved contract (crossVerify)');
275    console.log(verify);
276  }
277  else {
278    console.log('Nothing advertised with that Key: ', contractID);
279    var verify = "EMPTY CONTRACT (crossVerify function)";
280    return verify;
281  }
282  if (hashvalue === actual_hash) {
283    verify.accept.offer.status = "CONFIRMED";
284    verify.accept.status = "CROSSVERIFIED";
285    let accept = verify.accept;
```

**Lines 251-252** takes the supplied parameter (Contract ID) and finds the hashed key for this ID in the hashed data store using **getPrivateDataHash**

**Line 264-265** converts the encoded Byte Array to a hexadecimal SHA256 hash

**Line 281** compares the calculated hash (you do this in the ‘perform’ lab below) that’s supplied as a parameter (hashvalue) – to the converted hash from line **265**

Depending on the comparison, it will either MATCH or FAIL MATCH. If a match is made lines **282-285** show that the status on the main channel ledger is updated to ‘CROSSVERIFIED’

**Line 301** is the **regulatorVerify** transaction – this is a ‘specialist’ transaction for use by a regulator.

```
301
302     async regulatorVerify(ctx, collection, contractID, hashvalue) {
303
304         let invokingMSPid = await this._getInvokingMSP(ctx);
305         console.log('regulatorVerify function called by..' + invokingMSPid); // written to the container
306
307         // if (invokingMSPid !== Regulator) return 'unauthorized to call this function as MSP: ' + invokingMSPid;
308
309         if (invokingMSPid !== Regulator) throw new Error('unauthorized to call this function as MSP: ' + invokingMSPid);
310
311         console.log('retrieving the hash from the PDC hash store of the buy transaction (fn: regulatorVerify)' + contractID);
312
313         let acceptKey = ctx.stub.createCompositeKey(Assetspace, [contractID]);
314         let pdHashBytes = await ctx.stub.getPrivateDataHash(collection, acceptKey);
315
316         if (pdHashBytes.length > 0) {
317             // gets back the hash from the hash store
318             console.log('retrieved private data hash from collection');
319         }
320         else {
321             console.log('No private data hash with that Key: ', acceptKey);
322             return 'No private data hash with that Key: ' + acceptKey;
323         }
324
325         // retrieve SHA256 hash of the converted Byte array -> string from private data collection's hash store (DB)
326         let actual_hash = pdHashBytes.toString('hex');
327
328         if (hashvalue === actual_hash)
329             return '\nCalculated Hash provided: \n' + hashvalue + '\n\n' MATCHES <-----> \n\nHash from Private Data Hash State \n';
330         else
331             return 'Could not match the Private Data Hash State: ' + actual_hash;
332
333     }
334
335     else {
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1116
1117
1118
1118
1119
1120
1121
1122
1123
1124
1125
1126
1126
1127
1128
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1165
1166
1167
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1175
1176
1177
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1196
1196
1197
1198
1198
1199
1200
1201
1202
1203
1204
1205
1205
1206
1207
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1215
1216
1217
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1225
1226
1227
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1237
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1247
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1255
1256
1257
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1265
1266
1267
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1277
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1294
1295
1296
1296
1297
1298
1298
1299
1300
1301
1302
1303
1304
1304
1305
1306
1306
1307
1308
1308
1309
1310
1311
1312
1313
1314
1314
1315
1316
1316
1317
1318
1318
1319
1320
1321
1322
1323
1323
1324
1325
1325
1326
1327
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1335
1336
1337
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1345
1346
1347
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1355
1356
1357
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1365
1366
1367
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1375
1376
1377
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1385
1386
1387
1387
1388
1389
1389
1390
1391
1392
1393
1394
1394
1395
1396
1396
1397
1398
1398
1399
1400
1401
1402
1403
1404
1404
1405
1406
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1414
1415
1416
1416
1417
1418
1418
1419
1420
1420
1421
1422
1423
1424
1425
1425
1426
1427
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1435
1436
1437
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1445
1446
1447
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1455
1456
1457
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1465
1466
1467
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1475
1476
1477
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1485
1486
1487
1487
1488
1489
1489
1490
1491
1492
1493
1494
1494
1495
1496
1496
1497
1498
1498
1499
1500
1501
1502
1503
1504
1504
1505
1506
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1514
1515
1516
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1524
1525
1526
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1534
1535
1536
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1544
1545
1546
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1554
1555
1556
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1564
1565
1566
1566
1567
1568
1568
1569
1570
1570
1571
1572
1573
1574
1575
1575
1576
1577
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1585
1586
1587
1587
1588
1589
1589
1590
1591
1592
1593
1594
1594
1595
1596
1596
1597
1598
1598
1599
1600
1601
1602
1603
1604
1604
1605
1606
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1614
1615
1616
1616
1617
1618
1618
1619
1620
1620
1621
1622
1623
1624
1625
1625
1626
1627
1627
1628
1629
1629
1630
1631
1632
1633
1634
1634
1635
1636
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1644
1645
1646
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1654
1655
1656
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1664
1665
1666
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1674
1675
1676
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1684
1685
1686
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1694
1695
1696
1696
1697
1698
1698
1699
1700
1701
1702
1703
1704
1704
1705
1706
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1714
1715
1716
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1724
1725
1726
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1734
1735
1736
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1744
1745
1746
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1754
1755
1756
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1764
1765
1766
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1774
1775
1776
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1784
1785
1786
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1794
1795
1796
1796
1797
1798
1798
1799
1800
1801
1802
1803
1804
1804
1805
1806
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1814
1815
1816
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1824
1825
1826
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1834
1835
1836
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1844
1845
1846
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1854
1855
1856
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1864
1865
1866
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1874
1875
1876
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1884
1885
1886
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1894
1895
1896
1896
1897
1898
1898
1899
1900
1901
1902
1903
1904
1904
1905
1906
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1914
1915
1916
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1924
1925
1926
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1934
1935
1936
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1944
1945
1946
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1954
1955
1956
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1964
1965
1966
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1974
1975
1976
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1984
1985
1986
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1994
1995
1996
1996
1997
1998
1998
1999
2000
2001
2002
2003
2004
2004
2005
2006
2006
2007
2008
2008
2009
2010
2011
2012
2013
2014
2014
2015
2016
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2024
2025
2026
2026
2027
2028
2028
2029
2030
2031
2032
2033
2034
2034
2035
2036
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2044
2045
2046
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2054
2055
2056
2056
2057
2058
2058
2059
2060
2061
2062
2063
2064
2064
2065
2066
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2074
2075
2076
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2084
2085
2086
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2094
2095
2096
2096
2097
2098
2098
2099
2100
2101
2102
2103
2104
2104
2105
2106
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2114
2115
2116
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2124
2125
2126
2126
2127
2128
2128
2129
2130
2131
2132
2133
2134
2134
2135
2136
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2144
2145
2146
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2154
2155
2156
2156
2157
2158
2158
2159
2160
2161
2162
2163
2164
2164
2165
2166
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2174
2175
2176
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2184
2185
2186
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2194
2195
2196
2196
2197
2198
2198
2199
2200
2201
2202
2203
2204
2204
2205
2206
2206
2207
2208
2208
2209
2210
2211
2212
2213
2214
2214
2215
2216
2216
2217
2218
2218
2219
2220
2221
2222
2223
2224
2224
2225
2226
2226
2227
2228
2228
2229
2230
2231
2232
2233
2234
2234
2235
2236
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2244
2245
2246
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2254
2255
2256
2256
2257
2258
2258
2259
2260
2261
2262
2263
2264
2264
2265
2266
2266
2267
2268
2268
2269
2270
2271
2272
2273
2274
2274
2275
2276
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2284
2285
2286
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2294
2295
2296
2296
2297
2298
2298
2299
2300
2301
2302
2303
2304
2304
2305
2306
2306
2307
2308
2308
2309
2310
2311
2312
2313
2314
2314
2315
2316
2316
2317
2318
2318
2319
2320
2321
2322
2323
2324
2324
2325
2326
2326
2327
2328
2328
2329
2330
2331
2332
2333
2334
2334
2335
2336
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2344
2345
2346
2346
2347
2348
2348
2349
2350
2351
2352
2353
2354
2354
2355
2356
2356
2357
2358
2358
2359
2360
2361
2362
2363
2364
2364
2365
2366
2366
2367
2368
2368
2369
2370
2371
2372
2373
2374
2374
2375
2376
2376
2377
2378
2378
2379
2380
2381
2382
2383
2384
2384
2385
2386
2386
2387
2388
2388
2389
2390
2391
2392
2393
239
```

Once again, on line 301, we see the parameters are the collection name, the contract ID and a calculated hashvalue to compare against the ledger state as written by **PutPrivateData()** in the offer/accept phase.

**Lines 307-308** show that this transaction can ONLY be called by someone that is in the Regulator organisation (there is a transaction that checks the calling identity's MSP id), so identities from other organisations will not be able to execute this specialist transaction

**Line 312-313** once again show the use of the `getPrivateDataHash` transaction

**Lines 326-328** show what actions are taken dependent on whether there is a MATCH or failed MATCH between the hashes.

Further transactions added are also helper transactions like **ShowTransient** (show inputs for a given Transient set), **invokeBadCollection** (what you see when an attempt to write private data to an invalid or non-existent collection) and an **initLedger** transaction that is called once as **David** (Digibank) and **Eva** (EcoBank) to create some sample/demo private data in their respective collections (data is used for querying later).

#### 4.2 Performing the Cross Verification of Private Data

The next step is to perform the verification: EcoBank had accepted the offer details provided by DigiBank, but now want to verify it against hash record of what DigiBank had originally created in their own collection.

- **123.** In VS Code, click the **terminal** pane at the bottom of the screen. (If you cannot see this, click the VS Code menu “View->Terminal”.)

When the prompt appears, paste in the following line exactly (including the escape characters)

```
echo -n "{\"privatebuyer\":\"DigiBank\",\"privateprice\":9999}" |shasum -a 256
```



A screenshot of the VS Code interface showing the terminal tab selected. The terminal window displays a command being run: "echo -n "{\"privatebuyer\":\"DigiBank\",\"privateprice\":9999}" |shasum -a 256". The output of the command, which is a long SHA-256 hash starting with "2261634aed735c9b5ad38810571379a7f1502628b9dc231bc59b725f369bb669", is also visible in the terminal window.

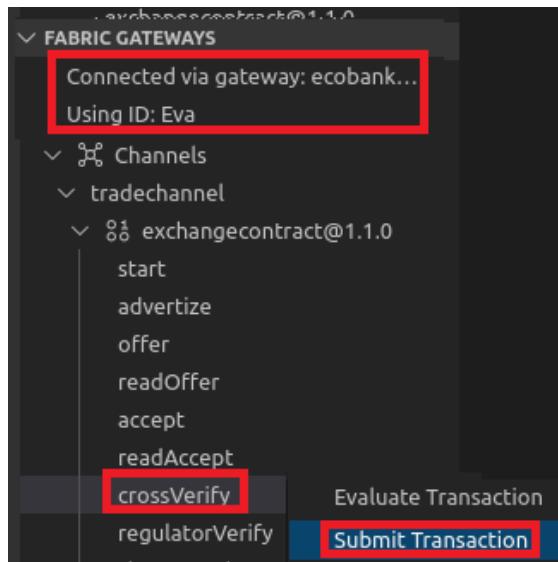
This will calculate a SHA 256 hash value of the source data EcoBank had agreed with DigiBank.

The command above returns a hash value beginning 22616. For convenience, we will provide this verification hash in the parameter string which you will shortly pass into the new smart contract transaction

- **124.** In the **Fabric Gateways** panel disconnect from click **ecobank\_gw** and connect with identity **Eva**

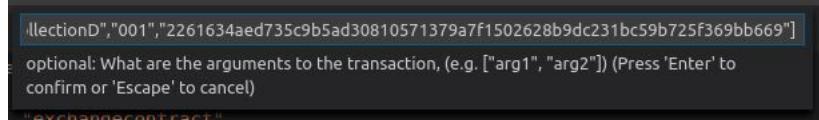
- **125.** Expand the **Channels/exchangecontract@1.1.0**. Right-click **crossVerify** and click **Submit Transaction**.

Note that we are running ‘Submit Transaction’ and not ‘Evaluate Transaction’ because we will be updating the status on the channel ledger for the trade contract “001” to be ‘CROSSVERIFIED’ if the verify pattern returns a TRUE result or ‘MATCH’.



- **126.** When prompted, paste in the following parameter list as a one-liner in the VS Code parameter field – replacing the existing ‘[]’ text (again this is available in the text file in gedit)

```
[ "CollectionD", "001", "2261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669" ]
```



- **127.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces {}

- **128.** Wait for the transaction to complete, then select the Output pane and review the messages.

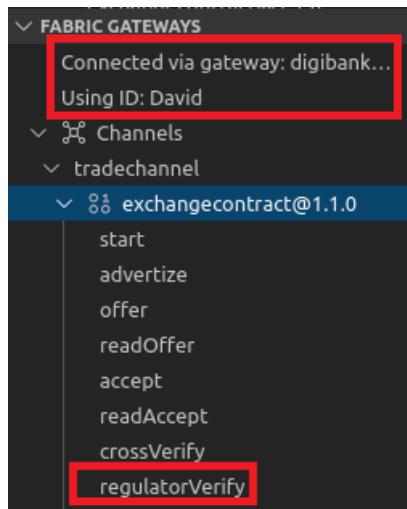
It should provide a match of the source data hash (for which we calculated a hash from the terminal) against the private data hash store record for Contract “001” – a **‘MATCHES’** banner should appear, and a confirmation of the hash values that were compared:

```
[10/25/2019 4:38:31 PM] [SUCCESS] Connecting to ecobank_gw
[10/25/2019 4:43:19 PM] [INFO] submitTransaction
[10/25/2019 4:43:52 PM] [INFO] submitTransaction
[10/25/2019 4:45:40 PM] [INFO] submitting transaction crossVerify with args CollectionD,001,
2261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669 on channel tradechannel
[10/25/2019 4:45:42 PM] [SUCCESS] Returned value from crossVerify:
Calculated Hash provided:
2261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669
[ ] | MATCHES <----->
Hash from Private Data Hash State
2261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669
```

Note that equally, DigiBank, can take on the role of verifier (in the scenario where it needs to be verified by the other party): DigiBank can verify that the EcoBank hash store ('CollectionE') proves that the accept price stored by EcoBank privately, matches what was agreed by DigiBank, when the transaction was committed to the blockchain.

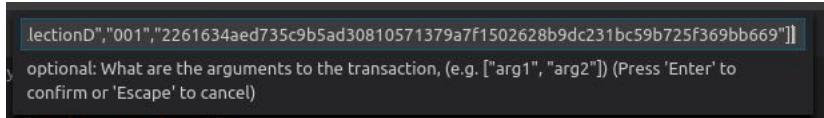
The next step is to cross-verify data as a Regulator auditing the data. The transaction **regulatorVerify** was added as part of the code you pasted into the smart contract earlier. This transaction can only be called by someone from the regulator organisation.

- **129.** Still connected as the identity 'David' from DigiBank, highlight the **regulatorVerify** transaction, **right-click** on **regulatorVerify** and click **Evaluate Transaction**

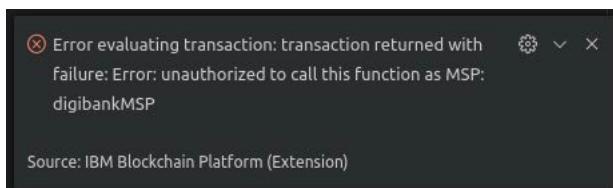


- **130.** Provide the following one-liner parameter list as arguments when prompted, replacing the existing '[]' text:

```
[ "CollectionD", "001", "2261634aed735c9b5ad30810571379a7f1502628b9dc231bc5  
9b725f369bb669" ]
```

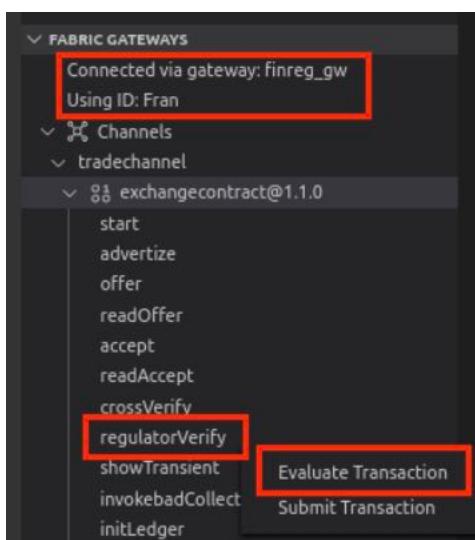


- **131.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces {}
- **132.** You should get an error, with a message explaining that you are not authorized to call this transaction because you're not an identity issued by the regulator organisation. (The **regulatorVerify** transaction checks the MSP id of the invoking identity.)



We will now switch to FinReg and invoke the transaction with an authorized identity.

- **133.** On the **Fabric Gateways** panel disconnect from DigiBank's gateway and click **finreg\_gw** and connect with identity **Fran**.
- **134.** Once again, right-click the **regulatorVerify** transaction from the transaction list and click **Evaluate Transaction**



- **135.** Provide the same parameter list as before when prompted:

```
[ "CollectionD", "001", "2261634aed735c9b5ad30810571379a7f1502628b9dc231bc5  
9b725f369bb669" ]
```

- 136. Again, there is no transient data for this transaction so just press **enter** leaving the empty curly braces {}

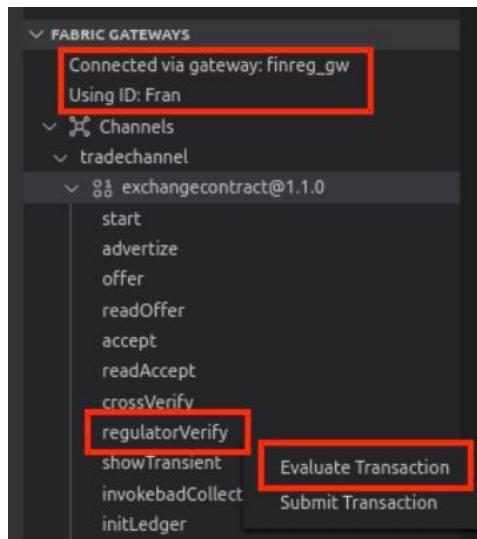
Now you should see that the evaluation is successful this time and the hashes match. You used evaluate for this transaction because there is no status update to the ledger by the transaction and the regulator only uses read transactions.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
[10/24/2019 11:56:02 AM] [INFO] evaluating transaction regulatorVerify with args CollectionD,001,  
tradechannel
[10/24/2019 11:56:02 AM] [SUCCESS] Returned value from regulatorVerify:  
Calculated Hash provided:  
2261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669
| | | MATCHES <----->
Hash from Private Data Hash State
2261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669
```

Up to now, all the cross-verify and regulator verify transactions have shown a MATCH so now we will deliberately mis-MATCH a regulator verify

- 137. Once again, **right-click** the **regulatorVerify** transaction from the transaction list and click **Evaluate Transaction**



- 138. Provide the same parameter list as before except we change the first digit of the hash to be a **9** instead of a **2**:

```
[ "CollectionD", "001", "9261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669" ]
```

-- **139.** Again, there is no transient data for this transaction so just press **enter** leaving the empty curly braces {}

Now you should see that there is no match.

```
[11/1/2019 3:13:49 PM] [INFO] connecting to FinReg_gw
[11/1/2019 3:13:49 PM] [INFO] evaluateTransaction
[11/1/2019 3:14:03 PM] [INFO] evaluating transaction regulatorVerify with args CollectionD,001,
9261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669 on channel_tradechannel
[11/1/2019 3:14:03 PM] [SUCCESS] Returned value from regulatorVerify: Could not match the Private Data Hash
State: 2261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669
|
```

This illustrates that the transaction works as expected, reporting when there is no match. This situation might arise if the regulator has been provided with the wrong pricing information to hash up, and the regulator would have to initiate a detailed investigation with the trading organisations.

## Review

In this part of the lab

- You have modified the smart contract to add new transactions, installed on peers for all 3 organisations and upgraded the running version to v1.1.0
- You have worked with a **crossVerify** transaction that both **EcoBank** (as the seller/acceptor) and **DigiBank** (as the buyer) can execute against collection hash stores to verify data shared for verification purposes.
- You have worked with a **regulatorVerify** transaction, which enables Fran (an identity associated with the regulator **FinReg**) to cross-verify or ‘audit’ that the private data evidence matches against the hashed records in the private data hash stores.
- You have tested the **regulatorVerify** with a hash that does not match.

## 5 Add Demo Data and Private Query Transaction

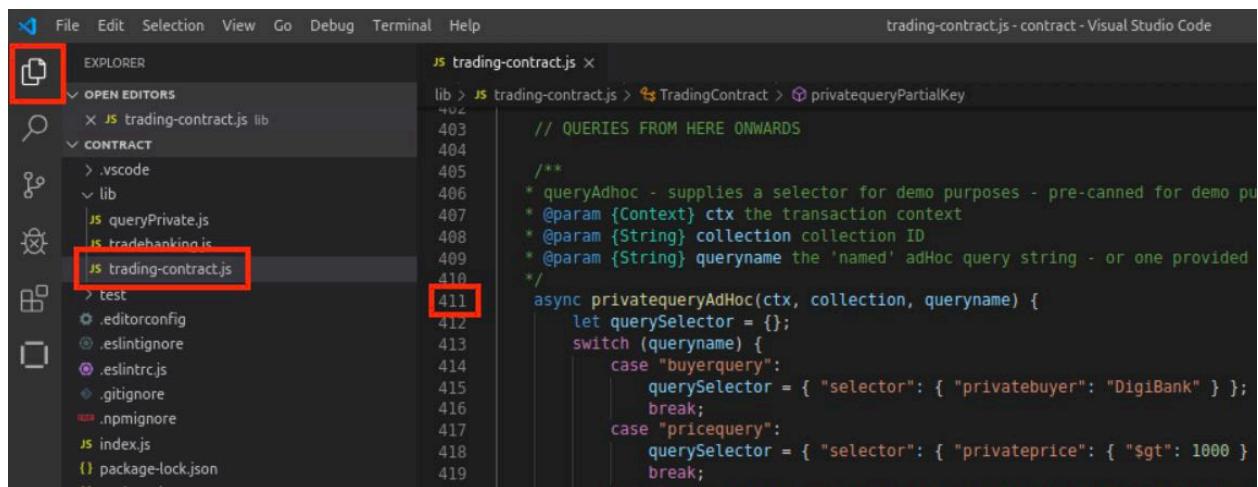
In this section of the lab we are going to focus on working with transactions to:

- Add sample data to two separate Private Data collections for query purposes – and
- Execute Private Data query transactions that allow you to perform ‘rich queries’ against the private collection data.

The demo data transaction **initLedger()** was added as part of the upgrade, as were the query transactions **privatequeryAdhoc()** and **privatequeryPartialKey**.

Let's look at the transactions:

- 140.** Click the **Explorer** icon open **trading-contract.js** if it is not already displayed.  
Scroll to line 411 in the code.



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Debug, Terminal, Help
- Explorer Sidebar:** Shows the project structure with files like `trading-contract.js`, `lib`, `test`, and configuration files (`.vscode`, `.gitignore`, etc.). The `trading-contract.js` file is selected and highlighted with a red box.
- Code Editor:** The `trading-contract.js` file is open. Line 411 is highlighted with a red box. The code includes comments for a named query and an ad-hoc query selector.
- Title Bar:** trading-contract.js - contract - Visual Studio Code

```
// QUERIES FROM HERE ONWARDS
...
* queryAdhoc - supplies a selector for demo purposes - pre-canned for demo pu
* @param {Context} ctx the transaction context
* @param {String} collection collection ID
* @param {String} queryname the 'named' adHoc query string - or one provided
*/
async privatequeryAdHoc(ctx, collection, queryname) {
    let querySelector = {};
    switch (queryname) {
        case "buyerquery":
            querySelector = { "selector": { "privatebuyer": "DigiBank" } };
            break;
        case "pricequery":
            querySelector = { "selector": { "privateprice": { "$gt": 1000 } } };
            break;
    }
}
```

The **initLedger** transaction creates four DEMO contracts that are advertised and – against those contract IDs, creates some offer and offer/accept sample data in both DigiBank’s and EcoBank’s private data collections. Next, let’s briefly review the query transactions.

The first is **privatequeryAdhoc()** – this can take either a ‘named query’ (example; “pricequery” - queries Collection D for Contracts > 1000 USD ) or it takes an ‘ad-hoc’ query string – where you create a ‘custom query’ selector string (CouchDB/Mango format) – and supply that to the **privatequeryAdhoc()** transaction itself.

```
403     // QUERIES FROM HERE ONWARDS
404
405     /**
406      * queryAdhoc - supplies a selector for demo purposes - pre-canned for demo purposes or provide ad-hoc string
407      * @param {Context} ctx the transaction context
408      * @param {String} collection collection ID
409      * @param {String} queryname the 'named' adHoc query string - or one provided as a parameter
410     */
411     async privatequeryAdhoc(ctx, collection, queryname) {
412       let querySelector = {};
413       switch (queryname) {
414         case "buyerquery":
415           querySelector = { "selector": { "privatebuyer": "DigiBank" } };
416           break;
417         case "pricequery":
418           querySelector = { "selector": { "privateprice": { "$gt": 1000 } } }; // price is in integer format
419           break;
420         default: // its assumed its a custom-supplied couchdb query selector (collection, selector)
421           // eg - as supplied as a param to VS Code: ["CollectionD","{\\"selector\\":{\\\"privateprice\\\":\\\"$lt\\\":1000}}"]
422           querySelector = JSON.parse(queryname);
423       }
424       console.log('main: querySelector stringified is:' + JSON.stringify(querySelector)); // logged for audit in the Docker container
425       let queryObj = new Query(Namespace);
426       let results = await queryObj.queryAdhoc(ctx, collection, querySelector);
427
428       return results;
429     }
```

**Line 411** accepts the named query or ad-hoc as the queryname parameter, along with the collection name. Depending on the query, the selector is a pre-built query string, or the invoker of the query supplies a querystring themselves – you will see an example of using both later. Line 425-426 sets up the query object by calling a helper transaction from the **queryPrivate.js** source (where its class is defined) and then iterates through the query results, before (**line 428**) returning the query results to the VS Code output pane in this case.

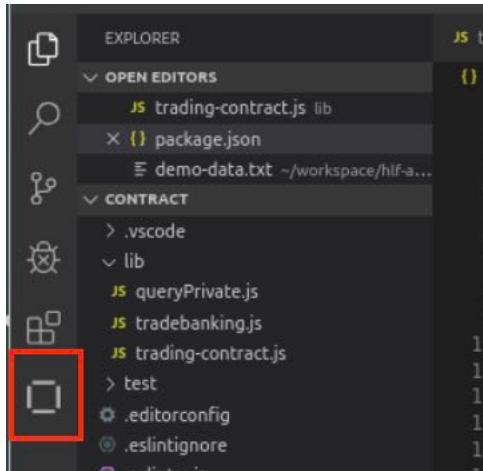
There is also a **privatequeryPartialKey** transaction that does a query of a private data collection by partial key (e.g. Asset namespace, to find all records for that namespace, in the collection supplied).

## 5.1 Create some Demo Data for querying data

We will now create demo data using the initLedger transaction, and we will be using both DigiBank and EcoBank identities to create the data. The result will be some additional records to run queries against.

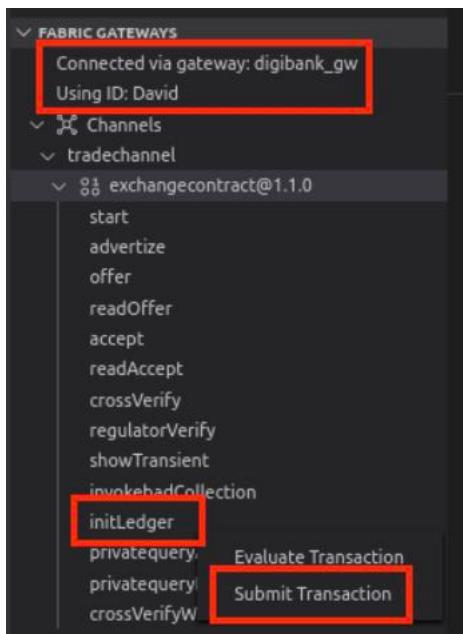
We need to return to the IBM Blockchain Platform extension.

**-- 141.** Click on the IBM Blockchain Platform extension icon.



-- 142. On the **Fabric Gateways** panel disconnect from FinReg's gateway and click **digibank\_gw** and connect with identity **David**.

-- 143. Expand the **exchangecontract** and right-click **initLedger**. Click **Submit Transaction**



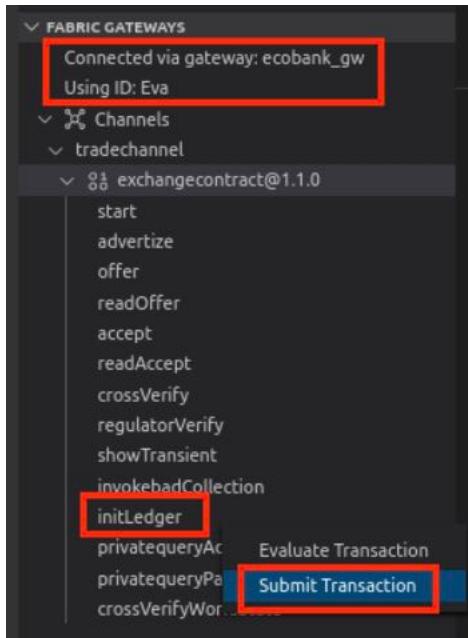
-- 144. There are no parameters, just press **enter** leaving the empty square brackets []

-- 145. There is no transient data for this transaction, just press **enter** leaving the empty curly braces {}

The transaction should return a 'success' message in the output pane, indicating the demo data was created:

Created some new demo contracts (main ledger) and private Data for DigiBank

- 146. On the **Fabric Gateways** panel disconnect from DigiBank's gateway and click **ecobank\_gw** and connect with identity **Eva**.
- 147. Scroll down to the transaction **initLedger**, right click then click **Submit Transaction**



- 148. Once again, there are no parameters, just press **enter** leaving the empty square brackets []
- 149. There is no transient data for this transaction, just press **enter** leaving the empty curly braces {}
- 150. Review the messages in the output pane – it should indicate that the private demo data was created this time for EcoBank

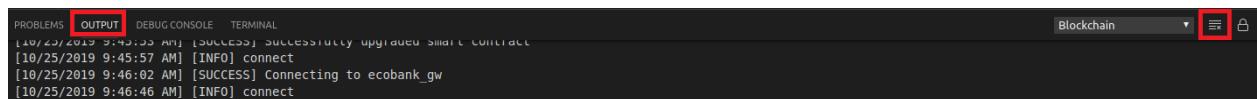
Returned value from initLedger: Create some private Data for EcoBank

We can now proceed with the next part – to try out the private data queries. In doing so, we will perform some simple queries that will query each of the private data collections for **DigiBank** and **EcoBank** – which contain both demo data and data you created during this lab exercise.

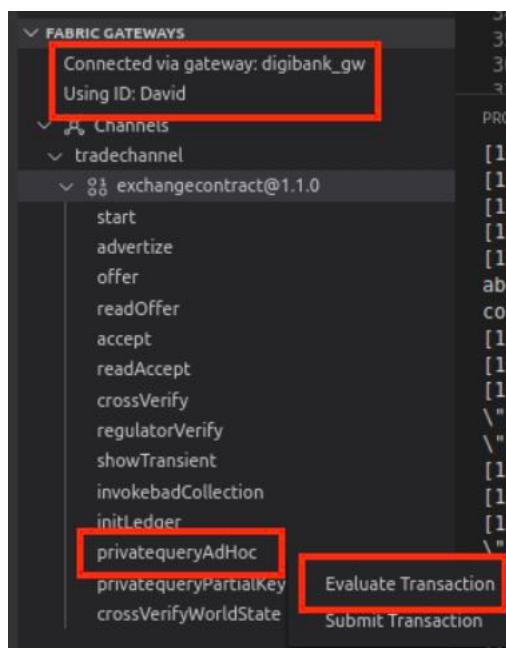
## 5.2 Executing Rich Queries against the Private Data Collections

So that we can more easily see what is happening, we're going to reset the Output pane and make it larger.

- 151. At the top right of the Output pane click on the Clear Output icon, then resize the pane up, to create a larger space for output messages.



- 152. On the **Fabric Gateways** panel disconnect from EcoBank's gateway and click **digibank\_gw** and connect with identity **David**.
- 153. Expand the **exchangecontract** and right-click **privatequeryAdhoc**. Click **Evaluate Transaction**



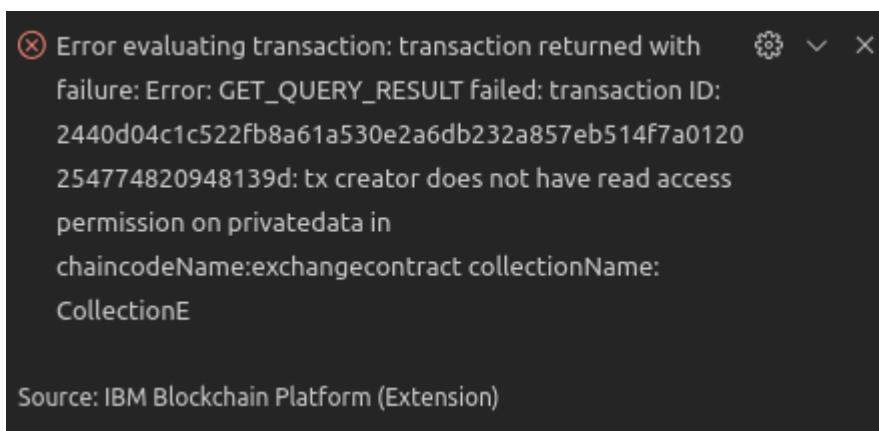
- 154. When prompted, **copy and paste** the following parameters, replacing the '[]' text offered with the following parameter list:

```
[ "CollectionE", "buyerquery" ]
```

The first parameter is the private collection name (recall that we have 2 collections CollectionD and CollectionE). The second parameter is the name of the query that we wish to run.

- \_\_ 155.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces {}

Notice that we provided **Collection ‘E’** in the parameter list – to which DigiBank does not have access – so we will get a popup error indicating the query could not be performed (access denied)



- \_\_ 156.** Close the error popup message, and clear the Output pane using the **Clear Output** icon, as shown previously.

- \_\_ 157.** Once again, right-click **privatequeryAdhoc** and click **Evaluate Transaction**.

- \_\_ 158.** When prompted, paste in the following parameters, replacing the ‘[]’ text offered with the following ‘named’ query called ‘buyerquery’ :

```
[ "CollectionD", "buyerquery" ]
```

Review the output pane showing your query results.

Let's try another query, a price query for a 'privateprice' value that's greater than 1000 USD

-- **159.** Once again, right-click **privatequeryAdhoc** and click **Evaluate Transaction**.

-- **160.** When prompted, paste in the following parameters, replacing the '[]' text offered with the following 'named' query, 'pricequery' :

```
[ "CollectionD", "pricequery" ]
```

-- **161.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces {}

-- **162.** Review the output pane for messages – you should see your query results – a screenshot of sample output is shown below – you will see different results. Missing from these results are two records from the earlier listed results that are under 1000 USD. This is because our example pricequery is hard coded to show only values over 1000.

```
[10/25/2019 1:52:57 PM] [INFO] evaluateTransaction
[10/25/2019 1:53:10 PM] [INFO] evaluating transaction privatequeryAdHoc with args CollectionD,pricequery on channel tradechannel
[10/25/2019 1:53:10 PM] [SUCCESS] Returned value from privatequeryAdHoc: [{"privatebuyer":"DigiBank","privateprice":9999}, {"privateprice":20000}].{"privatebuyer":"DigiBank","privateprice":30000}]
```

Next you will try some queries as EcoBank.

-- **163.** On the **Fabric Gateways** panel disconnect from DigiBank's gateway and click **ecobank\_gw** and connect with identity **Eva**.

-- **164.** Expand the **exchangecontract** and right-click **privatequeryAdhoc**. Click **Evaluate Transaction**

-- **165.** When prompted, paste in the following parameters, replacing the '[]' text offered with the following – and press **enter**

```
[ "CollectionE", "pricequery" ]
```

-- **166.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces {}

-- **167.** Review the messages in the output pane – you should see the results show the additional 'accept' data that's only recorded in EcoBank's private data collection.

```
[10/25/2019 1:25:52 PM] [SUCCESS] Connecting to ecobank gw
[10/25/2019 1:26:13 PM] [INFO] evaluateTransaction
[10/25/2019 1:26:40 PM] [INFO] evaluating transaction privatequeryAdHoc with args CollectionE,pricequery on channel tradechannel
[10/25/2019 1:26:40 PM] [SUCCESS] Returned value from privatequeryAdHoc: [{"acceptprice":9999,"privatebuyer":"DigiBank","privateprice":9999}, {"privateprice":20000}].{"acceptprice":30000,"privatebuyer":"EcoBank","privateprice":30000}]
```

Finally, let's supply an 'ad-hoc' query selector query string to the **privatequeryAdhoc** transaction – this enables a user to supply a custom query – in this case, to see the accept/offer records, whose values are less than 1000 USD.

**-- 168.** Once again, right-click **privatequeryAdhoc** and click **Evaluate Transaction**.

**-- 169.** When prompted, paste in the following parameters, replacing the '[]' text offered, with the following string, pasted in exactly as shown – and press **enter**:

```
[ "CollectionE", "{\"selector\":{\"privateprice\":{\"$lt\":1000}}}" ]
```

**-- 170.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces {}

**-- 171.** Review the messages in the output pane – you should see a different set of results, which are the two records that match the criteria we provided ("less than 1000").

```
[10/25/2019 2:04:21 PM] [INFO] connect
[10/25/2019 2:04:25 PM] [SUCCESS] Connecting to ecobank_gw
[10/25/2019 2:04:33 PM] [INFO] evaluateTransaction
[10/25/2019 2:04:39 PM] [INFO] evaluating transaction privatequeryAdHoc with args CollectionE,{"selector":{"privateprice":{"$lt":1000}}}
[10/25/2019 2:04:39 PM] [SUCCESS] Returned value from privatequeryAdHoc: [{"acceptprice":600,"privatebuyer":"EcoBank","privateprice":600}, {"acceptprice":900,"privatebuyer":"EcoBank","privateprice":900}]
```

This concludes the lab steps for working with Private Data queries.

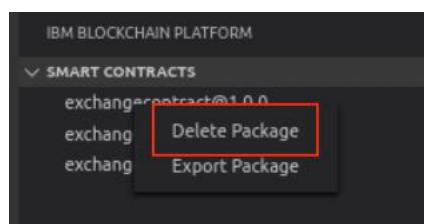
## Review

You've now successfully:

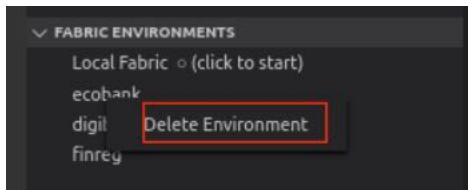
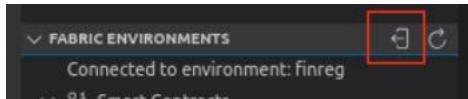
- created some additional demo or sample data
- performed queries against private data collections
- understood more about using rich queries to filter result sets, by using different criteria.

Finally you need to cleanup the virtual machine in preparation for the next lab.

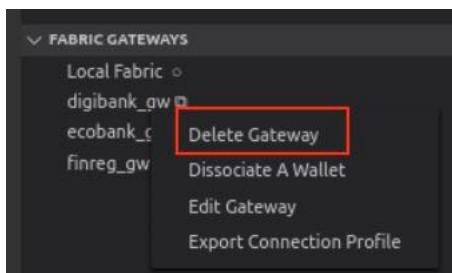
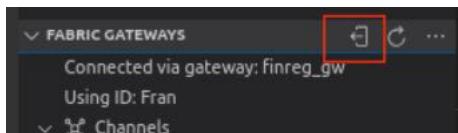
**-- 172.** In the **Smart Contracts** panel right-click each package in turn and click **Delete Package**.



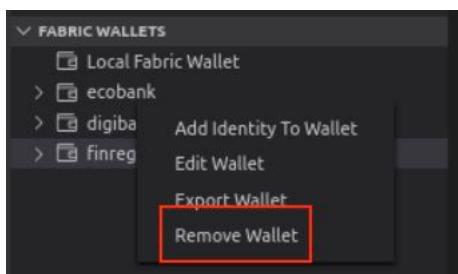
- 173. In the **Fabric Environments** panel **Disconnect** the current environment, and then **right-click** each environment and click **Delete Environment**.



- 174. In the **Fabric Gateways** panel **Disconnect** the current gateway, and then **right-click** each gateway in turn and click **Delete Gateway**.



- 175. In the **Fabric Wallets** panel **right-click** each wallet in turn and click **Remove Wallet**.



- 176. In the terminal window, use the following two commands to navigate to the **hlf-ansible-master** folder and teardown the custom network using a bash script:

```
cd ~/workspace/hlf-ansible-master
```

```
./teardown.sh
```

```
blockchain@ubuntu:~/workspace/hlf-ansible-master/exchangecontract$ cd ~/workspace/hlf-ansible-master/
blockchain@ubuntu:~/workspace/hlf-ansible-master$ ./teardown.sh
```

**Congratulations!**

## We Value Your Feedback!

- Please ask your instructor for an evaluation form. Your feedback is very important to us as we use it to continually improve the lab material.
- If no forms are available, or you want to give us extra information after the lab has finished, please send your comments and feedback to “**blockchain@uk.ibm.com**”

## Appendix 1 - transaction List Info for 'exchangecontract'

A table containing the smart contract transaction names, parameters and main objective of each – is shown below.

Fields in BOLD / RED are the private data elements, the remainder are stored in the ledger world state. The contract ID represents the trade commodity contract, being bought/sold/traded on the exchange marketplace FYI. Note that you may have seen that both private and public data are passed in the transient data set in the lab exercises.

transaction	Action	Parameters / {Transient Data}	Summary Objectives
advertize	submit	contractID, seller, symbol, quantity, reserveprice, marketdate	'Broadcast' the commodity for sale on the marketplace. The composite key is made up of an Assetspace (domain) + Contract ID
offer	submit	contractID, { <b>buyer</b> , <b>offerprice</b> , offerdate }	The buyer's bank (ie on behalf of his broker), makes an offer on the advertised commodity – its key is retrieved exactly as described above.
readOffer	evaluate	contractID	Helper transaction: The beholder of the actual private data, can use this simple transaction, to see what's written to 'its' Org's collection
accept	submit	contractID, { <b>buyer</b> , <b>offerprice</b> , <b>acceptprice</b> , acceptdate }	The seller's bank, notified by the buyer's bank, accepts the offer on behalf of his client – a status of UNVERIFIED will be written, pending a cross-verification to see the offer is genuine
readAccept	evaluate	contractID	Helper transaction: The beholder of the actual private data, can use this simple transaction, to see what's written to 'its' Org's collection

crossVerify	submit	Collection ID, contractID, calculated_hash	The seller's bank, must verify the integrity of the offer hash, written on the blockchain. It uses the offer data it received 'out-of-band', to calculate a hash and cross-verify the offer hash on the ledger. (Note: The regulator will also use this to verify both the offer/accept private data, later on)
invokebadCollection	evaluate	<none>	A transaction to show what the user would see (in VS Code) if an invalid collection is invoked in the smart contract.
privatequeryAdhoc	evaluate	collection, <<queryname <u>or</u> query selector>>	Query the private data by 'name' (by buyer, by offer price etc) or supply query selector (Mango) as a parameter
privatequeryPartialKey	evaluate	Assetspace prefix	Find all contracts in a private data collection, with given prefix

## Appendix 2 Access control considerations for Private data

If the private data is relatively simple and predictable (e.g. transaction dollar amount), channel members who are not authorized to the private data collection could try to guess the content of the private data via brute force hashing of the domain space, in hopes of finding a match with the private data hash on the chain. Private data that is predictable should therefore include a random “salt” that is concatenated with the private data key and included in the private data value, so that a matching hash cannot realistically be found via brute force. The random “salt” can be generated at the client side (e.g. by sampling a secure pseudo-random source) and then passed along with the private data in the transient field as transient data, at the time of chaincode invocation. For more information, see the Fabric documentation:

<https://hyperledger-fabric.readthedocs.io/en/latest/private-data-arch.html#access-control-for-private-data>

## Appendix 3 - Alternate Trade Network Fabric

In case of problems setting up the Fabric in Part 1 of this lab document, an alternate Fabric can be started using these steps. This Alternate Fabric should only be used at the explicit instruction of the Instructor.

1. In VS Code disconnect any connected Fabric Environment.
2. Right click on any Fabric Environment for Ecobank, digibank, finreg and **Delete Environment** (Remember to click **yes** in the bottom corner of the screen)
3. Disconnect any connected Fabric Gateway.
4. Right click on any Fabric Gateway for Ecobank, digibank, finreg and **Delete Gateway**
5. In a terminal window, run the following commands to cleanup the “regular” Fabric:  
`cd ~/workspace/hlf-ansible-master  
./teardown.sh`
6. Run these commands in the terminal window to work with the BackupFabric:  
`cd ~/workspace/hlf-ansible-master  
tar -xzf backupFabric.tar.gz  
cd backupFabric  
./start.sh`  
wait for the output to end with “Successfully submitted channel update”
7. In VS Code, hover over **Fabric Wallets**, and click the “+” sign to **add wallet**.  
At the top of the screen click the option to **Specify an existing file system wallet**  
Click **Browse** then navigate to the folder:  
**home/blockchain/workspace/hlf-ansible-master/backupFabric/adminWallets**  
and click **ecobank** and then **select**
8. In the same way add the new wallets for digibank and finreg.
9. Hover over the **Fabric Environments** and click the “+” sign to **add environment**  
At the top of the screen specify **ecobank** as the name  
Click **Browse** then navigate to the folder:  
**home/blockchain/workspace/hlf-ansible-master/backupFabric**  
and click **Ecobank-nodes** and then **select**

**Click done adding nodes**

10. Click on the **ecobank** environment and verify that it connects to the Fabric
11. Disconnect from the ecobank environment and using the same technique add the environments for **digibank** and **finreg**
12. Hover over **Fabric Gateways**, and click the “+” sign to **add gateway**.  
At the top of the screen click the option to **Create a gateway from a Fabric environment**  
Click **ecobank** as the environment to create from  
Press enter to accept **ecobank\_gw** as the name  
Click **caecobank.ecobank.example.com** as the ca for the gateway  
Click on the new **ecobank\_gw** and click **Eva** as the id for the connection
13. Disconnect from the **ecobank\_gw** gateway, and using the same technique add the gateways for **digibank\_gw** and **finreg\_gw**. When testing the gateways use the IDs **David** and **Fran** respectively.
14. Return to the main lab instructions at **Part 2 – Deploy a Base Version of the Smart Contract ...**