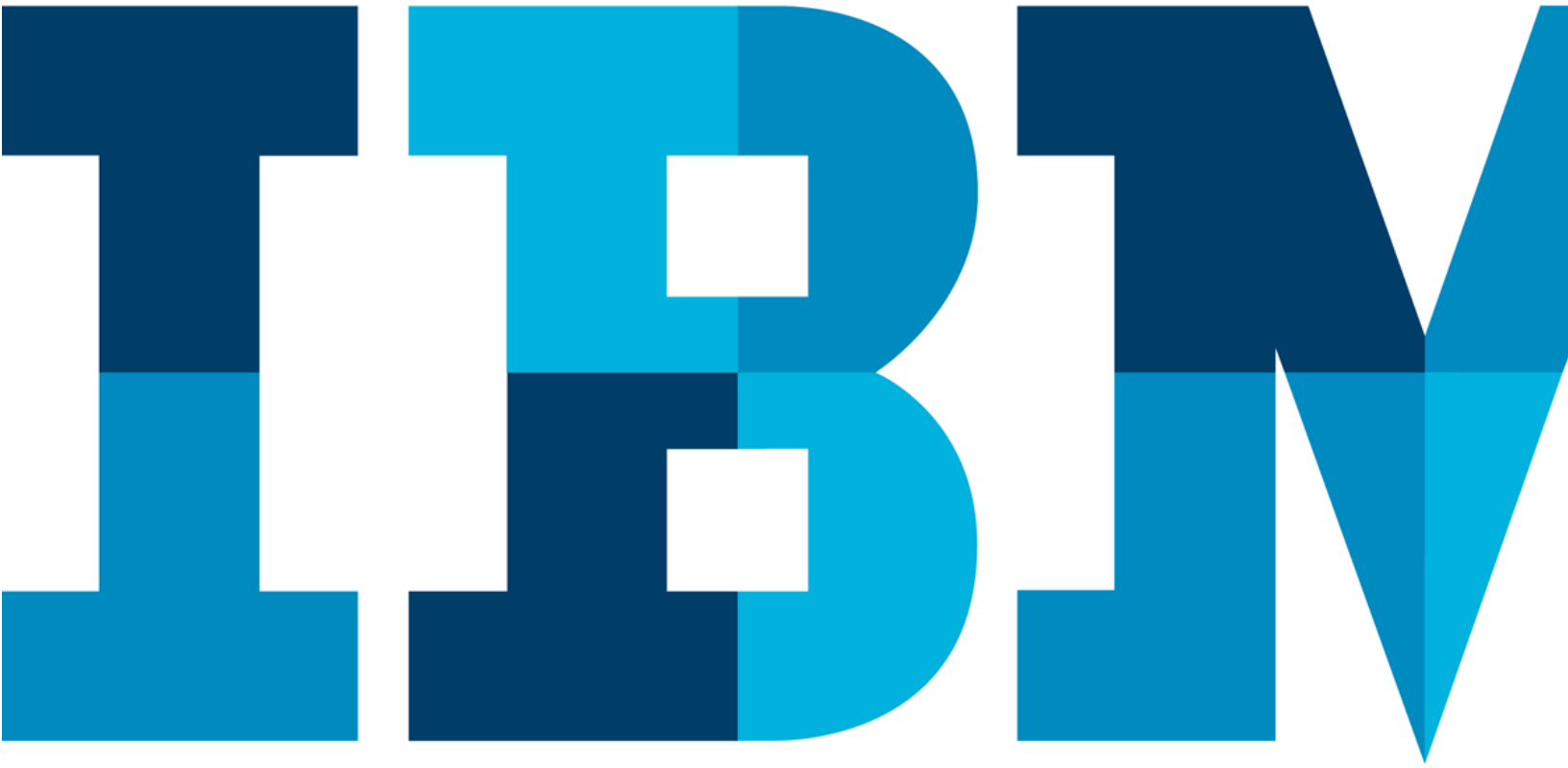
IBM Blockchain Platform Hands-On

Lab 4:

Verifiable Exchange with Private Data Collections



IBM

Table of Contents

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.

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 4th 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.

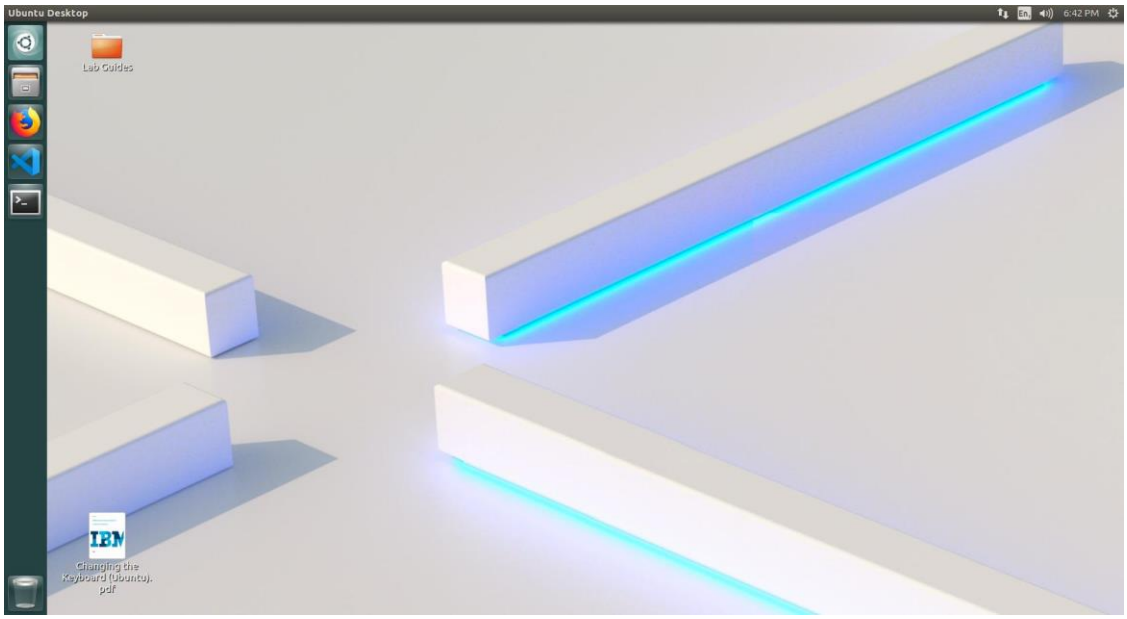
***Note**: The screenshots in this lab guide were taken using version **1.39.2** of **VS Code**, and version **1.0.12** of the **IBM Blockchain Platform** extension. If you use different versions, you may see differences from those shown in this guide.*

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**".



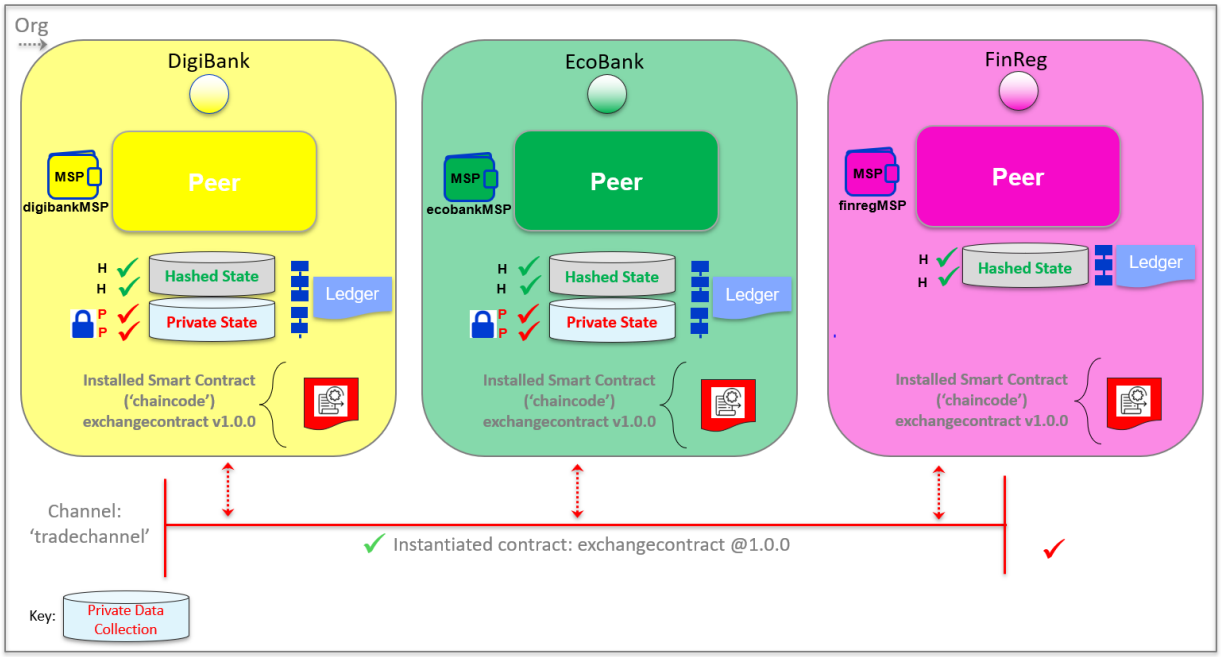
Simplified Commodity Trading Exchange

DigiBank and **EcoBank** participate in a Commodity Exchange marketplace – the buying and selling of **commodity contracts**It is a regulated marketplace that facilitates the trading of contracts whose total value is tied to the price of commodities (e.g., oil, corn, silver, gold). In general, a Contract is put up for sale by advertising it on the marketplace. Offers (and counter offers) are made, before finally an offer is accepted by the seller and a deal is made. The offer transaction, and the accept transaction are private to the buyer / seller respectively. It remains for the seller to cross-verify the offer details against the private data hash written by the buyer to the ledger. If they match, the seller can confirm the sale as cross-verified, as due process.

In this lab example we see that EcoBank and DigiBank have independent 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:

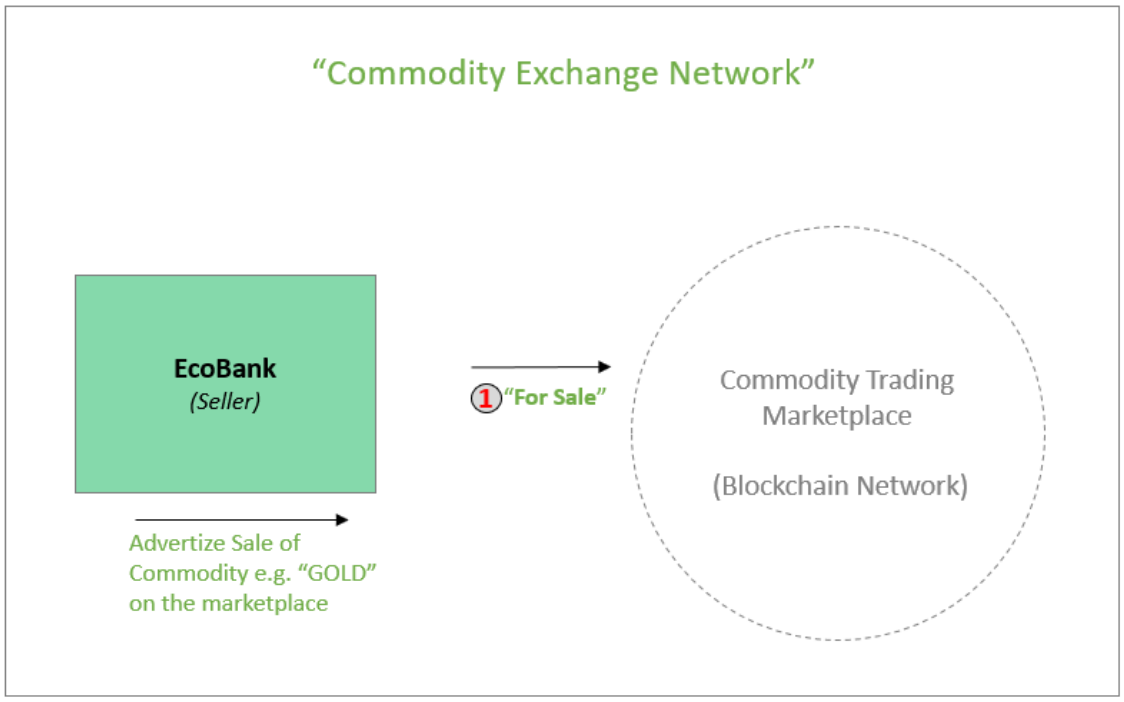
Commodity Trading Exchange: Overview



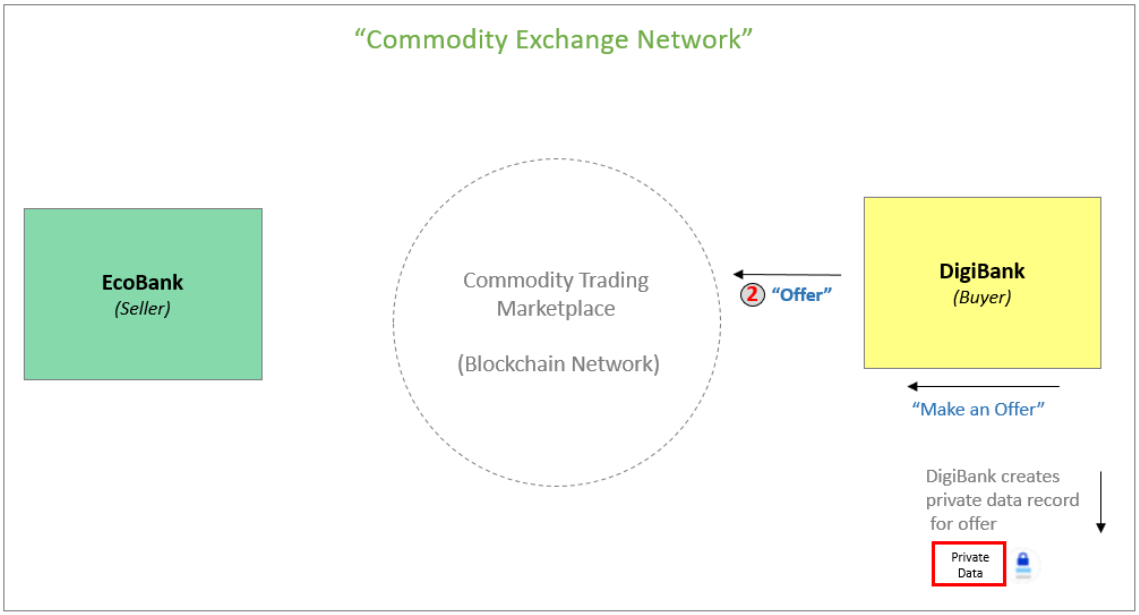
Notice in the above diagram, the regulator **FinReg** does not store any private data of its own – it does however have a copy of the ledger including the hashed state ledger. Its role is to audit records – e.g. third-party verification or dispute resolution of private data, shared with it. The hash 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 – let's examine the transaction in the lab:

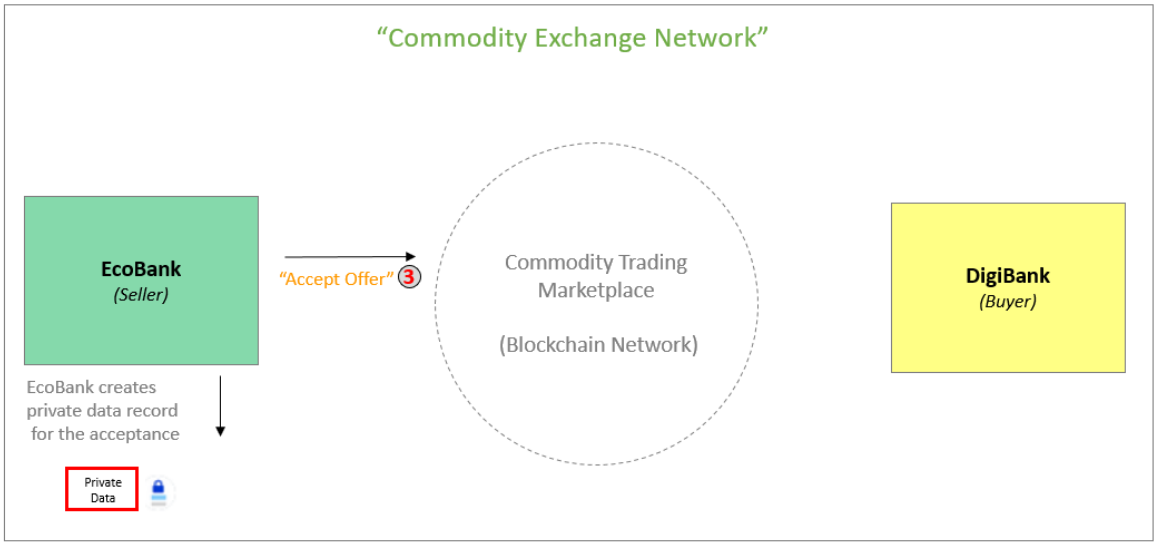
1. **EcoBank** advertises or 'broadcasts' a commodity contract for sale on the marketplace



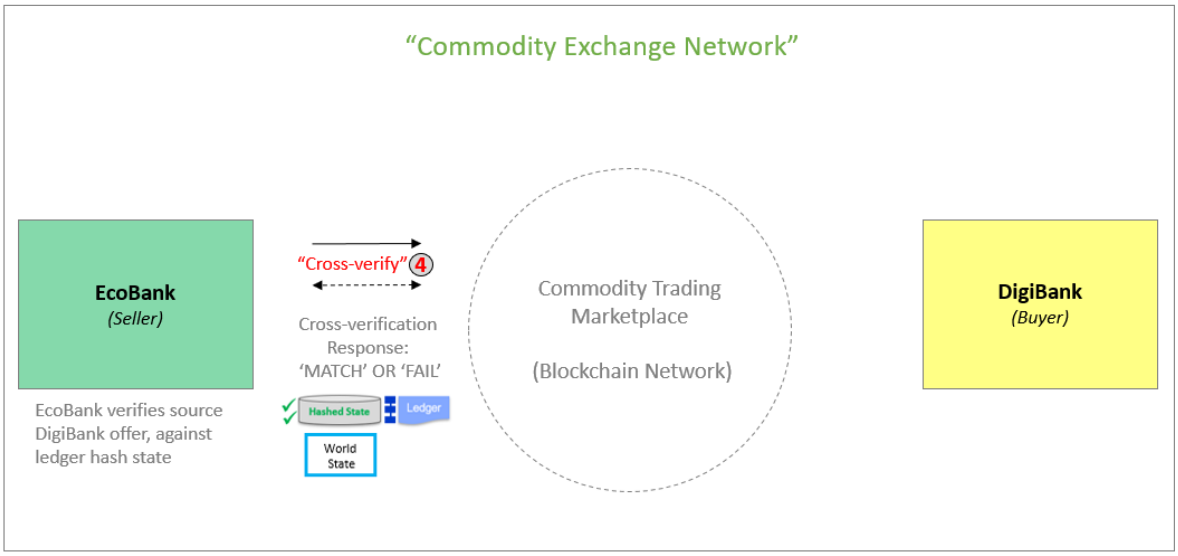
2. **DigiBank** makes an offer to EcoBank. Note that DigiBank writes a record of this offer, to its own private data store. A cryptographic 'hash' of the record is also recorded on the blockchain (channel world state).



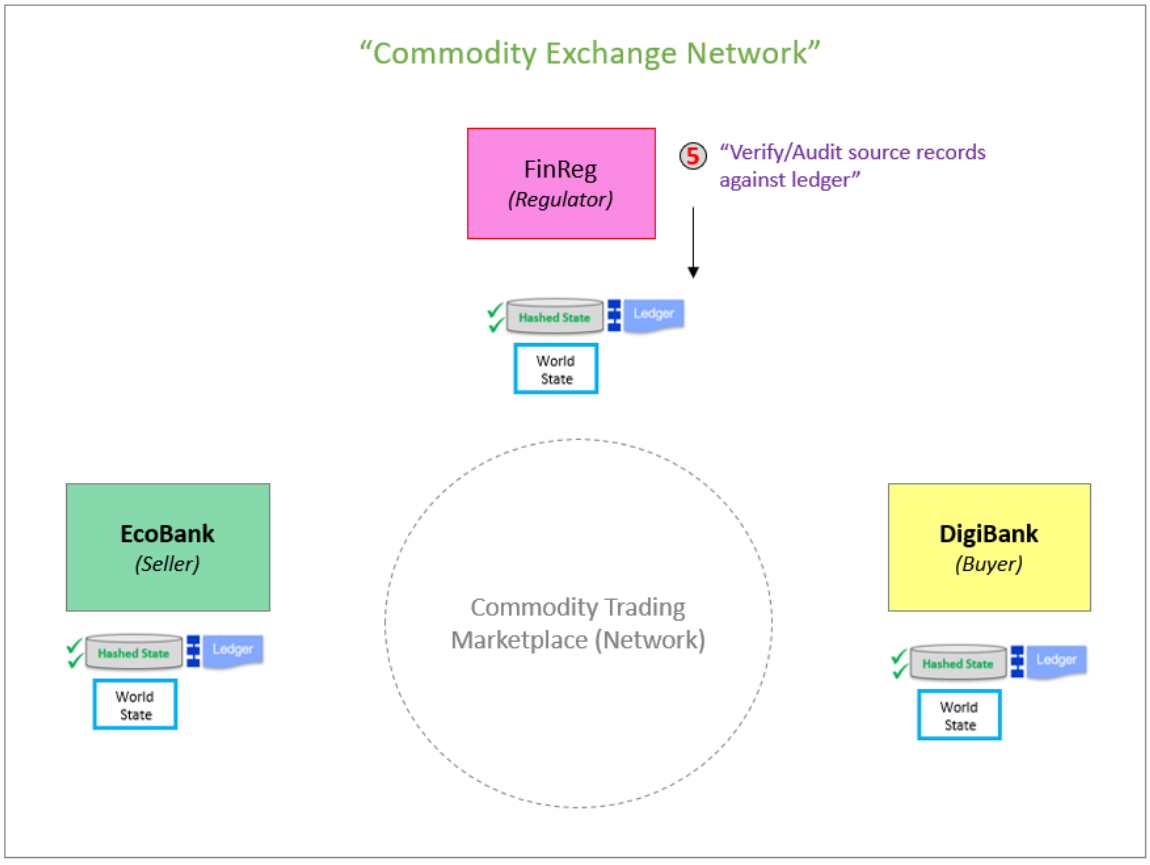
3. **EcoBank** receives the offer and informs DigiBank that it accepts the offer; EcoBank writes its own 'accept with offer' record to its own private data store – the offer on the ledger is as yet UNVERIFIED – meaning, it needs to be cross-verified.



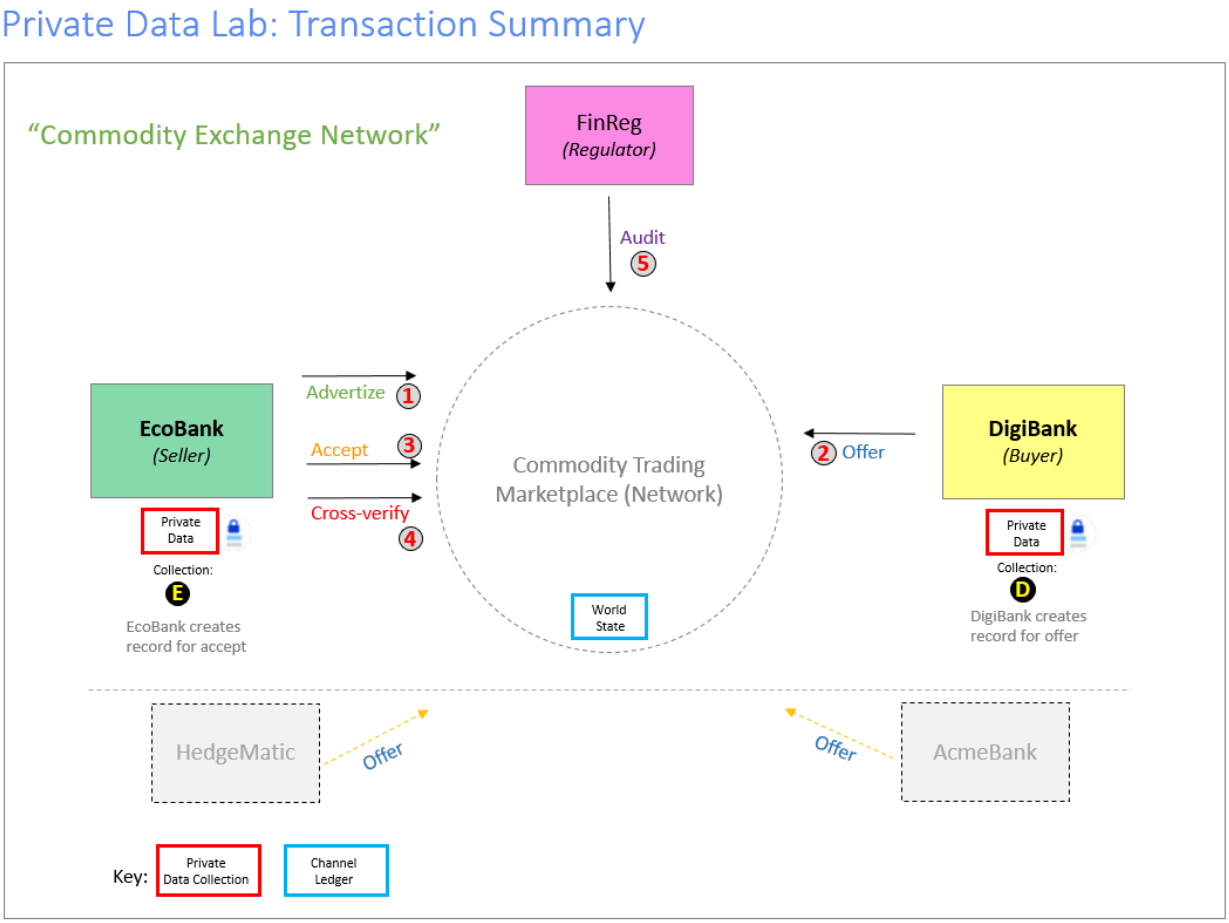
4. **EcoBank** calculates a hash of the source offer private data and verifies this hash, against the private data hash that was written by DigiBank in step 2. The smart contract compares the hashes and if they match, will update the accept/offer to 'CROSSVERIFIED' – this status attribute is not actually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



5. **FinReg**, a 3rd 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:



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.

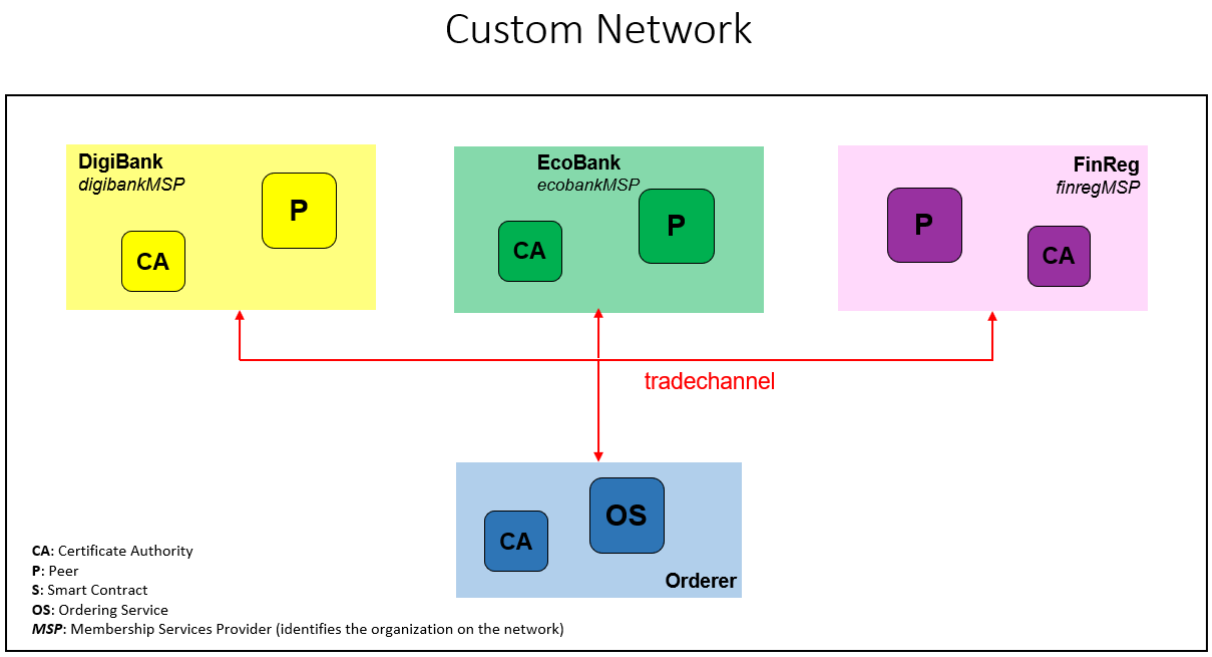
FinReg, the regulator – will, later in the lab, require to validate historical offers or accepts, to see they are the genuine, source data that each organisation (DigiBank, EcoBank) recorded, independently.

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 is a 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 IBM Blockchain Platform VS Code extension in the future to allow more realistic topologies to be available to developers.



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 5 parts.

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

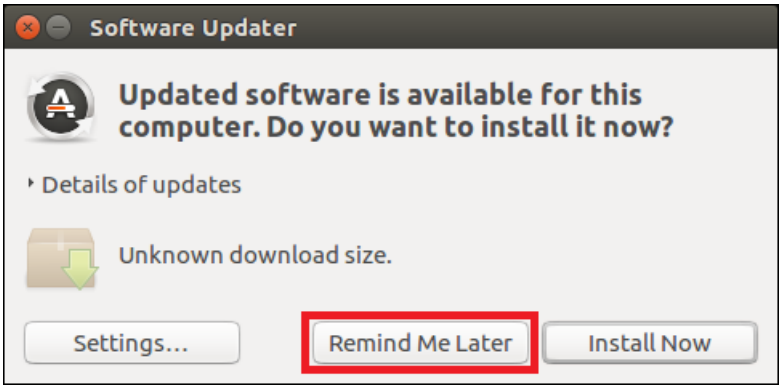
Part 2 of this lab will deploy the first version of the Smart Contract that is used to test and prove the Trade Network Fabric

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.

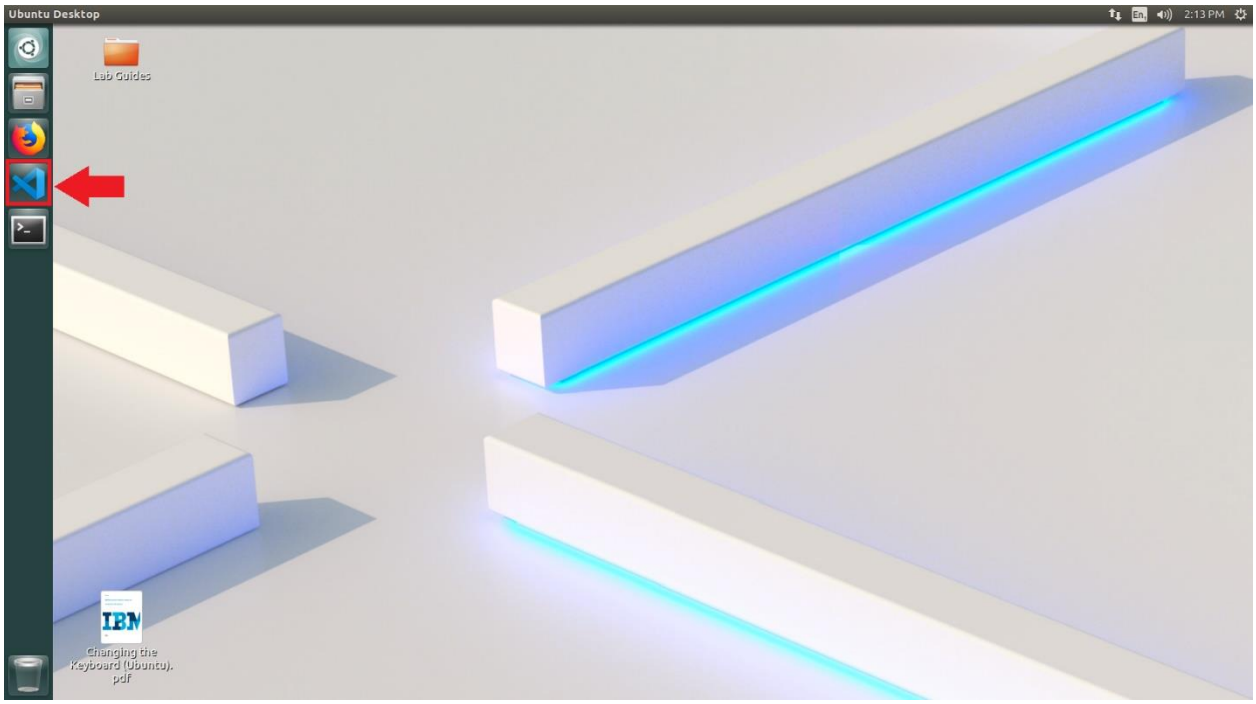
Part 5 adds additional sample private data for queries, and works with rich query transactions, running against the Private Data Collections

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

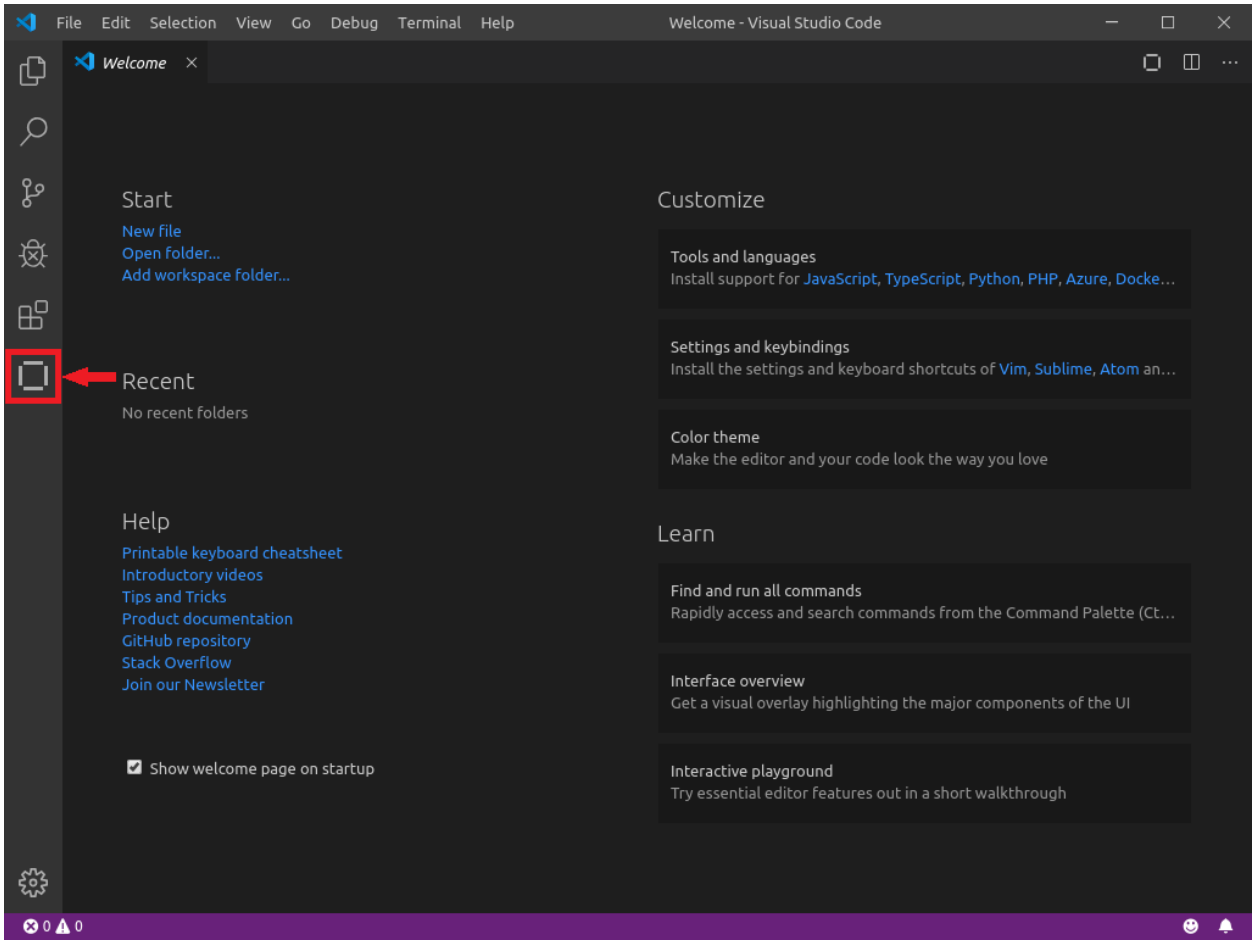


1 Build and Start Fabric for the ‘Trade Network’

__ **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.

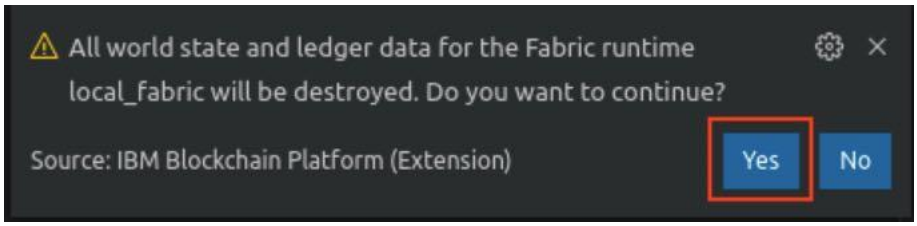


__ **4.** When VS Code opens, click on the IBM Blockchain Platform icon in the Activity Bar in VS Code as shown below.



__ **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.

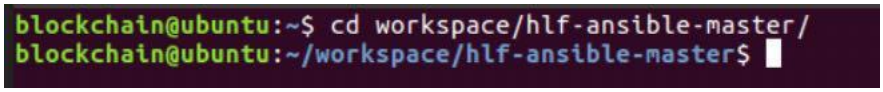


We are now going to start the custom Hyperledger Fabric network. This VM includes a script to automate the process.

__ **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/
```



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
```

(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
```

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

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.

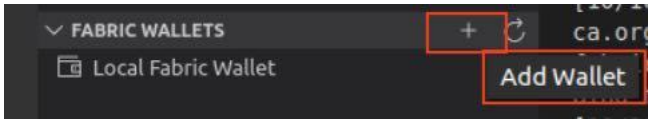
```
docker ps --format 'table {{.Names}}:\t {{.Ports}}'
```

```
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
```

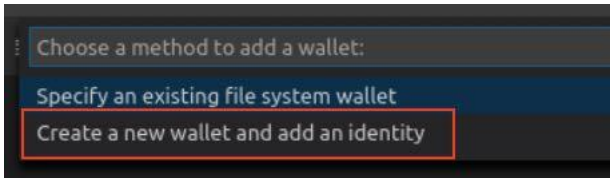
The output from the command should confirm that the network is now created and running ; it's time to create wallets and import identities into them. This will allow us to access the running network from within the IBM Blockchain Platform extension in VS Code.

First, create the wallet and import identities for **EcoBank**

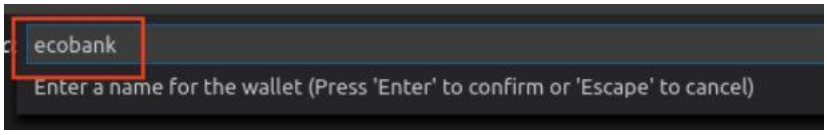
__ **10.** Back in VS Code, hover the mouse over the '**+**' sign in the Fabric Wallets panel and click **Add Wallet.**



__ **11.** At the top of the screen click the option to **Create a new wallet and add an identity**

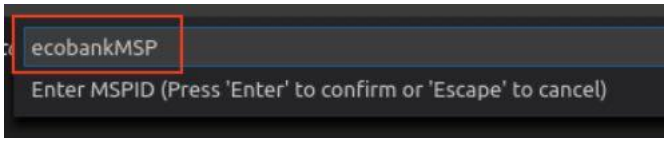


__ **12.** Specify **ecobank** as the name of the wallet (it is important to name this exactly with all lowercase letters)

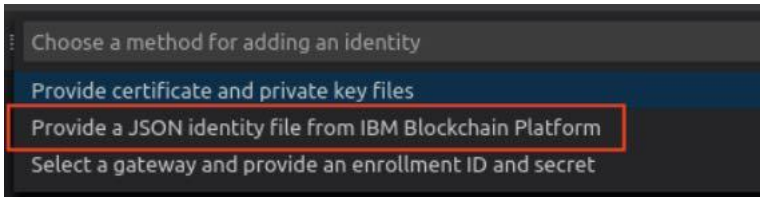


__ **13.** Specify **admin** as the name of the identity

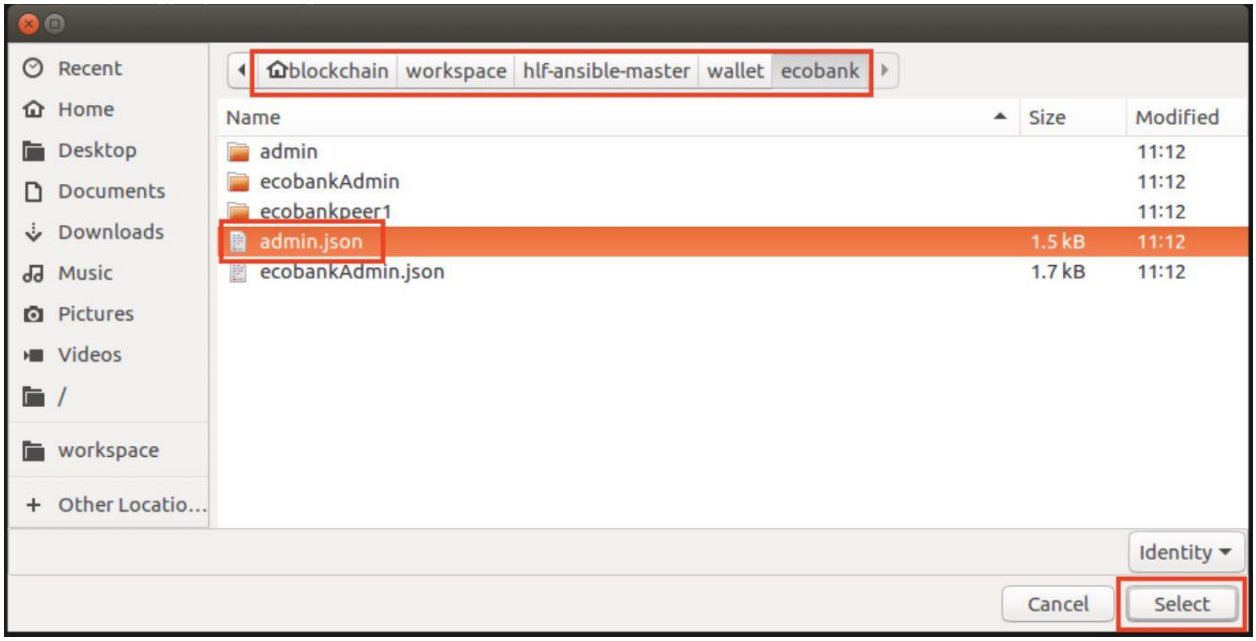
__ **14.** Specify **ecobankMSP** as the MSPID



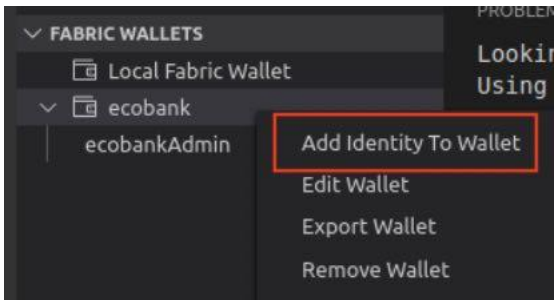
__ **15.** Select the option to Provide a JSON identity from IBM Blockchain Platform



__ **16.** 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**

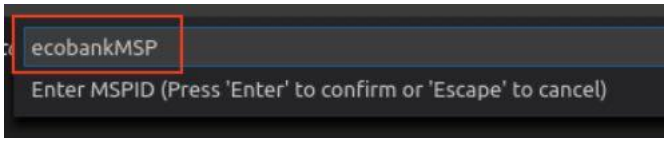


__ **17.** Right click the ecobank wallet icon on the **ecobank** wallet to add another identity

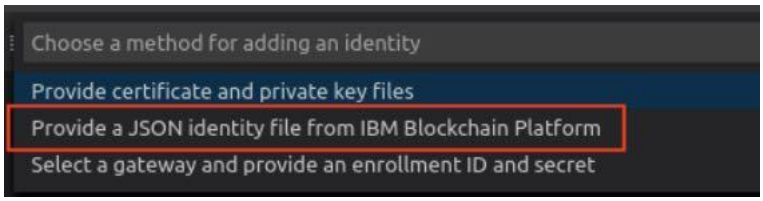


__ **18.** Specify **ecobankAdmin** as the identity name and press enter

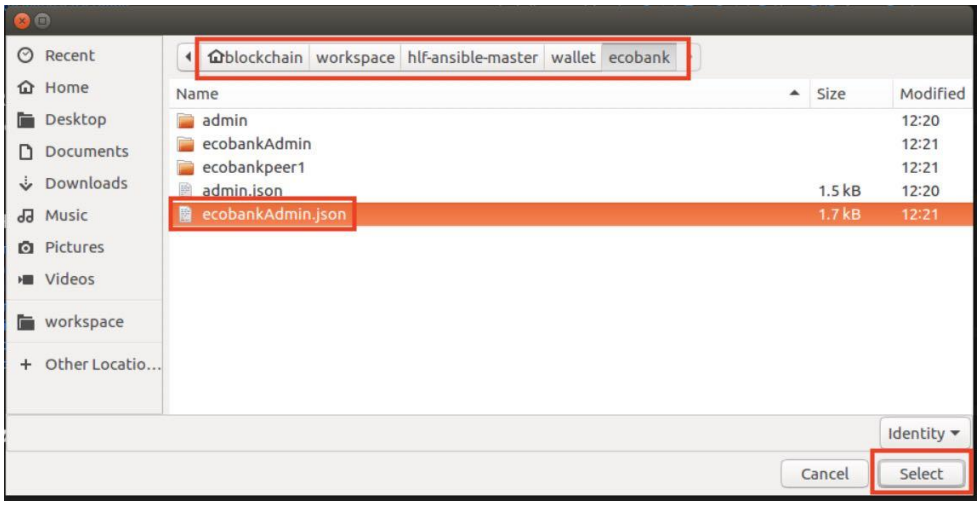
__ **19.** Specify **ecobankMSP** as the MSPID



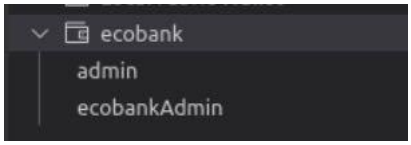
__ **20.** Select the option to Provide a JSON identity from IBM Blockchain Platform



__ **21.** 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**



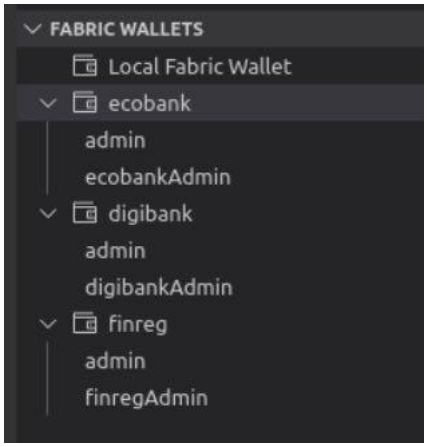
The newly created wallet and two imported identities will be shown in the Fabric Wallets panel:



__ **22.** Using Steps *12 – 21* 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

__ **23.** Using Steps *12 – 21* 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

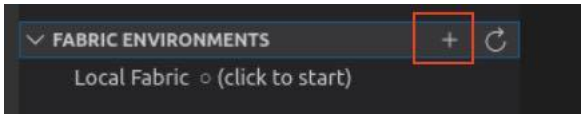
When all the wallets are created and the identities imported, they will display as follows:



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.

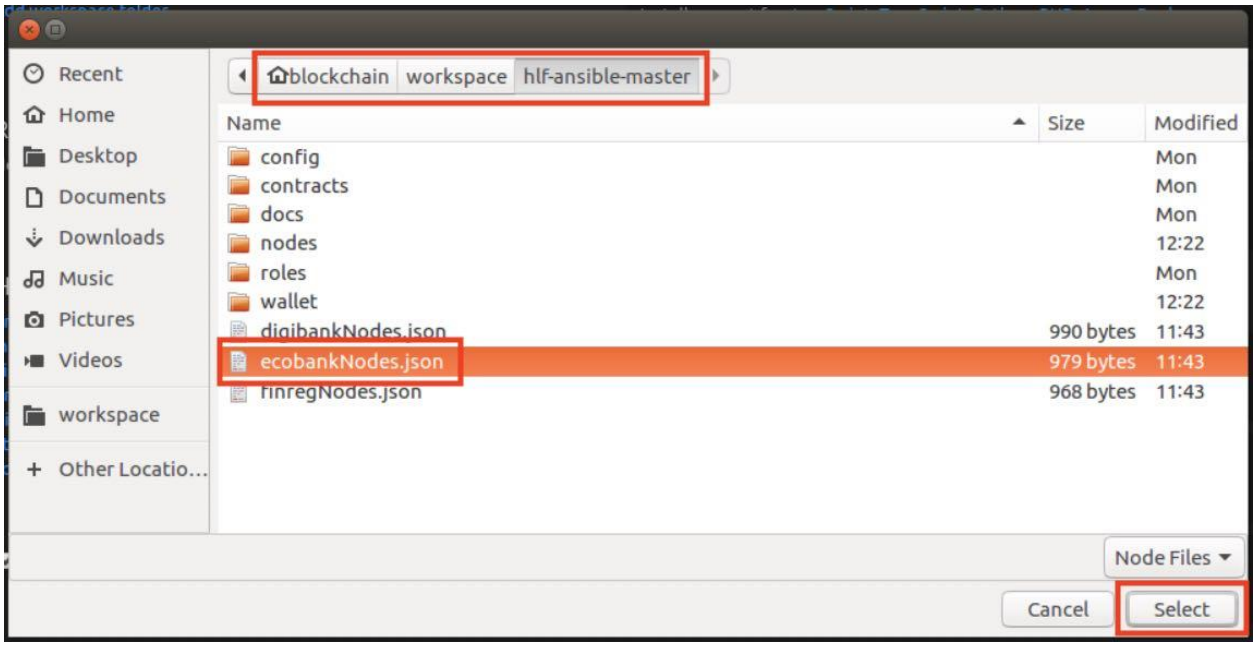
We will first create the Fabric Environment for EcoBank.

__ **24.** Import "Nodes" file to create a Fabric Environment for EcoBank.
On the Fabric Environments Panel click the '+' icon to add a new environment.

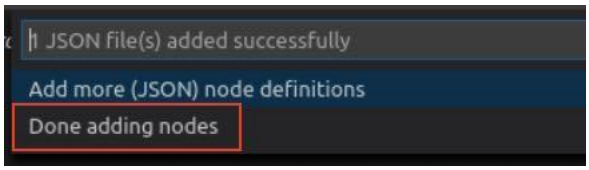


__ **25.** At the top of the screen type **ecobank** as the name of the new environment and press enter.

__ **26.** Click **Browse** and then navigate to the location:
"Home" > workspace > hlf-ansible-master
Click to highlight **ecobankNodes.json** click **Select**

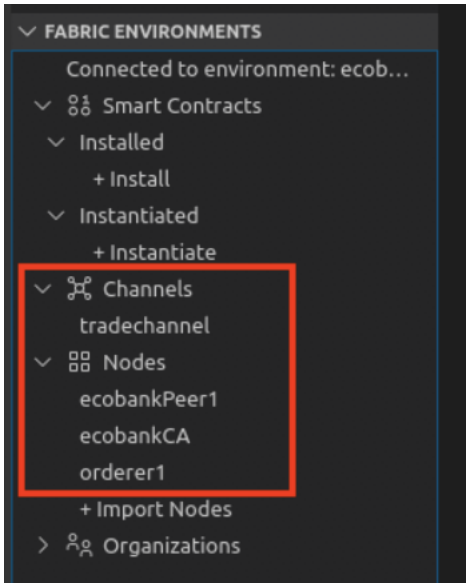


__ **27.** The JSON file you selected contains definitions for Peer, Certificate Authority and Orderer, so there are no more files to import for EcoBank. At the top of the screen 1 file is shown as imported – click **Done adding nodes**



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.

__ **28.** Click on the newly created **ecobank** environment to check that it connects as expected, and expand the **Channels** and **Nodes** twisties.



__ **29.** Hover over the Fabric Environments panel, and click the icon to disconnect from the Ecobank environment

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

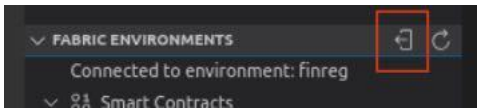
__ **31.** Again, using steps 25 – 30 as a guide, create a Fabric Environment for **finreg** using the file "Home" > workspace > hlf-ansible-master > **finregNodes.json**.

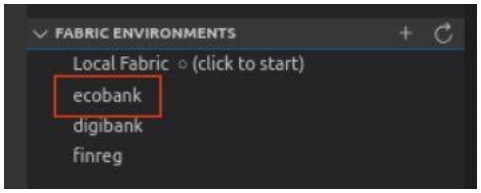
These environments will now be used to create additional (non-administrator) IDs to allow more realistic testing of the smart contract. These IDs are

Eva for EcoBank
David for DigiBank and
Fran for FinReg

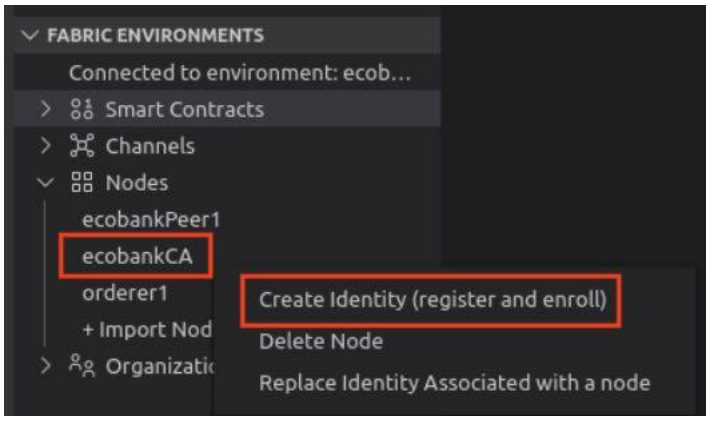
We will start by creating the ID for Eva.

__ **32.** Disconnect from any connected Fabric Environment and click on **ecobank** to connect to the EcoBank environment.



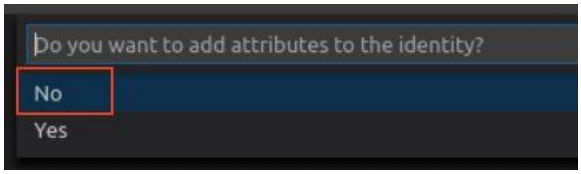


__ **33.** Expand the **Nodes**, right-click on the **ecobankCA** and then click **Create Identity (register and enrol)**

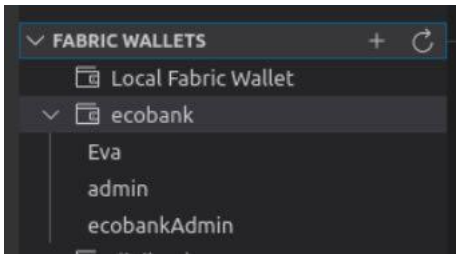


__ **34.** Specify **Eva** as the identity

__ **35.** There are no attributes required for this lab – click **No**



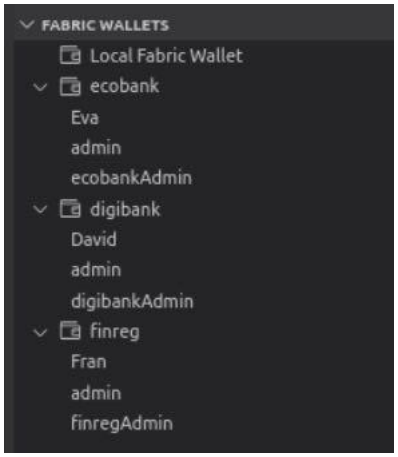
The Identity **Eva** has been added to the **ecobank** wallet



__ **36.** Using steps *33 – 35* as a guide, connect to the **digibank** environment and Create and Identity **David**

__ **37.** Using steps *33 – 35* as a guide, connect to the **finreg** environment and Create and Identity **Fran**

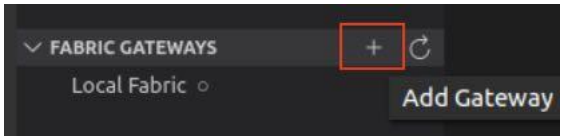
The complete list of wallets and identities for the 3 organisations is shown below



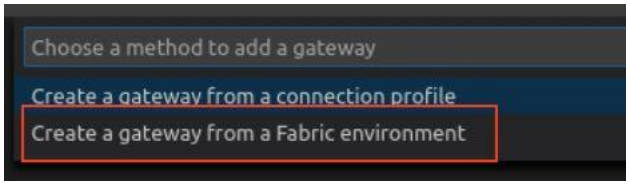
The identities we have just created will connect to the network using ***Gateways***.

We'll first add the gateway for Ecobank.

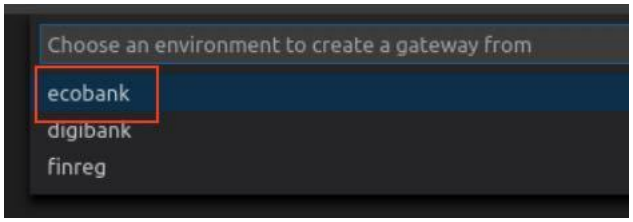
__ **38.** Add the Gateway for **ecobank**
Hover over Fabric Gateways and click the '+' to add a gateway



__ **39.** The simplest way to create a Gateway is to base the definition on an existing Environment. Click **Create a gateway from a Fabric environment**

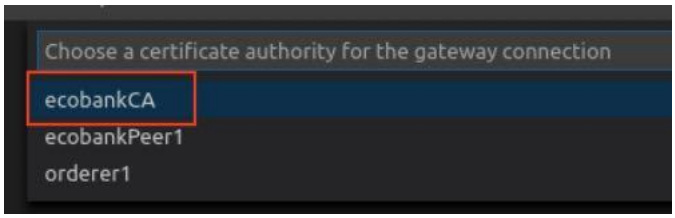


__ **40.** Click **ecobank** as the environment to copy from

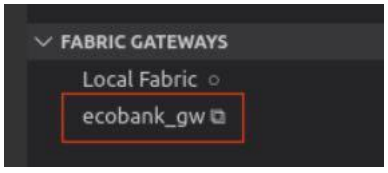


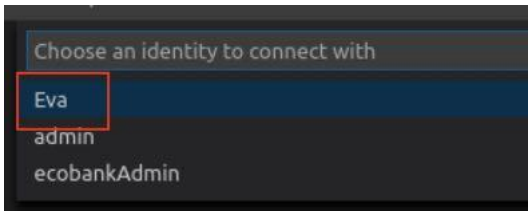
__ **41.** Use the suggested name **ecobank_gw** and press **enter**

__ **42. Click** the **ecobankCA** as the certificate authority

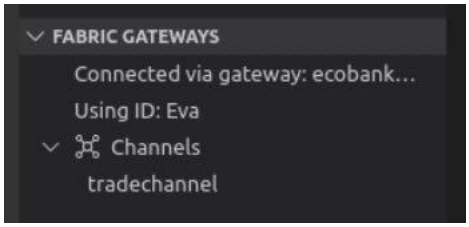


__ **43. Click** on the new gateway **ecobank_gw** and choose **Eva** as the identity to connect with

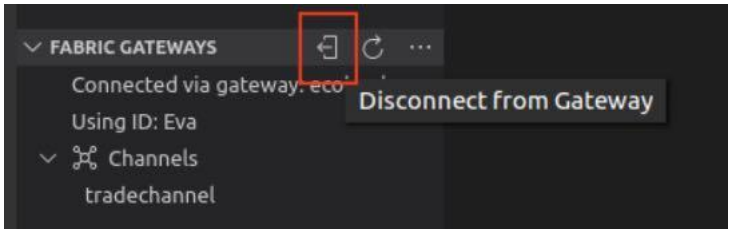




__ **44.** Verify that Eva can see the **tradechannel**



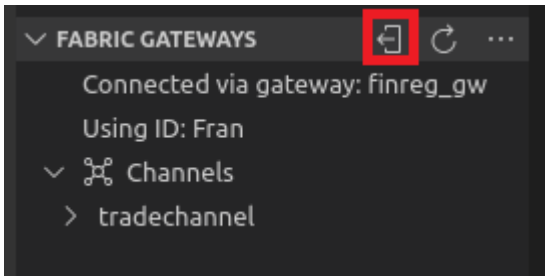
__ **45.** Hover over the Fabric Gateways panel and **click** the **Disconnect** icon to disconnect from the gateway



__ **46.** Using steps *39 – 45* as a guide, add a **digibank** gateway called **digibank_gw** and connect with the identity **David.** Verify that David can see the tradechannel channel.

__ **47.** Using steps *39 – 45* as a guide, add a **finreg** gateway called **finreg_gw** and connect with the identity **Fran**. Verify that Fran can see the tradechannel channel.

__ **48.** As the final step in this section, hover over the Fabric Gateways panel and **click** the **Disconnect** icon to disconnect from the **FinReg** gateway.



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 a smart contract is deployed.

2 Deploy the first version of the Smart Contract

2.1 Introduction

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 organisations' 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.

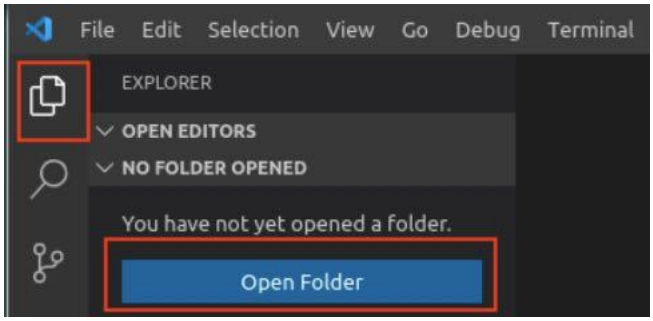
The second is **Private Data collection definition files**. These are policy-based definitions that describe which organisation(s) belong to a Private Data collection – it could be one, two or more organisations (as policy dictates), that govern who can access private data in a given collection, on a channel. It is commonplace for the definition file to contains one or more collections (each has a name and the participating organisation(s) 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 the smart contract, you will be supplying the definition of the Private Data Collections.

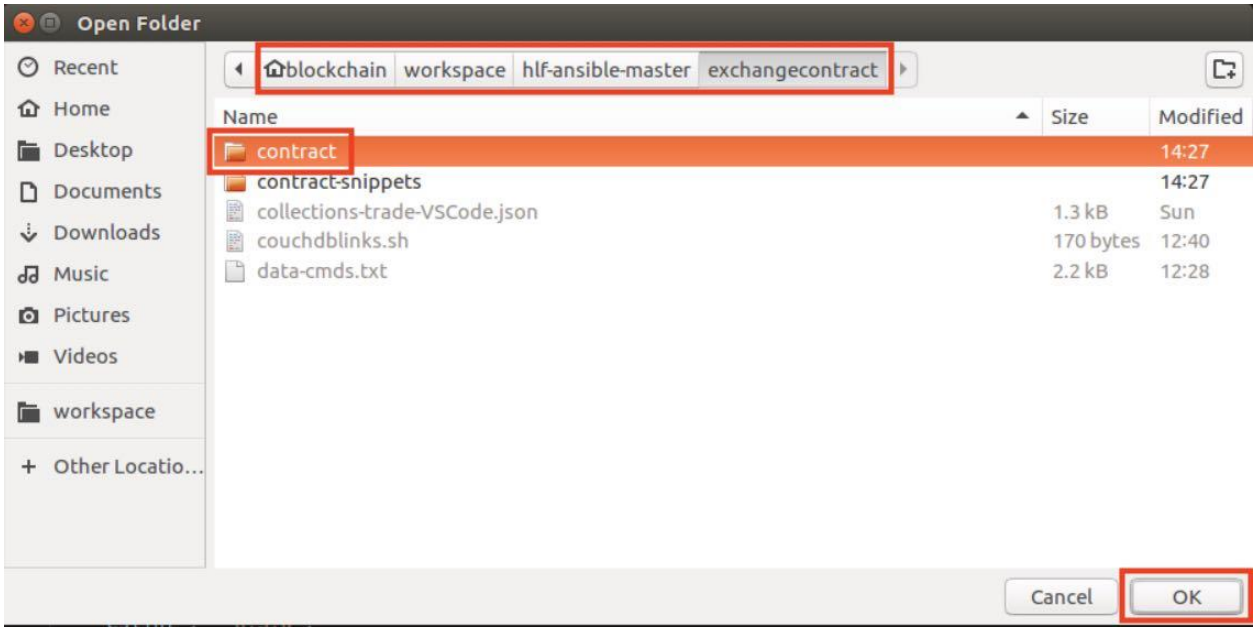
The definition file for this lab is shown below:

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**

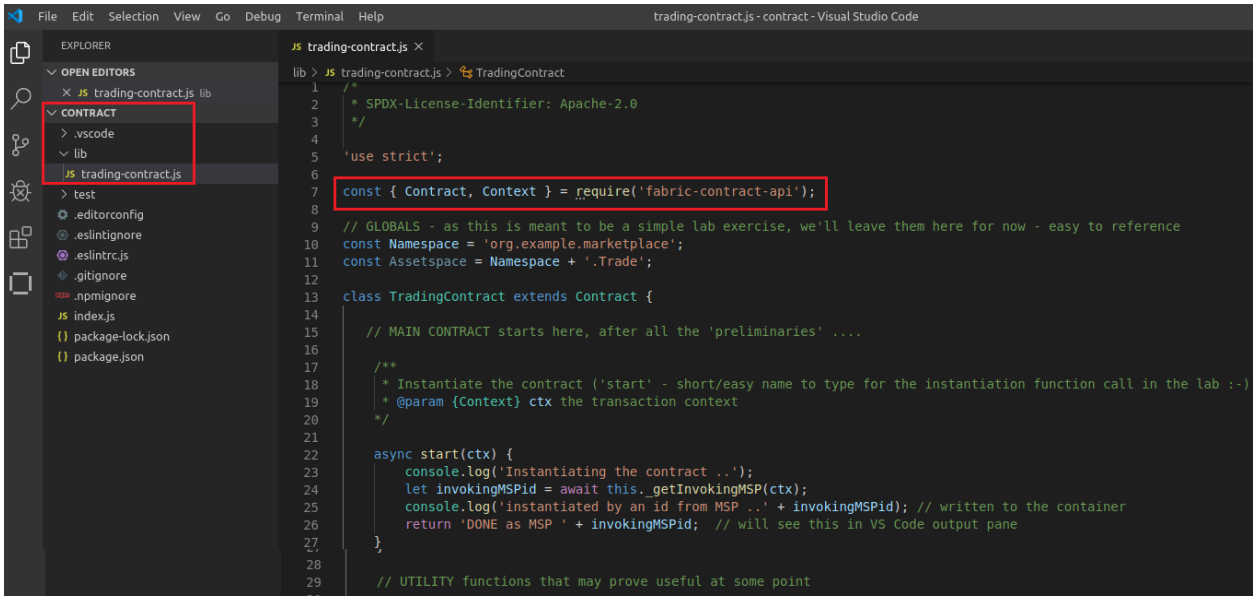


Navigate to the location:
"Home" > workspace > hlf-ansible-master > exchangecontract
Click to highlight the **contract** folder and click **OK**

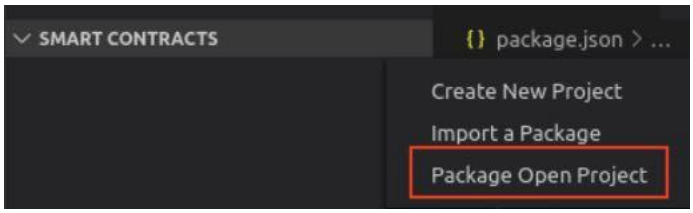


__ **50.** VS Code will display the **Contract folder** tree. Expand the twisty for the **lib** folder and double-click **trading-contract.js** to view it.

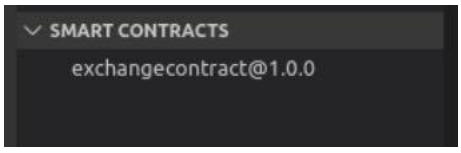
.



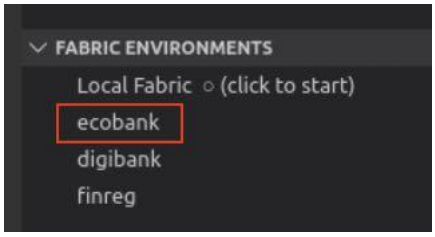
__ **51.** Click the icon  to return to the IBM Blockchain Platform extension, hover over the **Smart Contracts** panel, click the Ellipses icon to select “More actions” and then select **Package Open Project**



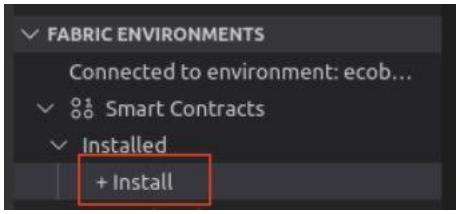
This will cause the smart contract you just imported to be packaged into a CDS file that can be installed on a peer. The package will then appear in the Smart Contracts list.



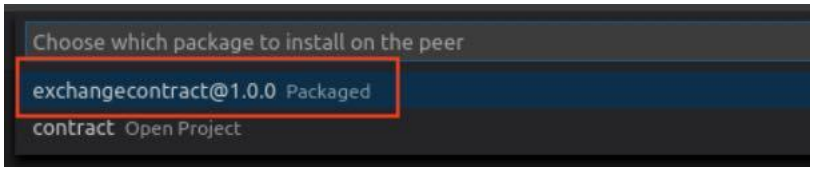
__ **52.** On the Fabric Environments panel click on **ecobank** to connect to the EcoBank environment:

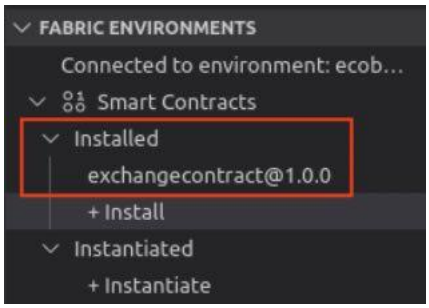


__ **53.** Click **+Install** to install a contract on the peer



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

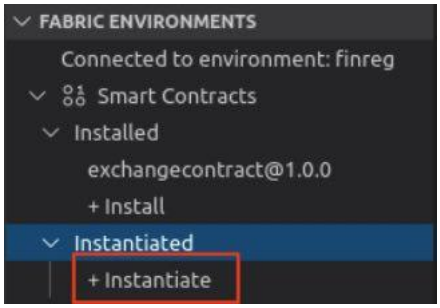


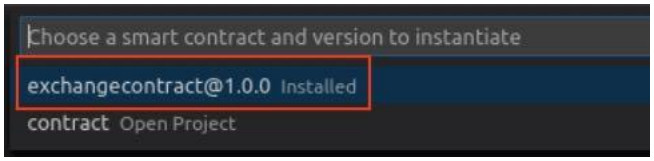


__ **54.** Disconnect from the EcoBank Fabric Environment, and using the previous two steps as a guide, install the same smart contract package in the **digibank** environment.

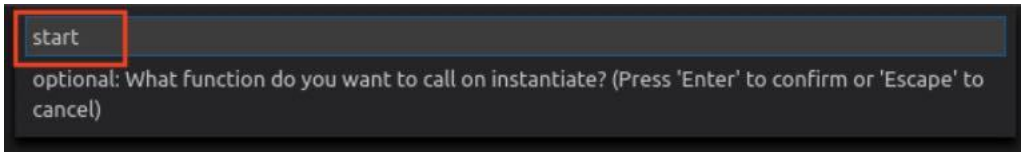
__ **55.** Disconnect from the DigiBank Fabric Environment and using the previous two steps mentioned as a guide, install the same smart contract in the **finreg** environment.

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

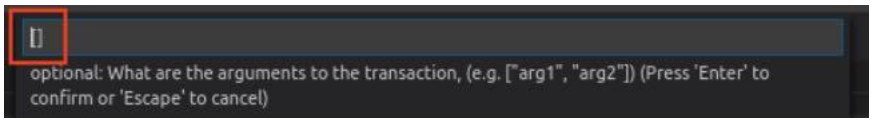




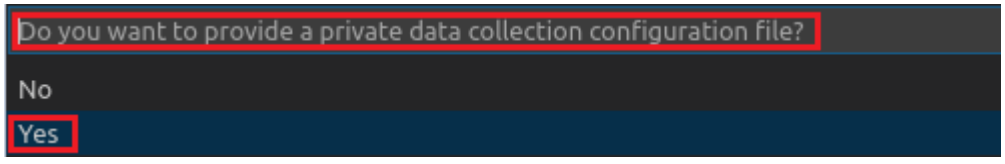
__ **57.** Specify **start** as the function to be used with instantiate



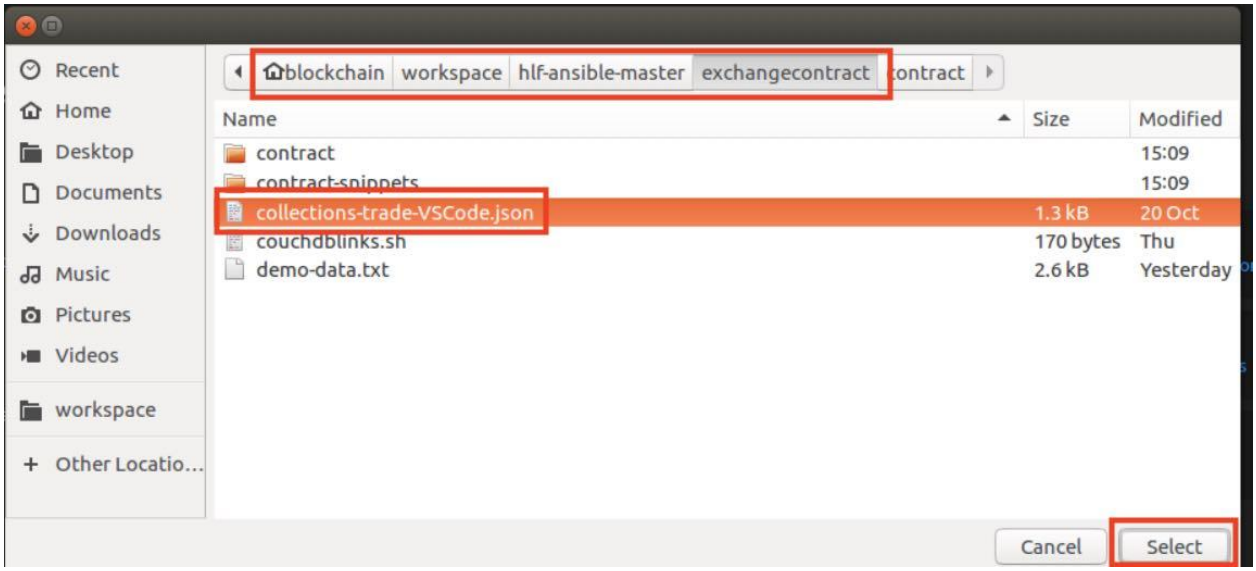
__ **58.** There are no parameters to pass to the start function – just press **enter** with the default empty array **[]**



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

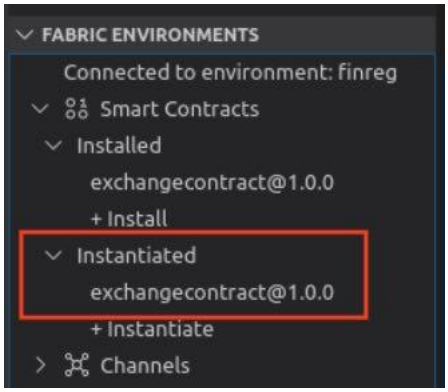


__ **60.** Click **Browse** and navigate to the folder:
"Home" > workspace > hlf-ansible-master > exchangecontract
and select the file **collections-trade-VSCode.json**.



This will take a minute or so to complete.

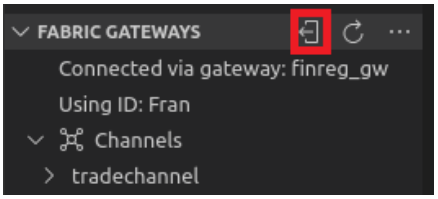
When the contract is instantiated it will show as follows:



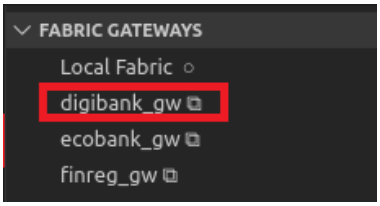
The smart contract has been instantiated on the **tradechannel** channel.

The dockerized chaincode container has been started for **FinReg** following instantiation, but we also need to bring them up as operational containers, in the other organisations in our multi-organisational network before proceeding

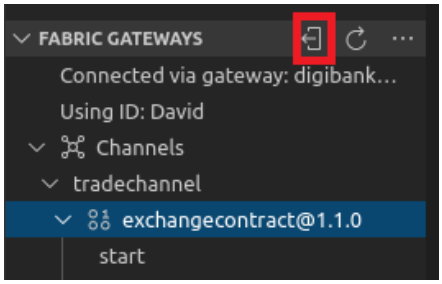
__ **61.** **Disconnect** from **FinReg's** gateway by hovering over Fabric Gateways and clicking the icon



__ **62.** Connect to **DigiBank**'s gateway under Fabric Gateways as identity **David**



__ **63.** Expand the channel tradechannel, then expand the twisty for **exchangecontract@0.0.1 –** the 'spinning' icon, indicates its bring up the chaincode container in the background, for DigiBank's environment. It will take 10-12 seconds do do and the result is an expanded list of transactions when completed. Once done, **disconnect** from DigiBank's gateway by hovering over the Fabric Gateways title bar.



__ **64.** Using the previous two steps as a guide, connect to **Ecobank's** gateway, and again expand the twisty for **exchangecontract@0.0.1** to bring up the container and confirm you see the list of transactions.

__ **65.** Finally, **disconnect** from EcoBank's gateway, as you had done for DigiBank above.

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)
- Operationally started the chaincode containers on the two remaining Organisations.
- Instantiated the Smart Contract with the definition of the Private Data Collections

3 Private Data for the Offer and Accept Transactions

3.1 Introduction

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

Part 3 of the lab is to review and execute these transactions so that we can create private data. After an initial **advertize** transaction is completed, you invoke an **offer/accept** set of as 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 request data that results in private data records for each organisation. Worth noting that the functions will also update the advertised commodity contract asset itself on the main channel ledger with 'non-private' data eg. 'status' or 'offer date'. Later, you will follow on with a 'cross-verification' set of transactions, involving different smart contract functions - this will be covered in Part 4. (For a complete list of the function names, their objectives and a brief description of each, please see the Appendices).

Private data input to the **offer** and **accept** transactions is passed as 'transient data' to a client - 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 nominated private data collection.

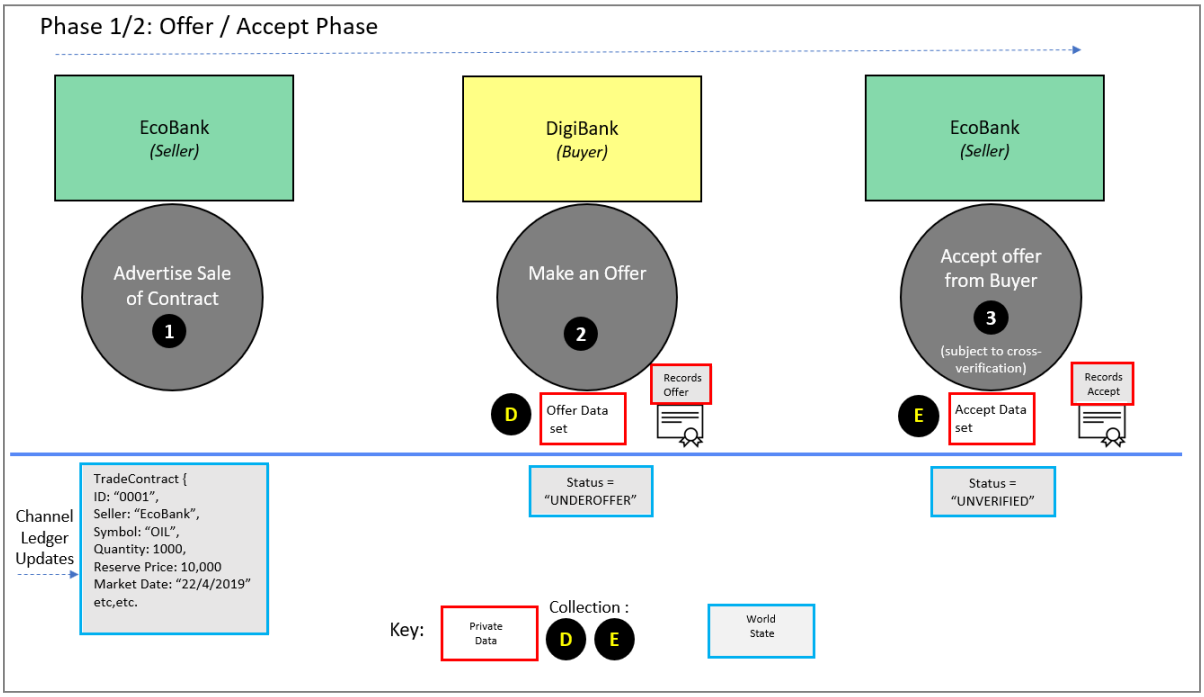
Note: It is good and common practise to 'salt' this transient data with a random element, especially if it contains more predictable private data so that a matching hash cannot realistically be found via brute force. See the appendix 'Access Control considerations for Private data"' for more information.

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

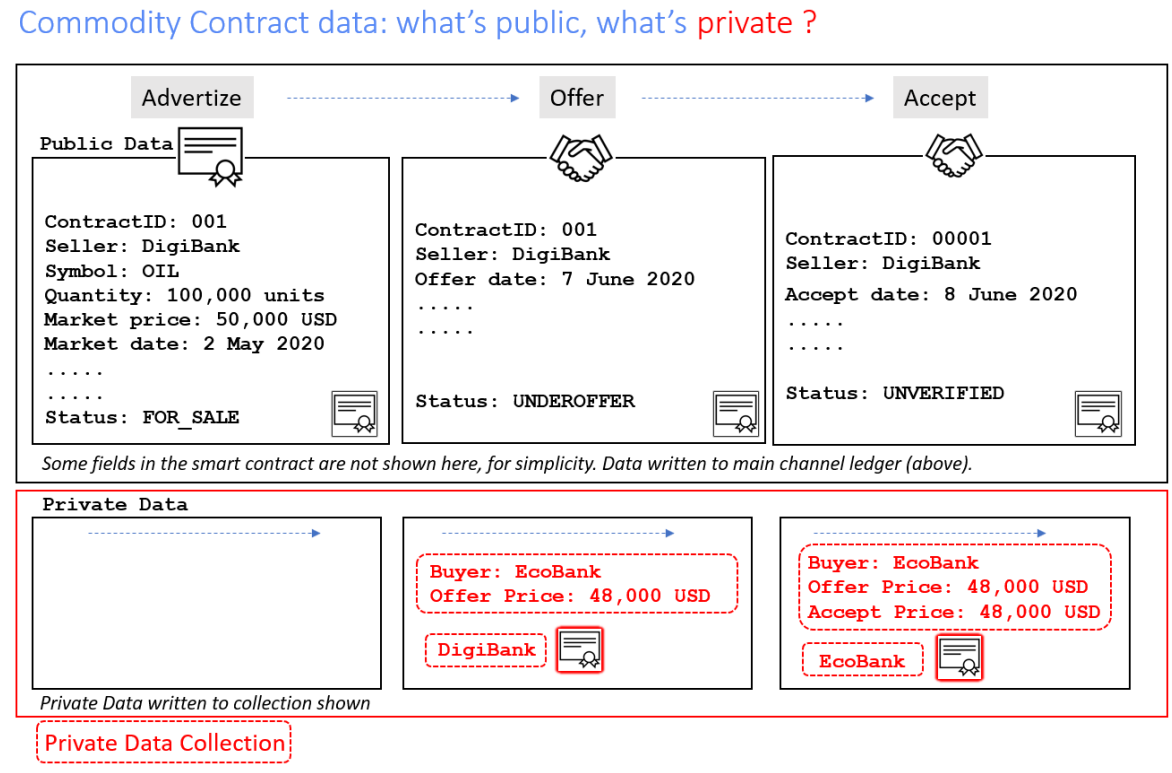
1. Seller **EcoBank** *advertises* the nominal sale price / info on the marketplace and it is recorded on the blockchain - this is written to the channel-wide ledger and all would-be buyers get notified (via applications, out-of-band).
2. Buyer **DigiBank**, makes an *offer* to buy the contract – it 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 (that's 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:



During the lab exercise, you will see both private data and channel ledger data generated, as transactions get invoked. From a labs execution standpoint, its useful to see what happens 'under the covers' in CouchDB – we can access each organization's CouchDB database tables, by using the Firefox browser (one tab for each data CouchDB instance) to open database views. So let's first open these CouchDB views.

3.2 Launch CouchDB Views and Copy/Paste Helper file

Prior to invoking transactions, its useful to examine the 'before and after' states of database tables created, as a result of private data being added, and indeed status/attribute changes in the **tradechannel** ledger world state.

In addition, certain commands sequences can be lengthy and thus are more easily and efficiently sourced from a text file. We'll open these items as a preparatory step to completing the hands-on exercises.

__ **66.** 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$
```

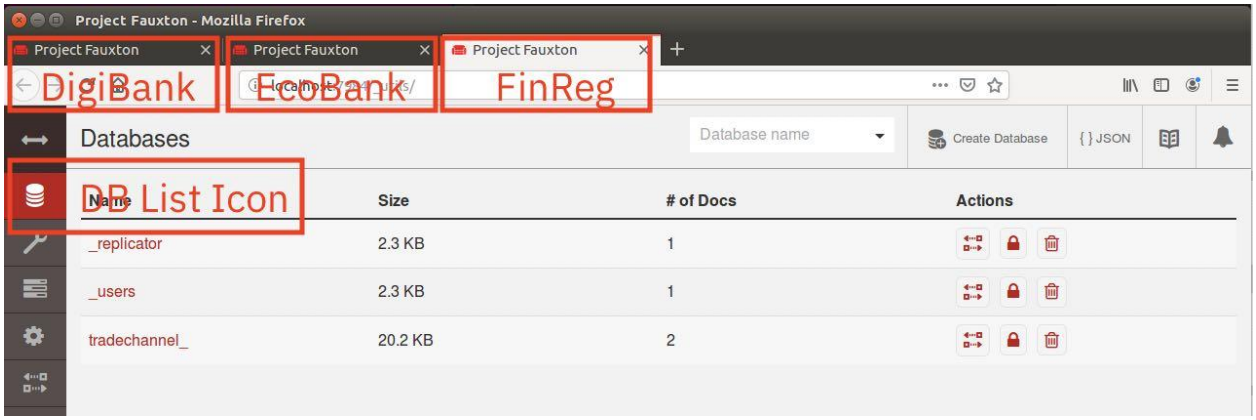
When the script is complete you should see a Firefox browser window with three tabs displayed. Note that the script may take a couple of minutes to complete, and there will be a pause between tabs opening.

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

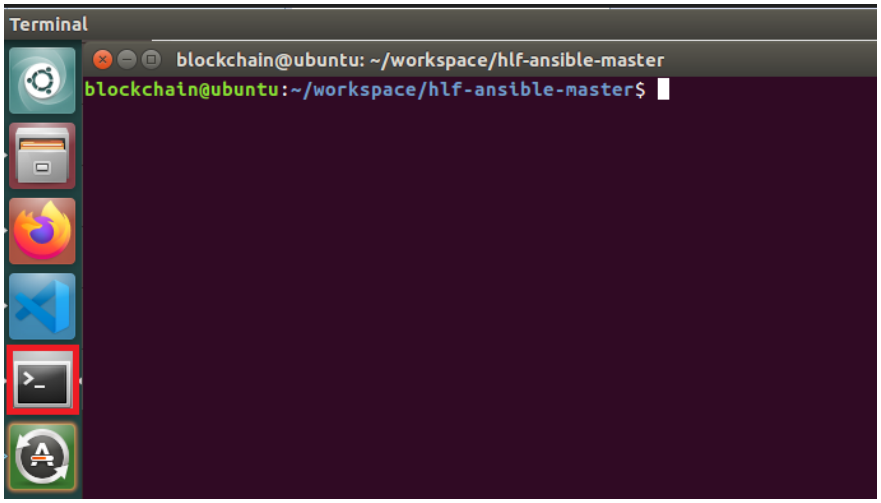
Each tab represents the view for a different organization:

- 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 CoucbDB Web interface **would be blocked** for security, but it is useful during development and testing.



__ **67.** As mentioned, we've also provided you a file called **demo-data.txt** from which you can copy and paste – it's located under the **workspace/hlf-ansible-master/exchangecontract** folder. Click once on the **Terminal** icon to launch a terminal window.



__ **68.** Change directory to the subdirectory **exchangecontract**

```
cd exchangecontract
```

__ **69.** Issue the following command to open the text file in VS Code in the **exchangecontract** sub-directory:

```
code demo-data.txt
```

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

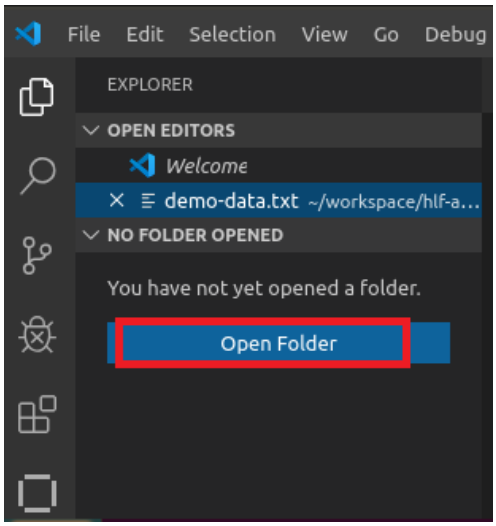
You can copy and paste from this text file in the background in the steps that follow.

3.3 Review and Perform the 'advertize' transaction

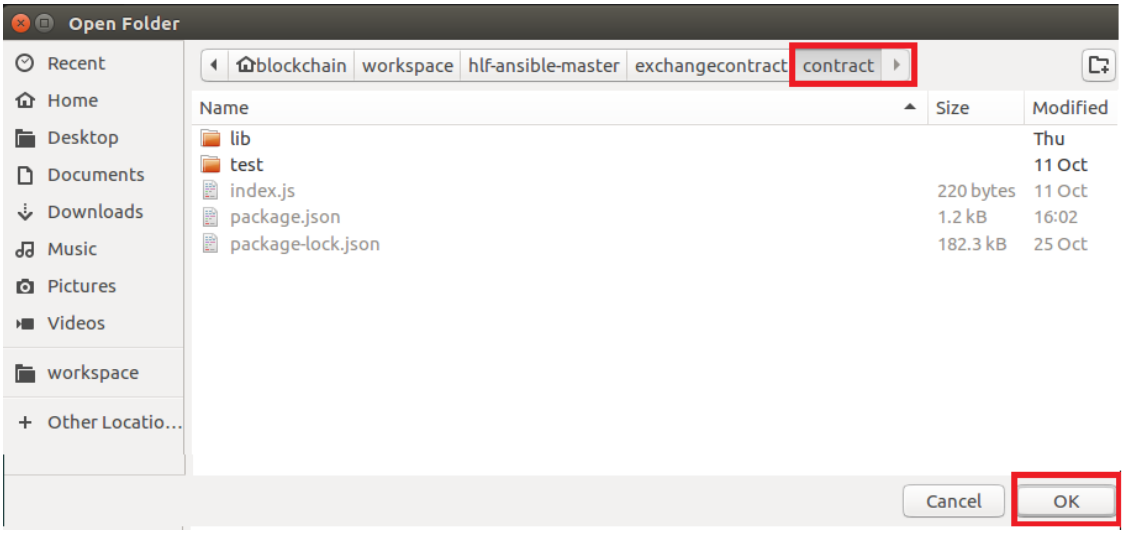
Let's briefly explore the transaction functions provided in the instantiated contract, starting with **advertize**.

__ **70.** IN VS Code, click on the **Explorer** icon

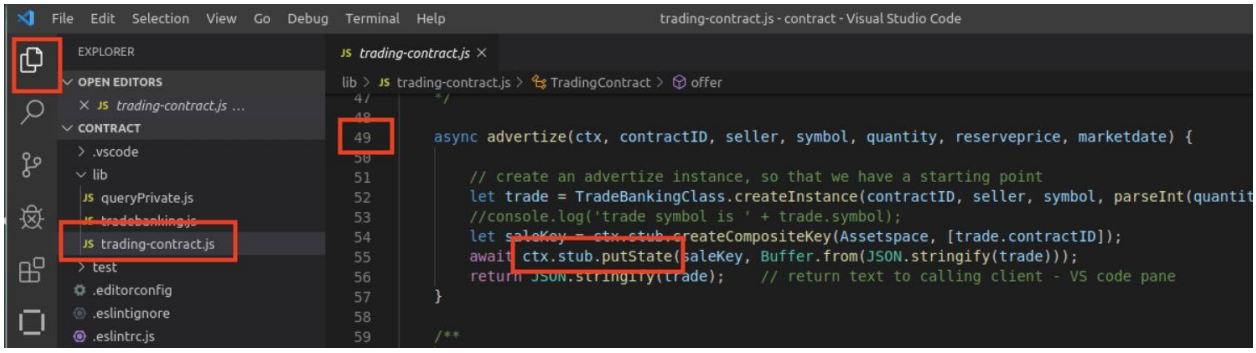
__ **71.** Click on the **Open** Folder button to open the existing smart contract



__ **72.** Navigate to the **contract** folder under **exchangecontract** and click **OK**

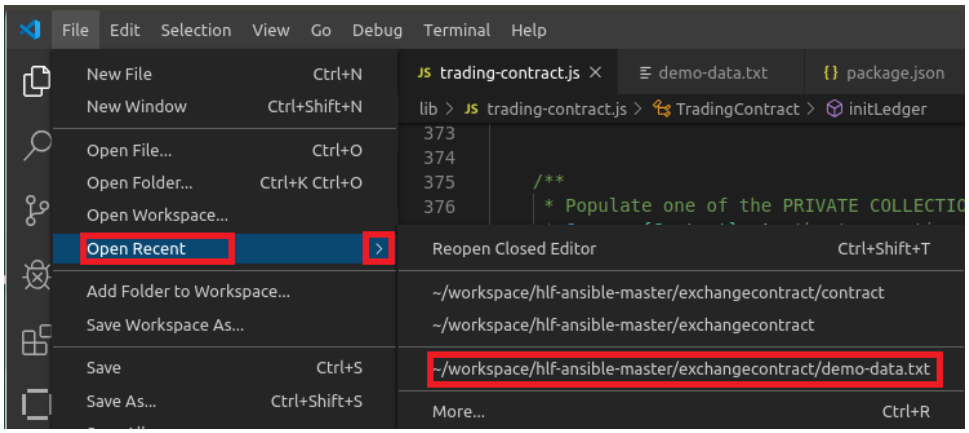


__ **73.** Click on the file **trading-contract.js** under the **lib** directory in Explorer and locate the function beginning with **async advertize** function at **line 49**.



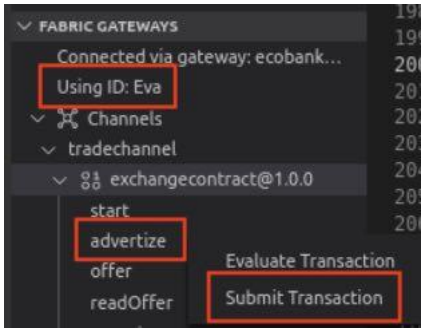
The **advertize** transaction creates a saleable Commodity contract (identified by a 'contract ID') on the channel ledger. For information, it uses the Fabric **putState API** to write non-private data to the ledger (no private data created just yet). This transaction function is called by someone from EcoBank (the seller).

__ **74.** To make our 'helper' commands file available in VS Code, simply do a **File....Open Recent** and select the file **demo-data.txt** to open



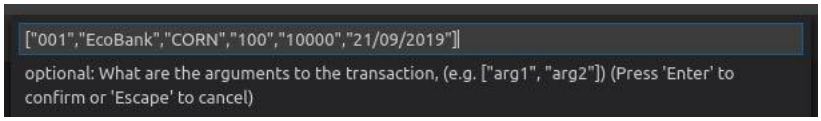
__ **75.** Next, let's invoke the transaction on the instantiated contract - switch back to the IBM Blockchain Platform VS Code extension using the icon, and in the Fabric Gateways panel click **ecobank_gw** and connect to it using the identity **Eva**

__ **76.** Expand Channels and tradechannel - then expand the exchangecontract@1.0.0 twisty, then right click on the **advertize** transaction and click **Submit transaction**.



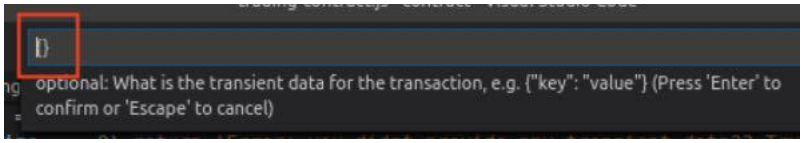
__ **77.** Next, enter the following parameters for arguments when prompted – overwrite the '[]' brackets offered in the popup with the parameter list copied from **demo-data.txt**

["001","EcoBank","CORN","100","10000","21/09/2020"]



As you may recall from the advertize function in the contract, the parameters represent the contract ID, seller, symbol, quantity, reserve price and market date respectively.

__ **78.** 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

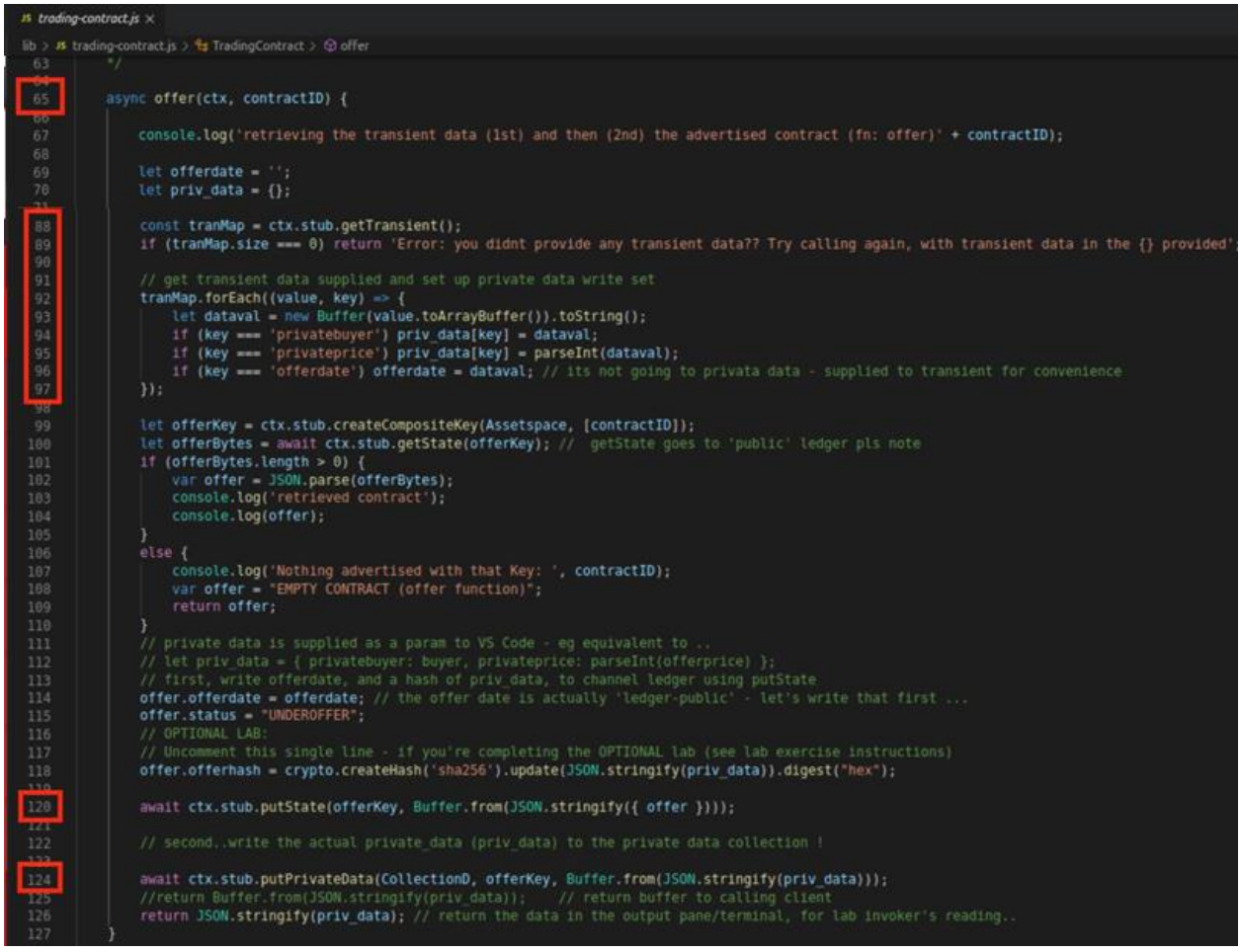
3.4 Review and Perform the 'offer' transaction

__ **79.** IN VS Code, scroll to the **offer** transaction at **line 65** in trading-contract.js**.**

The offer transaction takes a commodity contract ID as a parameter and reads in 'transient data' (during invocation) as private data – that data is supplied when invoked from the VS Code extension or a client app. FYI, transient data is passed as binary data and then decoded in the function starting with the Fabric API **getTransient** method.

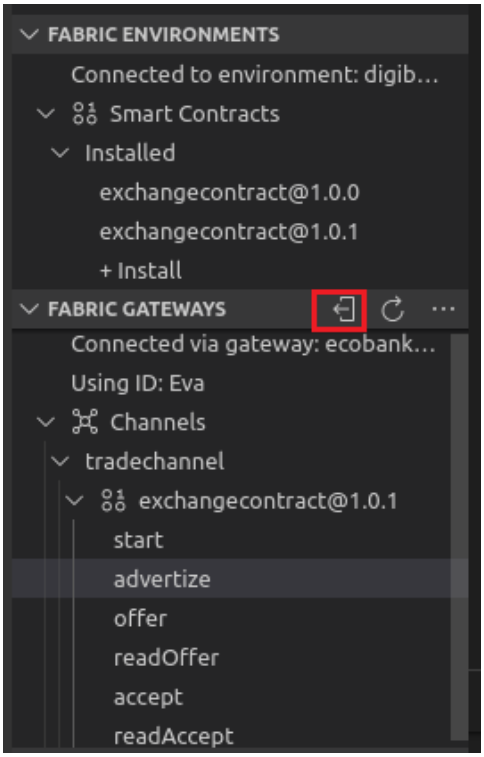
Lines 88-97 show the private data elements being committed to DigiBank's private collection using the Fabric API method **PutPrivateData** in **line 124**

Line 120 causes the 'non-private' elements to be written to the main ledger using **putState** inside the same offer transaction – i.e. data like 'Status' or 'Offer Date' is intentionally visible to others on the channel.



Now let's invoke an **offer** transaction for the advertised commodity contract (with Contract ID `"001"`) – after, we will verify the private data was added using the **readAccept** transaction.

__ **80.** The **offer** transaction needs to be done as a Digibank trader – first, we must disconnect from the EcoBank gateway – **click** on the **disconnect** icon

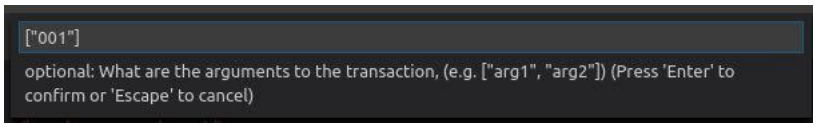


__ **81.** Next, click on the DigiBank gateway (**digibank_gw**) – when prompted, select **David** as the identity to connect with

__ **82.** Right-click on the **offer** transaction and click **Submit Transaction**

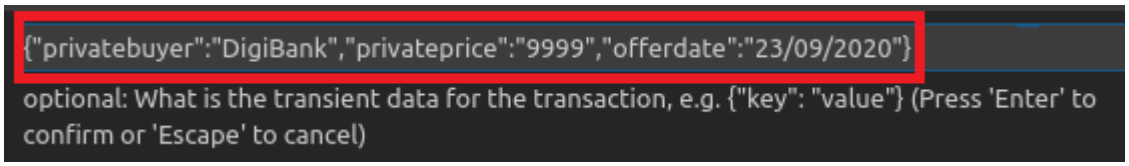
__ **83.** When prompted for arguments, enter the following parameter (overwriting any existing '[]' data) – and press **enter**

["001"]



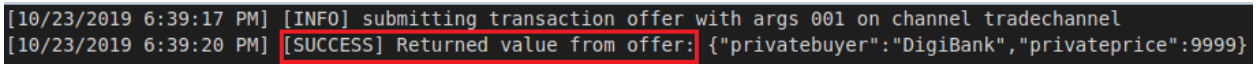
__ **84.** Next, you will be prompted to provide 'transient data'. Our private offer is confidential and so we want to use this 'transient' field to supply this information so that it is not captured on the ledger. Enter the following parameter list - or paste from 'demo-data.txt' commands file - the parameter list, overwriting the '{}' text presented. **Notice that the data is wrapped in curly brackets this time – this is required when passing transient data!**

{"privatebuyer":"DigiBank","privateprice":"9999","offerdate":"23/09/2020"}



__ **85.** Check the output pane at the bottom for a SUCCESS message confirming that the offer transaction was submitted successfully.

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



__ **86.** Review the additional helper functions **readOffer** (line 134) in the contract:
The read-only transaction **readOffer** is a helper function, to enable you to verify what was written to private data earlier by the offer transaction.

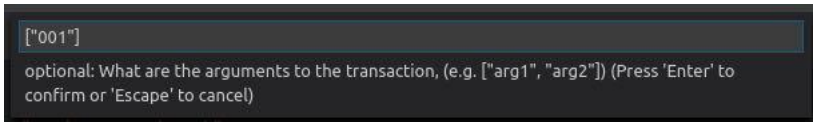
```
130     * Read the offer - can be done by someone from the buyer's bank - this function will read private data from the collection
131     * @param {Context} ctx the transaction context
132     * @param {String} contractID contractID
133     */
134    async readOffer(ctx, contractID) {
135
136        console.log('retrieving the offer from Private data (fn: readOffer)' + contractID);
137
138        let readKey = ctx.stub.createCompositeKey(Assetspace, [contractID]);
139        let pdBytes = await ctx.stub.getPrivateData(CollectionD, readKey);
```

Next, we will call the **readOffer** transaction to verify our private data – still connected to the DigiBank Gateway as **David**.

__ **87.** Right-click on transaction **readOffer** and click **Evaluate Transaction**

__ **88.** When prompted for parameters, enter the following:

["001"]



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 001 on channel tradechannel
[10/31/2019 9:43:04 AM] [SUCCESS] Returned value from readOffer: {"privatebuyer":"DigiBank","privateprice":9999}
```

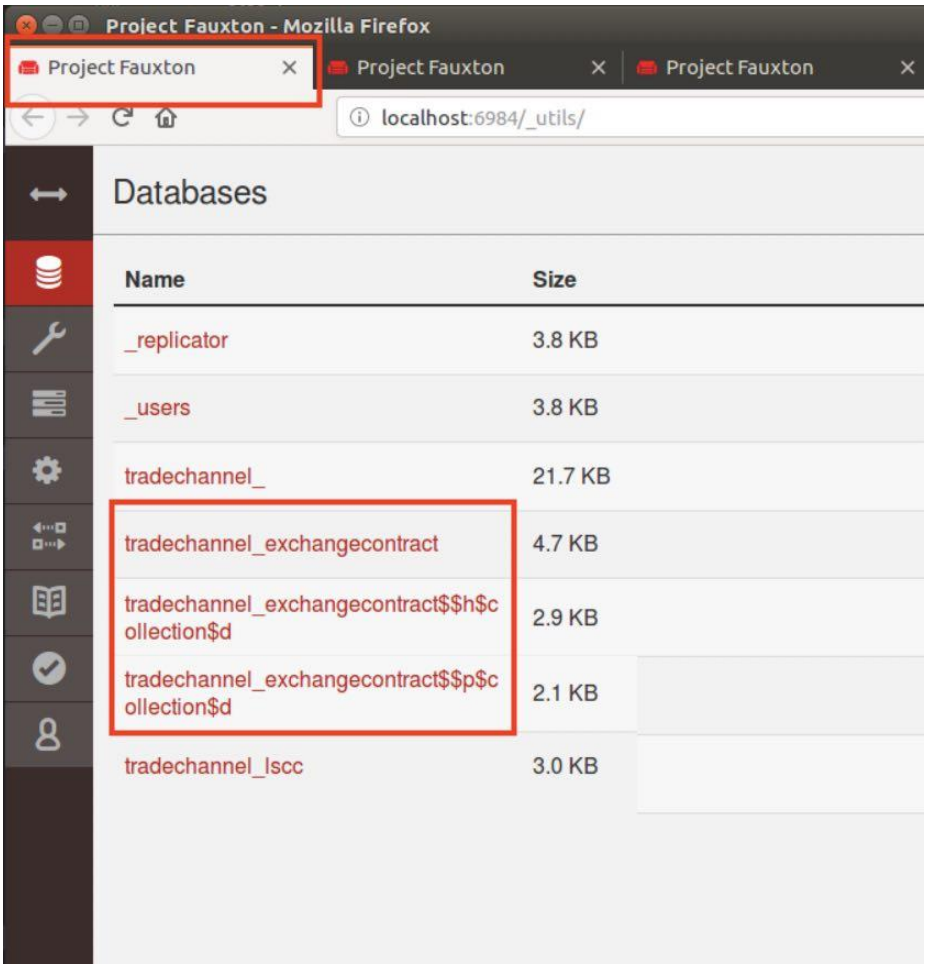
Currently, only David from DigiBank can view this private information.

From a 'learning consolidation' perspective, we shall next review the 'private data artifacts' created in CouchDB, it provides some context. You recall that earlier, we launched 3 Firefox tabbed sessions each containing a CouchDB view and each listing a set of additional databases - you can click through each to explore the individual records we've created, following the **advertize/offer** transactions that create private and channel-wide data.

__ **89.** Switch to the **firefox** browser window in your VM

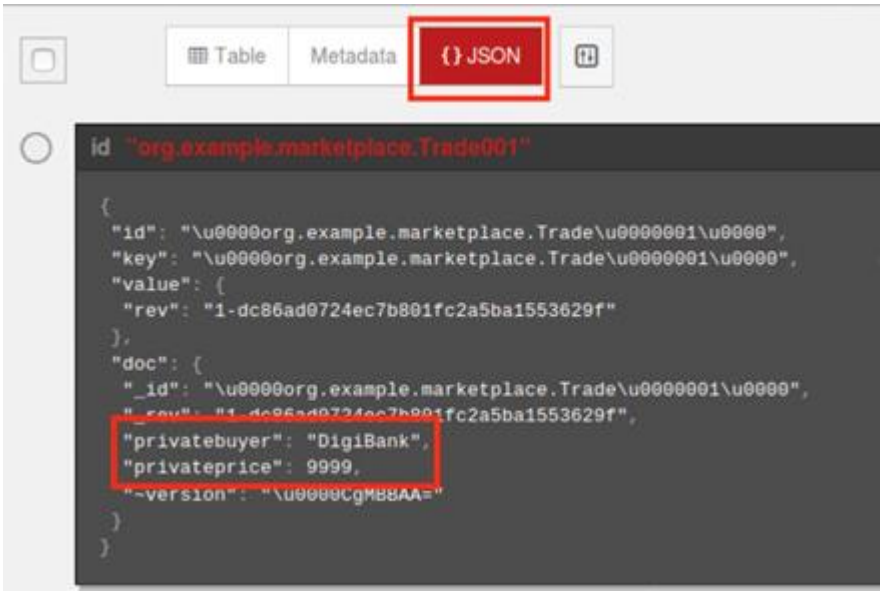
__ **90.** Click on the DigiBank CouchDB tab in Firefox – this is the URL running on **port 6984** and is the **first tab**. Refresh the page in the browser to see the updated information.

__ **91.** Review the databases created following the 'offer' transaction unit of work:
tradechannel_exchangecontract is the world state database for the exchangecontract on the tradechannel
tradechannel_exchangecontract$$h$collection$d is the private data collection **hashstore** for 'CollectionD'
tradechannel_exchangecontract$$p$collection$d is the private data collection for 'CollectionD' (which is only available in DigiBank's CouchDB)

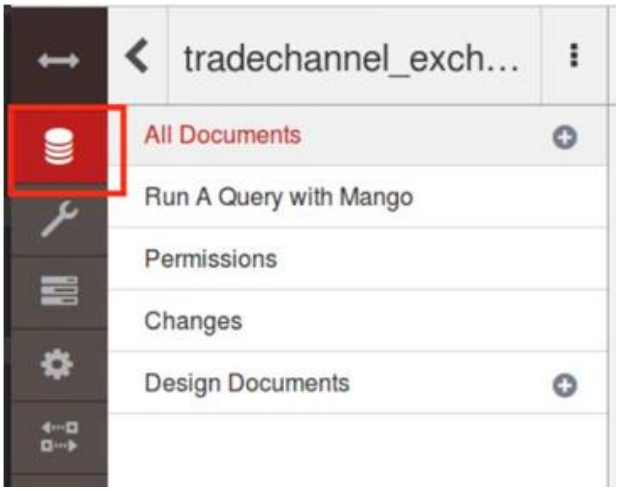


__ **92.** Click on the private collection tradechannel_exchangecontract$$p$collection$d

__ **93.** Click on the **{} JSON** view button (it's the JSON button alongside 'Metadata' as shown) and you can see a JSON record of the private data that was written for Contract ID "001".

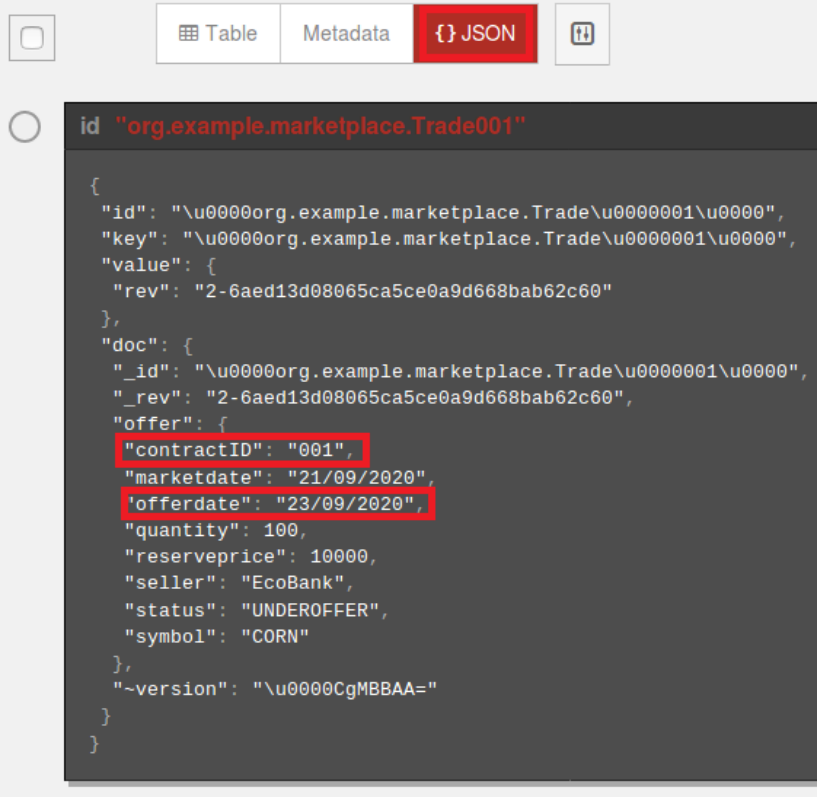


When you have reviewed the data, click the **All DBs** icon to return.



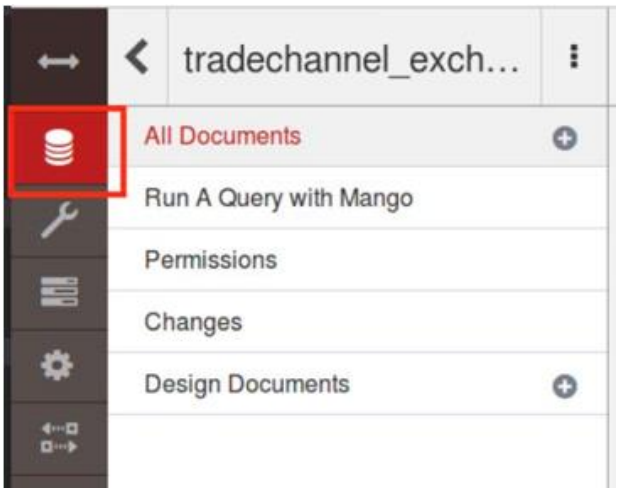
__ **94.** Back in the main database list – click on the world state database called **tradechannel_exchangecontract**

__ **95.** Click on the **{} JSON** view once again and you can see the record that contains `"contractID": "001"`



Notice here, that an offer date **is** shown, but the actual private offer data is not.

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



__ **96.** Still using the **first tab** in firefox for the **DigiBank** CouchDB open the collection hash database **tradechannel_exchangecontract$$h$collection$d** and verify that the key and value is unreadable.

Using the **second tab** for the **EcoBank** CouchDB, look at the database list and notice that EcoBank does not have the private database (named below) (**tradechannel_exchangecontract$$p$collection$d**)

This is expected, because the data is confidential to DigiBank.

3.5 Review and Perform the 'accept' transaction

Having verified the data written to the databases in CouchDB in the previous section, we can proceed to perform the **accept** transaction. It is performed by Eva at EcoBank – she accepts the offer from DigiBank. Once again, we'll review the function, then after, invoke the accept.

__ **97.** In **trading-contract.js** in VS Code, scroll to the implementation of the **accept** transaction function at **Line 157.** The accept transaction has similar private data logic as the offer function, i.e. decoding the transient data **(lines 179-190)** then writing Private Data to EcoBank's collection at **line 212** and like the offer transaction, updating non-private data on the channel ledger for tradechannel at **line 208**.

The **accept** transaction writes a different private data set – i.e. an accept price in addition to the offer data set that was agreed with DigiBank. It again uses the **putPrivateData** Fabric API method to write private data and **putState** to update the channel ledger with non-private data as before.

The read-only transaction **readAccept** is a helper function, similar to readOffer earlier – in that it enables the seller identity, to verify what was written to the private collection, following the earlier 'accept' invocation.

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

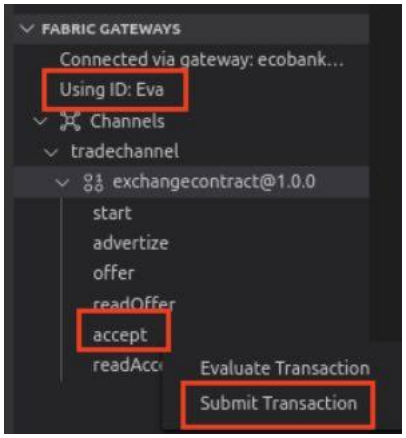
Now let's perform the actions of the **accept** transaction function – after, we will verify the private data was added using the **readAccept** transaction.

__ **98.** Return to the VS Code editor and in the Fabric Gateways panel, click the Disconnect icon to **disconnect** from DigiBank's gateway.

__ **99.** In the Fabric Gateways panel connect to **ecobank_gw** and choose **Eva** as the identity

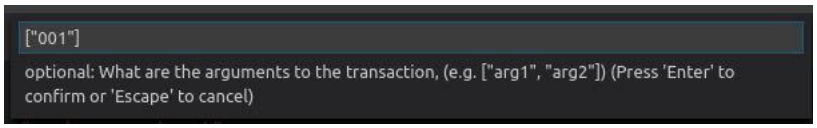
__ **100.** Expand the transaction list as before, and **right click** on the **accept** transaction and click **Submit Transaction.**

This will create some private data in EcoBank's private data collection.



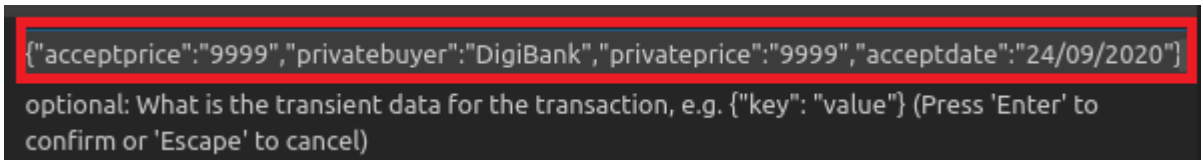
When prompted to enter arguments enter the following Contract ID parameter:

```
["001"]
```



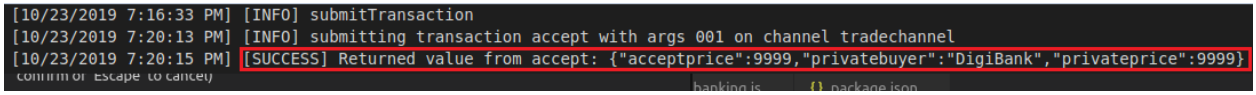
__ **101.** Enter or paste in from the file 'demo-data.txt' (still open in VS Code) the following transient data for this transaction, replacing the '{}' provided with the following one-liner and hit **enter**.

{"acceptprice":"9999","privatebuyer":"DigiBank","privateprice":"9999","acceptdate":"24/09/2020"}



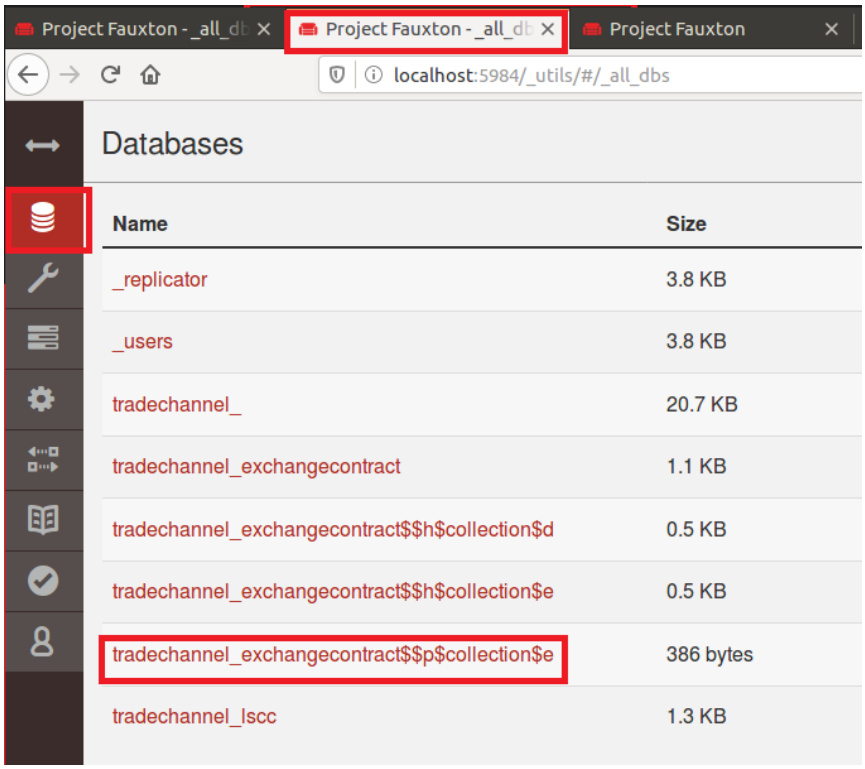
The transaction is now submitted.

__ **102.** Review the output pane at the bottom for results – you should see a SUCCESS message like this one containing the private data being written:



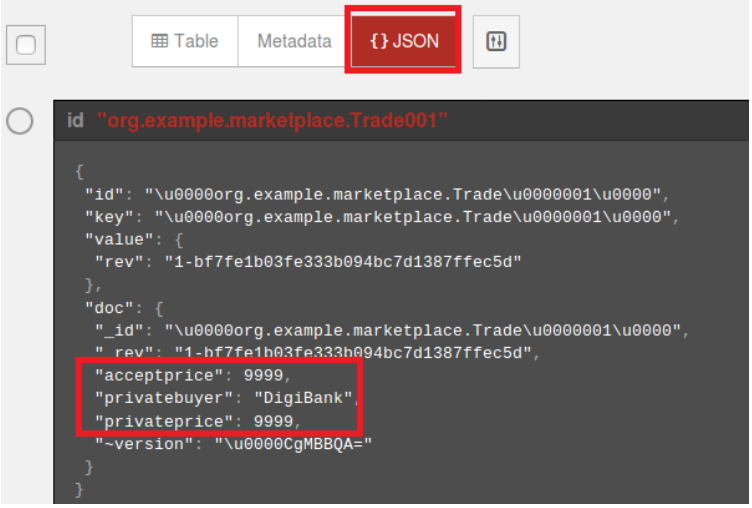
Notice that the 'acceptdate' value is not part of this SUCCESS message. This is because 'acceptdate' is a 'non-private' field written to the channel ledger – the output only reflects the private data written to the collection.

__ **103.** Return to Firefox to review activity in CouchDB on the **second tab** for the **EcoBank CouchDB** running on **Port 5984** - Click the **All DBs** icon (on left) – click on the database for the private collection CollectionE called **tradechannel_exchangecontract$$p$collection$e**

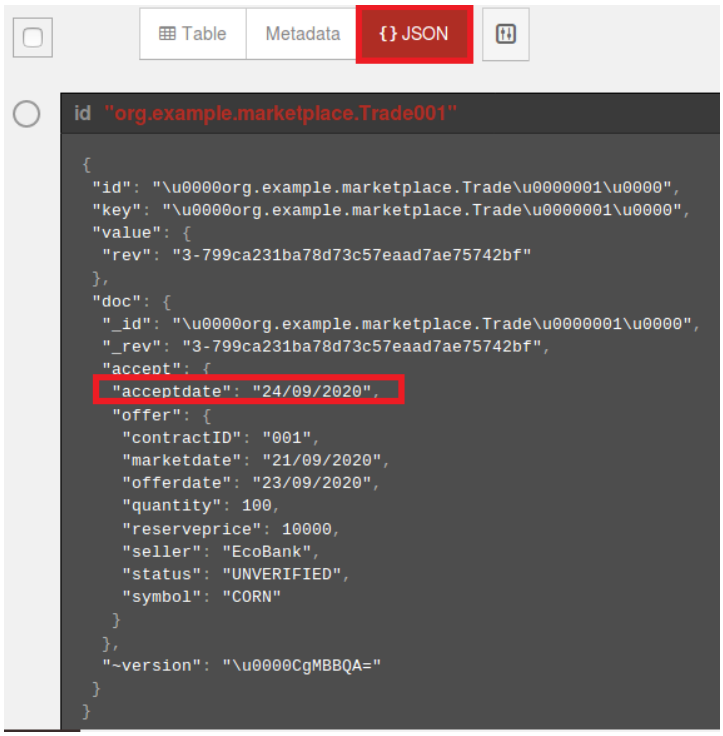


__ **104.** Click on the **{} JSON** format icon alongside

Notice that the record for trade contract 001 has the private data for the **accept** transaction.



__ **105.** As you had done before, click on the 'All Databases' icon, and this time, click on the main channel ledger database called '**tradechannel_exchangecontract'** and open the 'Trade001' record as JSON (adjacent to 'Metadata' as shown), to see that the accept date was written to the world state.



We are now going to attempt to read EcoBank's private data with a DigiBank ID - which should be denied - proving that the data is confidential to EcoBank.

__ **106.** Return to VS Code and in the Fabric Gateways panel click the **Disconnect** icon to disconnect from the current gateway.

__ **107.** Connect to **digibank_gw** with the identity **David.**

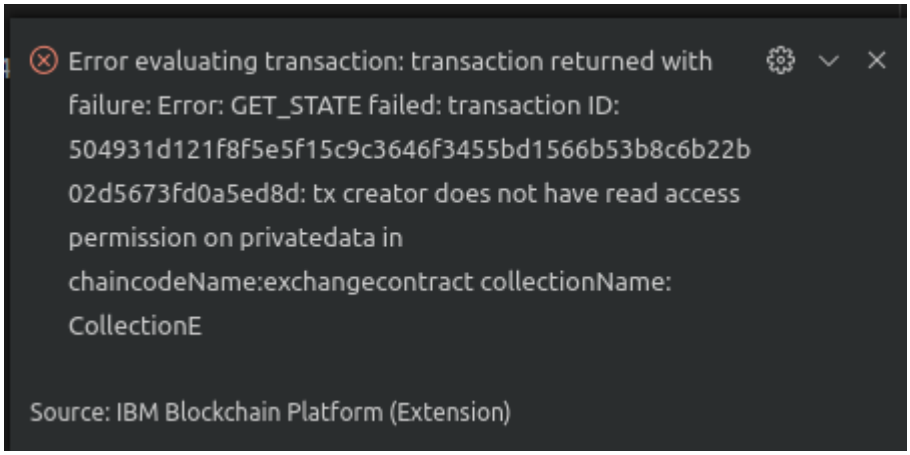
__ **108.** Right-click on the **readAccept** transaction and click **Evaluate Transaction.** When prompted enter the following in the parameters

```
["001"]
```

__ **109.** There is no transient data for this function, just press **enter** leaving the empty curly braces **{}**

The evaluate should return an error, see bottom right in VS Code.

__ **110.** 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.



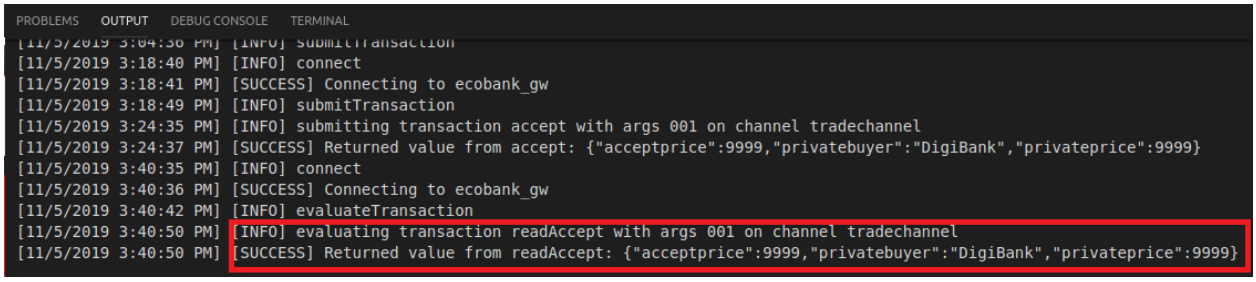
__ **111.** Now **disconnect** from the DigiBank Gateway in the VS Code extension.

__ **112.** **Connect** to EcoBank's Gateway as **Eva**

__ **113.** Highlight the transaction **readAccept** and do a right-click ... 'Evaluate Transaction' -and at the prompt, replace the existing '[]' string entirely with the following:

```
["001"]
```

__ **114.** Review in the output pane that you can read the accept private data as Eva.



__ **115.** As a final step, **disconnect** from **EcoBank's** gateway

Review

- In this part of the Lab you have reviewed the private data transaction functions already provided in the 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 offer/accept functions to create private data in those collections.
- You have seen through CouchDB views, that the transactions also create a private data hash store, containing the public hash of the corresponding private data collection.
- You have seen that the private data created by an organisation is indeed private – David from DigiBank could not see the EcoBank private data collection.

4 Work with the Cross Verify Functions

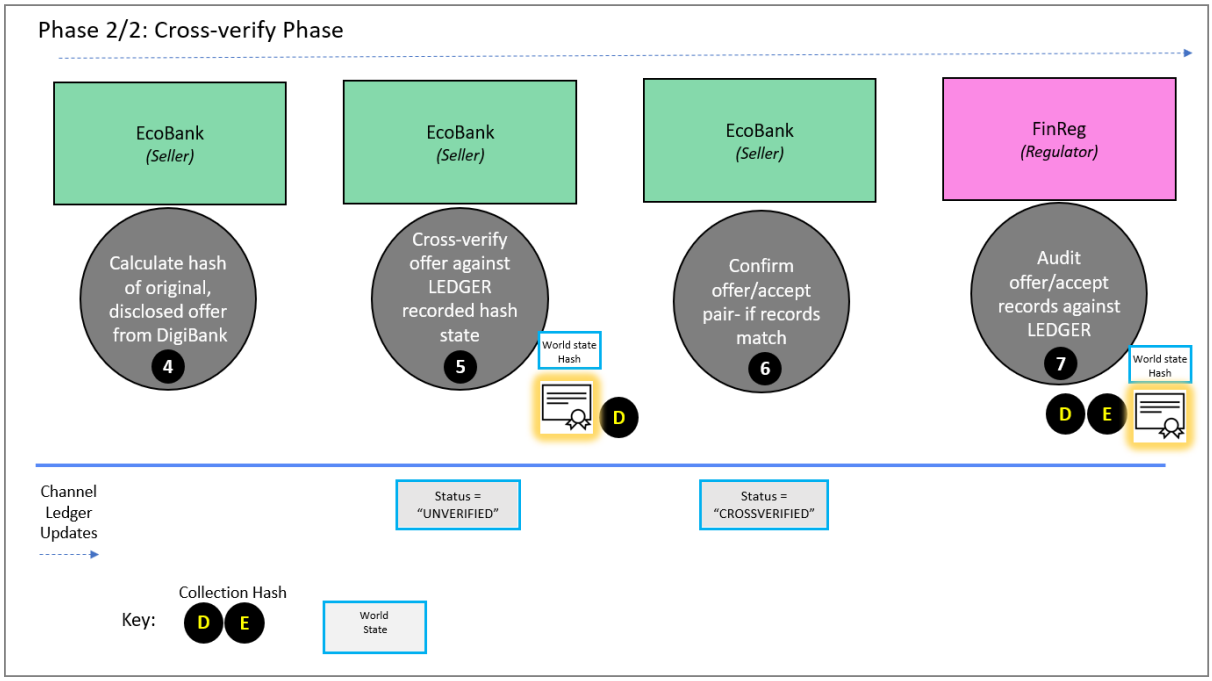
4.1 Introduction

In this section we will upgrade our smart contract to include cross-verification functionality, which will enable the smart contract to verify a calculated hash of source data (done by a verifier) against the private data hash record and also, update the channel ledger to confirm it as a 'CROSSVERIFIED' record.
An identity from a proving organisation (for example, the seller) will invoke this cross-verify function which checks whether the two hashes match, returning a MATCH / NO MATCH response depending on the comparison.
It's important to note that the verifying party only has access to the data hash store and a set of data to verify – it does not have access to the private collection containing the confidential information.

Here is an outline of the cross-verify process:

- **EcoBank** calculates a hash of the accepted offer data set, that they originally agreed to
- **EcoBank** cross-verifies this calculated hash against the written ledger hash of the offer for the contract in question, which was recorded at the same time the original private data transaction was written to the collection.
- **EcoBank** cross-verifies they match.
- 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
- 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 functionality, we will also implement add further smart contract functions, such as query transactions for use later in the lab. The reason is because, for convenience, we wanted to cut down on the number of smart contract upgrades required – given that it involves updating chaincode on 3 organisations.

4.2 Adding in cross-verify functionality and upgrading Smart Contract

The next phase is to copy in the source code that will add the cross-verification transactions. For convenience, we will also paste in the remaining code functions for this lab: namely, adding sample data, adding Private Data query functionality and adding an optional, custom approach cross-verify lab. This is because every upgrade needs the smart contract package installed to the peers of the three organisations in the lab. In the interests of saving time, it is added once below.

__ **116.** Click the Explorer icon in the VS Code editor and return to the **trading-contract.js** file.

__ **117.** Scroll down to **line 241,** where you find the code comment shown:

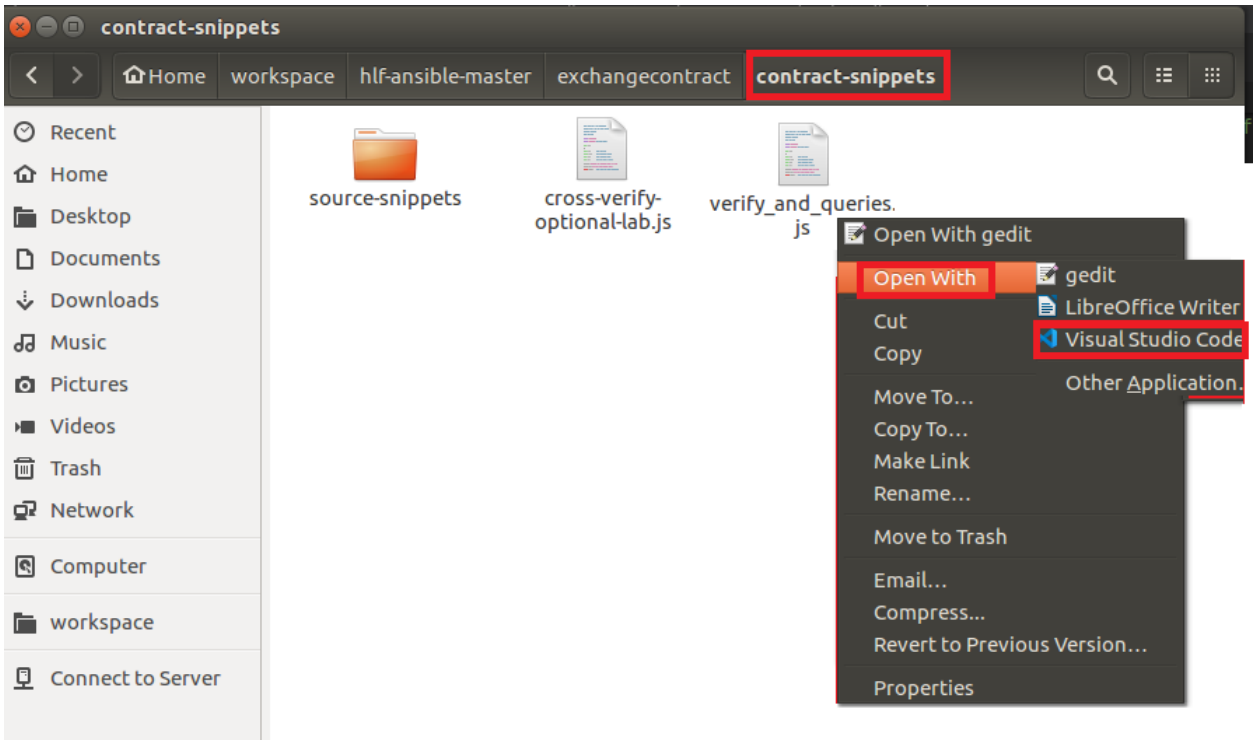
```
// UTILITY functions that may prove useful at some point
```

```
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 the exact point in the code, that you will paste in the remaining functions of the contract.

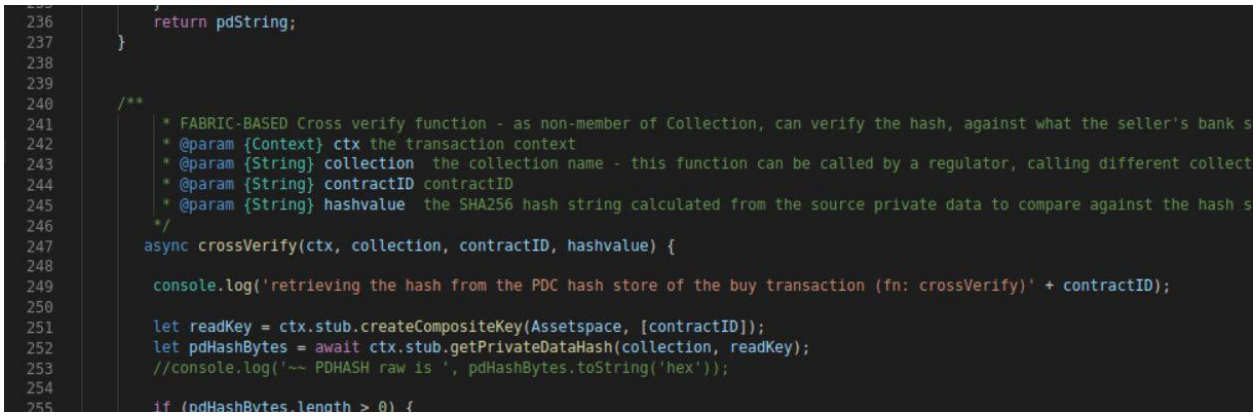
__ **118.** Open the Ubuntu File Explorer

__ **119.** Navigate to the following folder:
Home > workspace > hlf-ansible-master > exchangecontract > contract-snippets
Right click on `verify_and_queries.js` - select **Open With** ... **Visual Studio Code**



__ **120.** In VS Code, highlight all the code by pressing **CTRL+A** to select, then press **CTRL+C** to copy to the clipboard

__ **121.** Back in the **trading-contract.js** VS Code edit session, position the cursor at **line 239** and paste in the contents you copied to the clipboard using **CTRL+V**. Scroll back up to Line 239 to check it was pasted OK.

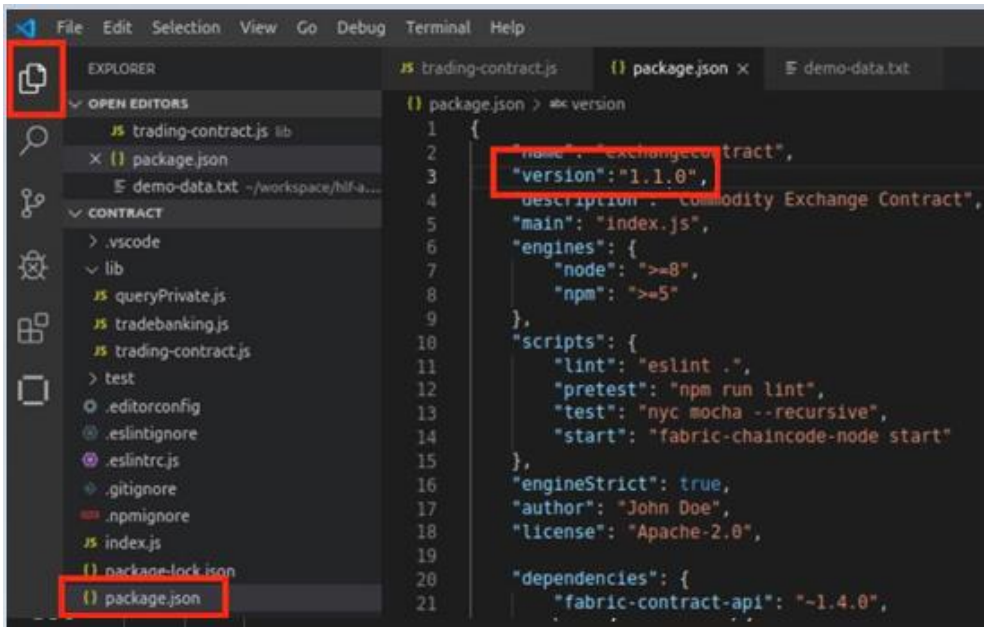


__ **122.** Right-click in VS Code and select **Format Document** to format the code and fix the indentation.

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

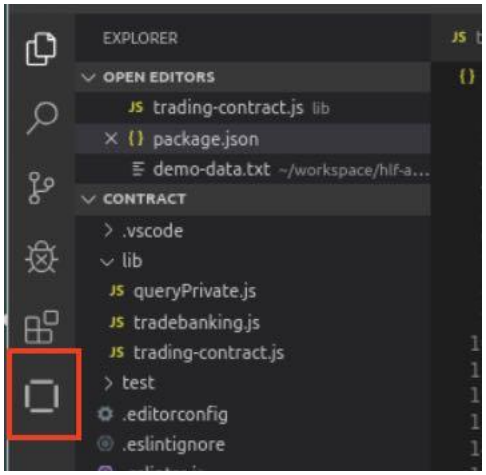
__ **123.** Save the **trading-contract.js** file by pressing **CTRL+S**.

__ **124.** 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.

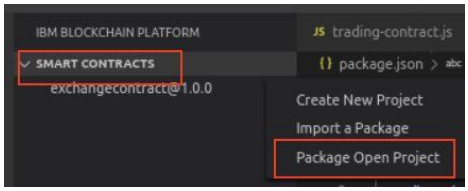


This will allow us to create a smart contract package with a new version number and enable us to upgrade the current contract instantiated on the channel.

__ **125.** Click on the IBM Blockchain Platform extension icon on the left in VS Code.

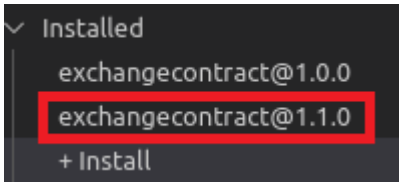


__ **126.** Click on the ellipses to the right of the Smart Contracts panel and select **Package Open Project**.



__ **127.** In the Fabric Environments view, click the Disconnect icon to disconnect from the current Fabric environment, and connect to the **digibank** environment.

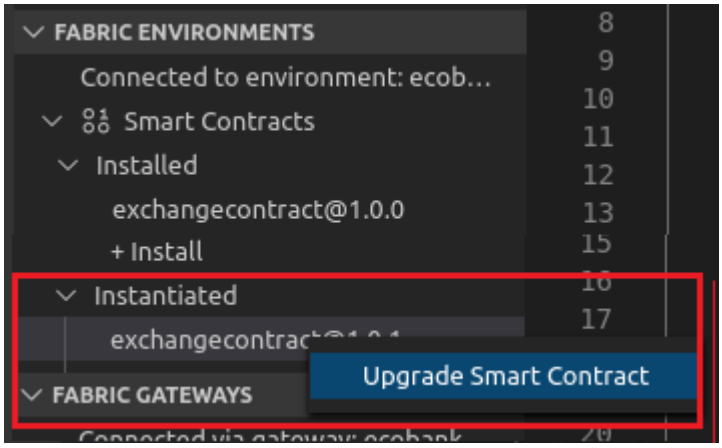
Click **+Install** and install the **exchangecontract@1.1.0** smart contract. Once completed, you'll get a confirmation it was performed successfully and it will appear in the list.



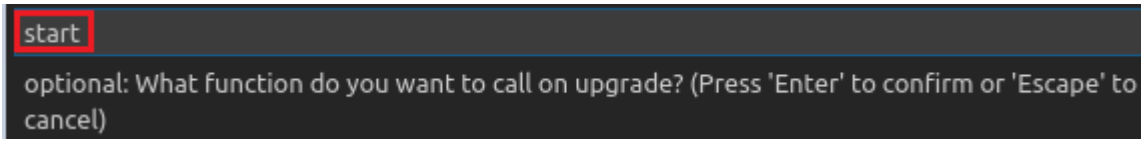
__ **128.** Repeat the previous step and install the new contract version onto the **finreg** environment.

__ **129.** Repeat the previous but one step and install the new contract version onto the **ecobank** environment.

__ **130.** 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.

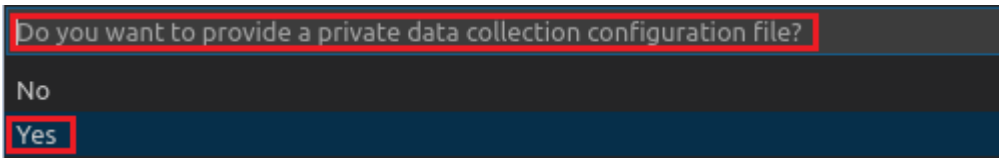


__ **131.** Enter **start** as the function to call on upgrade.

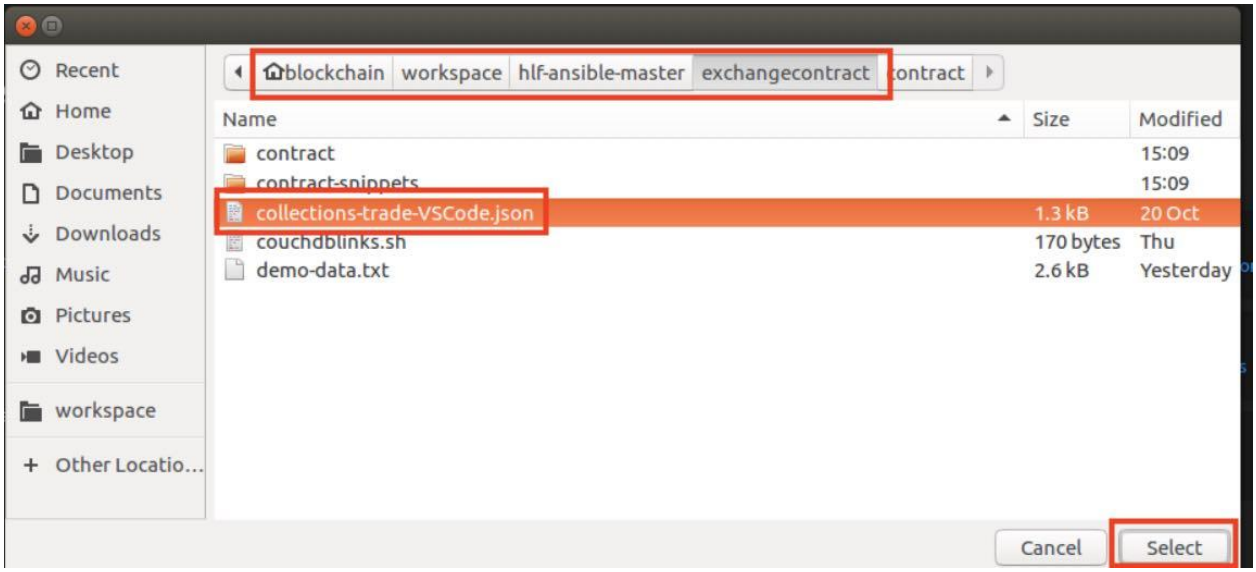


__ **132.** Press Enter to accept the default arguments to the function - there are none.

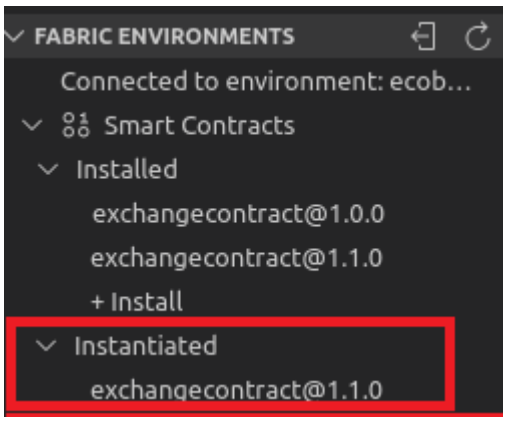
__ **133.** When asked if you 'want to provide a private data collection configuration file' – select '**Yes**'



__ **134.** 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:

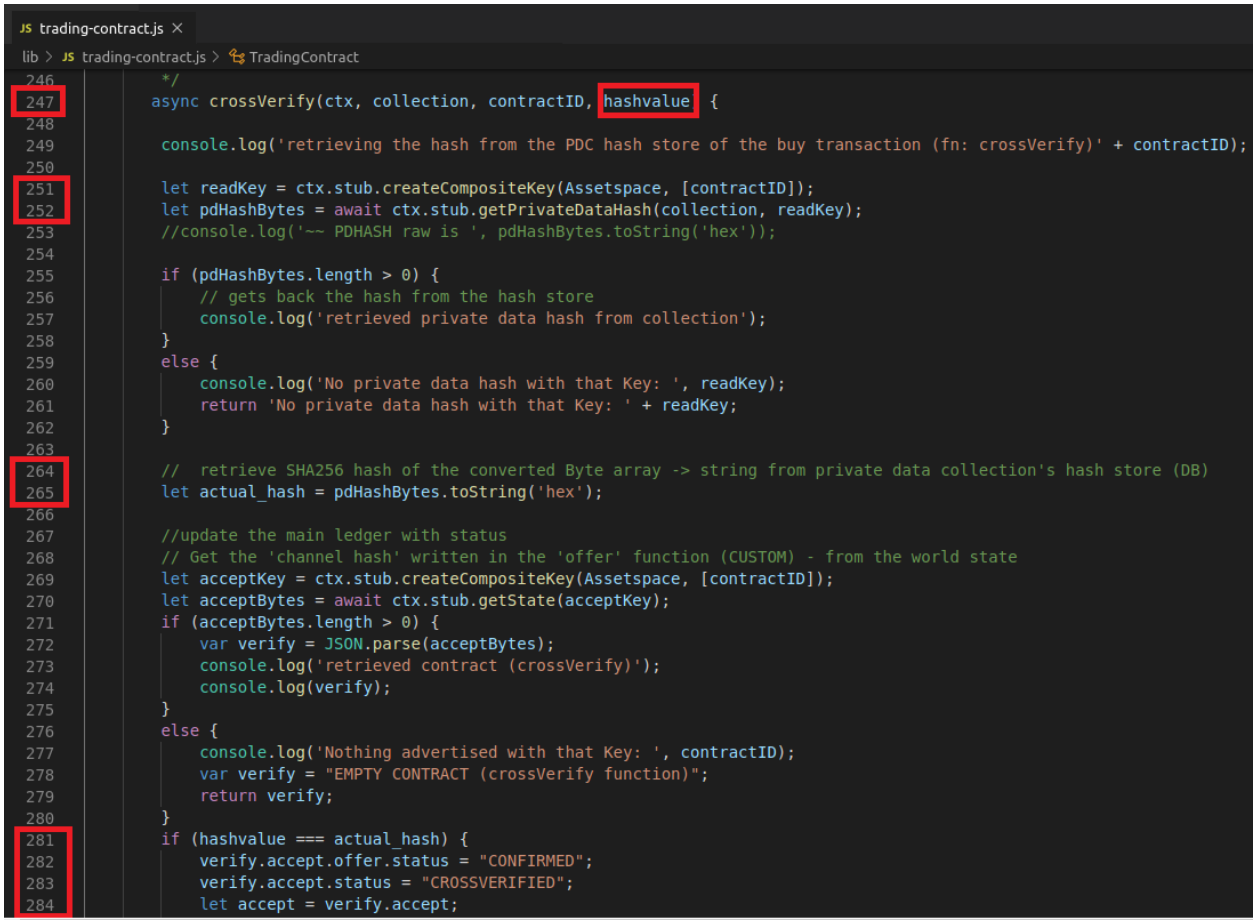


The smart contract has been instantiated on the tradechannel channel.

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.

__ **135.** In the **trading-contract.js** source in VS Code, scroll to **line 247.**

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



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 displayed a failed MATCH. If a match is made, lines **282-285** show that the status on the main channel ledger is updated to 'CROSSVERIFIED'

4.3 Perform the Cross Verification of Private Data

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

__ 136. 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
```

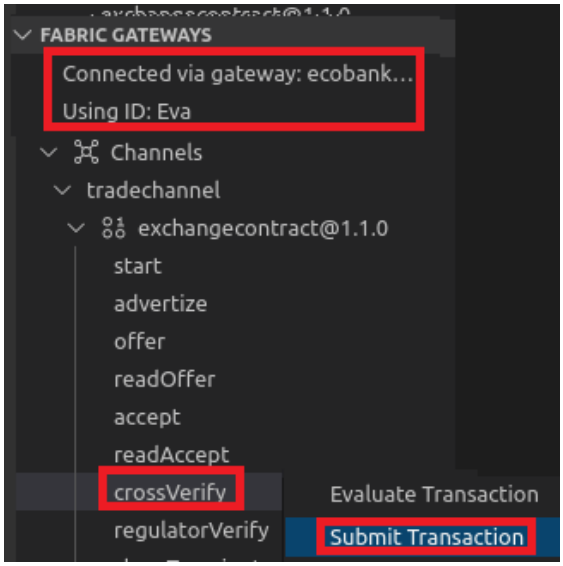
This will calculate a SHA 256 hash value of the source data EcoBank had agreed with DigiBank.

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 crossVerify smart contract transaction

__ 137. In the **Fabric Gateways** panel click **ecobank_gw** and connect with identity **Eva**

__ **138.** Expand the **Channels**/**exchangecontract@1.1.0.** Right-click **crossVerify** and click **Submit Transaction.**

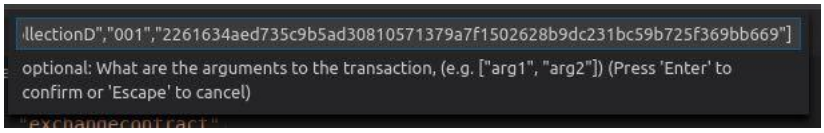
Note that we are running 'Submit Transaction' and not 'Evaluate Transaction' because we will be updating the status on the channel ledger for the trade contract "001" to be 'CROSSVERIFIED' if the verify pattern returns a 'MATCH' in the output pane.



__ **139.** 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 previously opened in VS Code

WARNING: if you copy/paste the parameter list 'as is' from the PDF document, you will NOT get a MATCH. This is because the 'paste' from PDF puts a space into the hash value instead of the carriage return. You should copy/paste the parameter list from the file 'demo-data.txt' supplied in the **exchangecontract** subdirectory – also ensure there are no trailing spaces after the closing ']' square bracket.

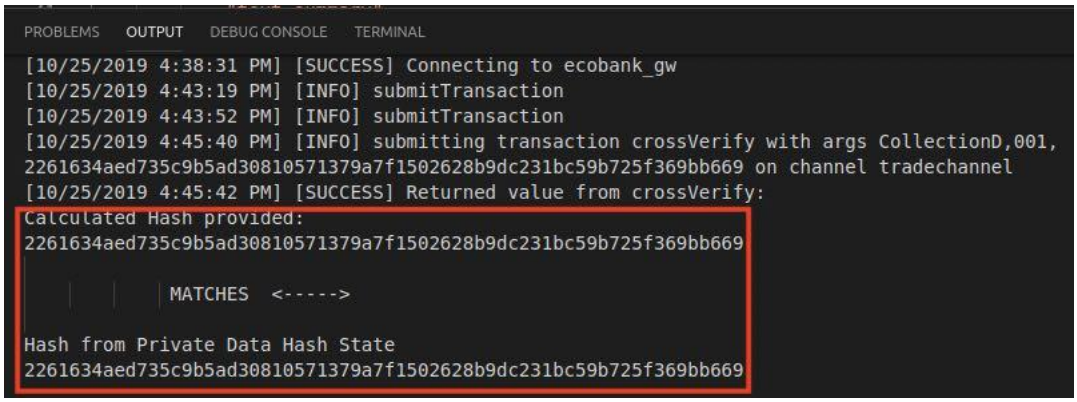
```
["CollectionD","001","2261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669"]
```



__ **140.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **141.** 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 for 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:

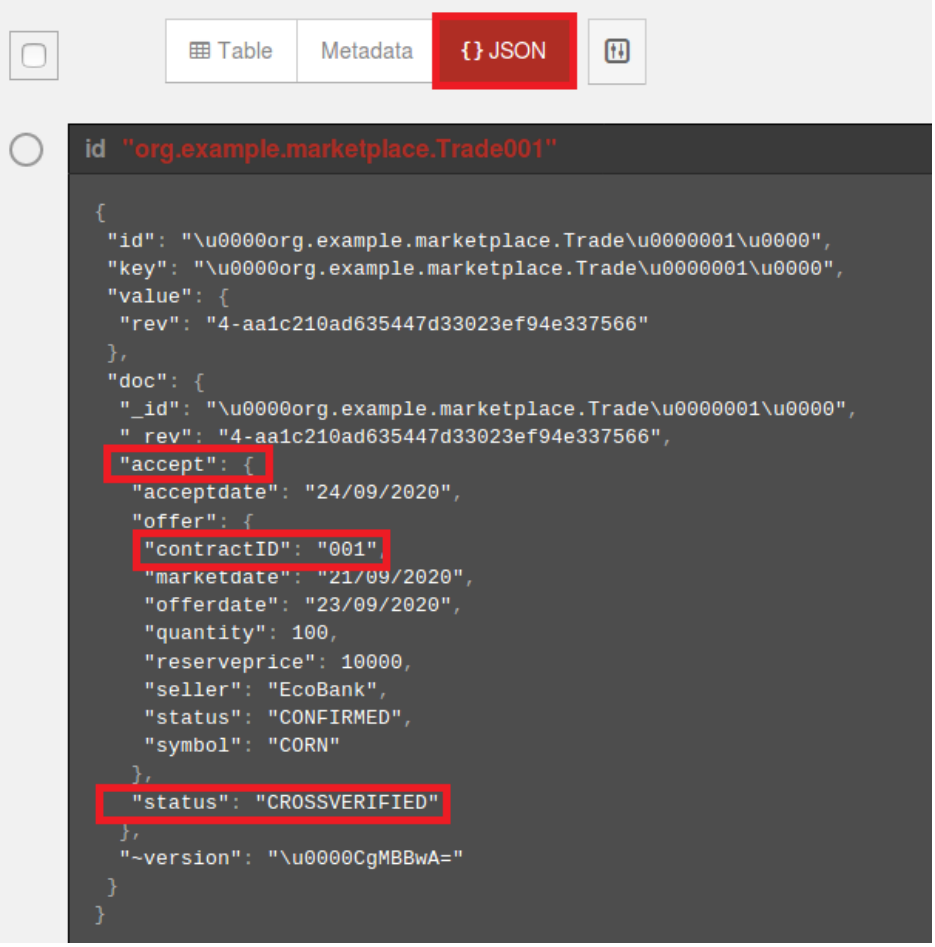


Having verified they match, we need to check that the channel ledger record itself, has a status of 'CROSSVERIFIED' (its updated by the **crossVerify** function).

__ **142.** Return to Firefox and click on the middle tab, which is **EcoBank's** CouchDB

__ **143.** As you've done before, click on the link to the channel ledger called **tradechannel_exchangecontract** in CouchDB

__ **144.** Click on the JSON button (alongside 'Metadata') for the open channel record and you should see that the status of the record is now set to 'CROSSVERIFIED' – this is the final step in processing an 'offer/accept' pair that has been cross-verified and is a status for all on the network to see.



Note that equally, **DigiBank, can take on the role of verifier** (in the scenario where it needs to be verified by the other party): That is, DigiBank can verify that the corresponding record in the EcoBank hash store ('CollectionE') provides the required evidence that the hash of the accept details stored, matches what was agreed by DigiBank (and calculated as a hash) – let's try this out

__ **145.** Disconnect as Eva and connect to the DigiBank gateway as **David**

__ **146.** Click on the **terminal** pane at the bottom of the screen in VS Code

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

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

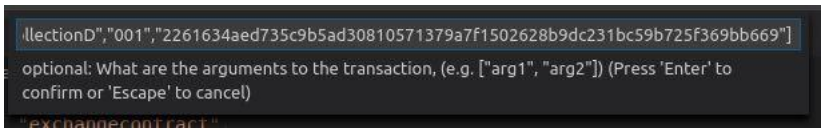
This will calculate a SHA 256 hash value of the source data provided to DigiBank to verify. It returns a hash value beginning **0720b**. For convenience, we will provide this exact 'accept' verification hash string, in the text file called 'demo-data.txt'.

__ **147.** Right-click on the **crossVerify** transaction and choose **Submit Transaction**

__ **148.** When prompted in the VS Code extension, paste in the following parameter list as a one-liner from the file 'demo-data.txt' in your VM image folder. Ensure you replace the existing '[]' text:

WARNING: if you copy/paste the parameter list 'as is' from the PDF document, it is likely you will NOT get a MATCH. This is because the 'paste' from PDF puts a space into the hash value instead of the carriage return. You should copy/paste the parameter list from the file 'demo-data.txt' supplied in the **exchangecontract** subdirectory – also ensure there are no trailing spaces after the closing ']' square bracket.

```
["CollectionE","001","0720b555e56eae6a658e735654a2b5ccd433a566d838f8e922
5df75bfae67360"]
```



__ **149.** There is no transient data for this transaction, just press **enter** when prompted, leaving the empty curly braces **{}**

__ **150.** Wait for the transaction to complete, then select the Output pane and review the messages.

It should provide a match of the EcoBank data hash (for which we calculated a hash for 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:

```
[11/5/2019 4:31:55 PM] [SUCCESS] Connecting to digibank_gw
[11/5/2019 4:36:22 PM] [INFO] submitTransaction
[11/5/2019 4:36:34 PM] [INFO] submitting transaction crossVerify with args CollectionE,001
[11/5/2019 4:36:36 PM] [SUCCESS] Returned value from crossVerify:
Calculated Hash provided:
0720b555e56eae6a658e735654a2b5ccd433a566d838f8e9225df75bfae67360

        MATCHES  <----->

Hash from Private Data Hash State
0720b555e56eae6a658e735654a2b5ccd433a566d838f8e9225df75bfae67360
```

4.4 Review the Regulator Verification function

Another transaction function we added is **regulatorVerify.** The code for this was also pasted in, immediately prior to the earlier upgrade. This is also a cross-verification function, but has checks to ensure that only the 3rd party regulator, **FinReg** can invoke - it is a read-only function. Let's examine the smart contract **regulatorVerify** function beginning from line **301** – as you can see, it has logic to check the calling MSP and reject if not the **FinReg** organisation.

```
301  async regulatorVerify(ctx, collection, contractID, hashvalue) {
302
303      let invokingMSPid = await this._getInvokingMSP(ctx);
304      console.log('regulatorVerify function called by..' + invokingMSPid); // written to the container
305
306      // if (invokingMSPid !== Regulator) return 'unauthorized to call this function as MSP:  ' + invokingMSPid;
307
308      if (invokingMSPid !== Regulator) throw new Error('unauthorized to call this function as MSP: ' + invokingMSPid);
309
310      console.log('retrieving the hash from the PDC hash store of the buy transaction (fn: regulatorVerify)' + contractID);
311
312      let acceptKey = ctx.stub.createCompositeKey(Assetspace, [contractID]);
313      let pdHashBytes = await ctx.stub.getPrivateDataHash(collection, acceptKey);
314
315      if (pdHashBytes.length > 0) {
316          // gets back the hash from the hash store
317          console.log('retrieved private data hash from collection');
318      }
319      else {
320          console.log('No private data hash with that Key: ', acceptKey);
321          return 'No private data hash with that Key: ' + acceptKey;
322      }
323
324      //  retrieve SHA256 hash of the converted Byte array -> string from private data collection's hash store (DB)
325      let actual_hash = pdHashBytes.toString('hex');
326
327      if (hashvalue === actual_hash)
328          return '\nCalculated Hash provided: \n' + hashvalue + '\n\n            MATCHES  <----->  \n\nHash from Private Data Hash State \n'
329      else
330          return 'Could not match the Private Data Hash State: ' + actual_hash;
331
332  }
276          else {
```

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 the **PutPrivateData** method 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.

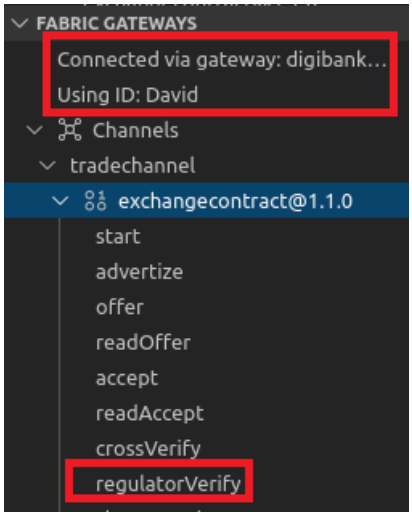
Lines 326-328 show what actions are taken dependent on whether there is a MATCH or failed MATCH between the hashes.

Further transactions added are also helper transactions like **ShowTransient** (show inputs for a given Transient set), 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 (the data is used for querying later in the lab).

4.5 Perform the Regulator Verification of Private Data

The next step is to cross-verify data as a Regulator. This transaction can only be called by someone from the regulator organisation and we'll test that next.

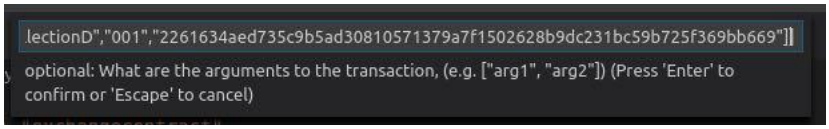
__ **151.** Still connected as the identity 'David' from DigiBank, highlight the **regulatorVerify** transaction, **right-click** on **regulatorVerify** and click **Evaluate Transaction**



__ **152.** Provide the following one-liner parameter list below as arguments when prompted, replacing the existing '[]' text:

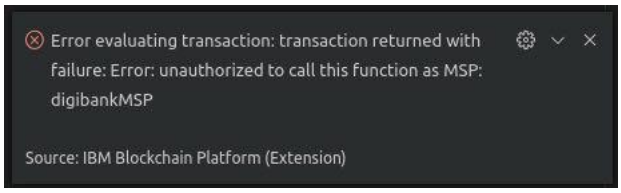
WARNING: if you copy/paste the parameter list 'as is' from the PDF document, you will NOT get a MATCH. This is because the 'paste' from PDF puts a space into the hash value instead of the carriage return. You should copy/paste the parameter list from the file 'demo-data.txt' supplied in the **exchangecontract** subdirectory – also ensure there are no trailing spaces after the closing ']' square bracket.
.

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



__ **153.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

You should get an error once you expand the message in VS Code. It explains that you are not authorized to call this transaction because you're not an identity issued by the regulator organisation.

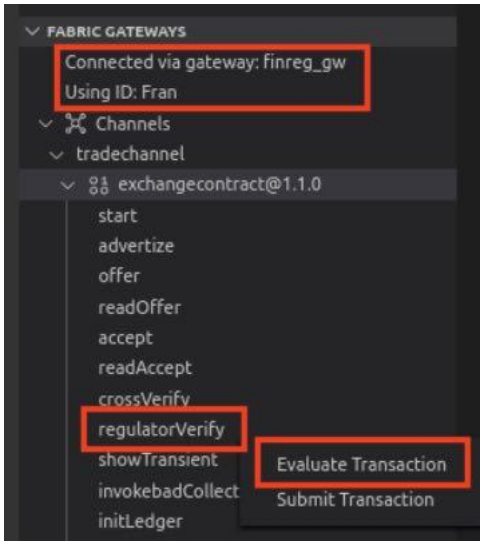


Next, you will now switch to **FinReg** and invoke the transaction with an authorized identity.

__ **154.** Hover over the **Fabric Gateways** panel and disconnect from DigiBank's gateway **digibank_gw**

__ **155.** Click on FinReg's gateway **finreg_gw**, connect with identity **Fran.**

__ **156.** Once again, **right-click** the **regulatorVerify** transaction from the transaction list and click **Evaluate Transaction**

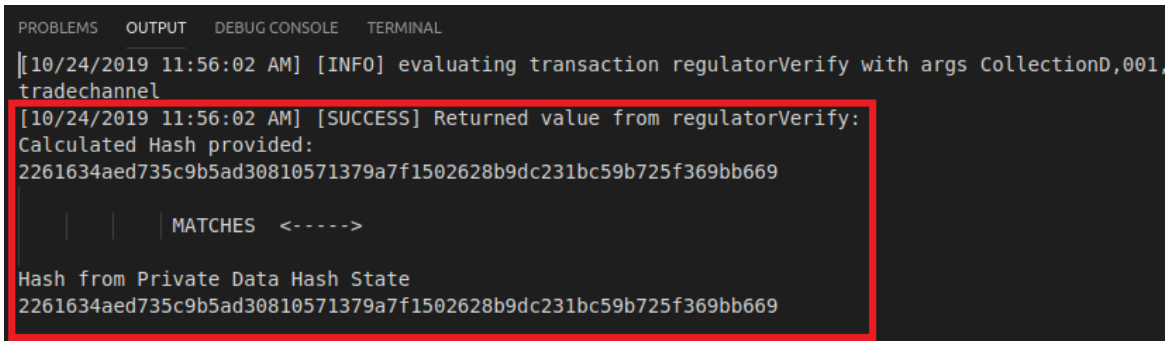


__ **157.** Provide the same parameter list as before when prompted:

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

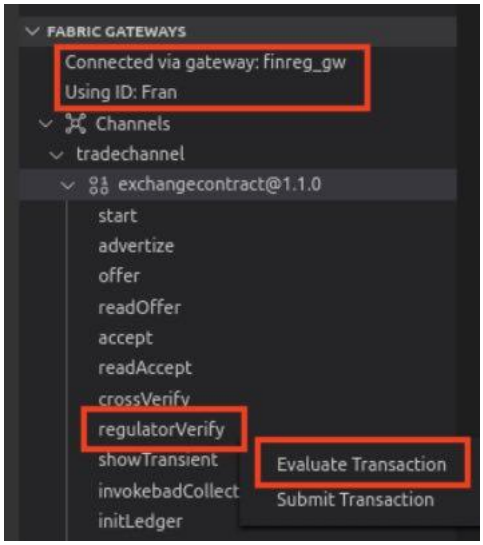
__ **158.** Again, there is no transient data for this transaction so when promoted, hit **enter** - leaving the empty curly braces **{}**

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 invokes a readonly transaction.



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 to see what output we get.

__ **159.** Once again as **Fran**, **right-click** the **regulatorVerify** transaction from the transaction list and click **Evaluate Transaction**



__ **160.** Provide the same parameter list as before – but this time - change the first digit of the hash to become a **9** instead of a **2**:

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

__ **161.** Again, there is no transient data for this transaction so just press **enter** leaving the empty curly braces **{}** as-is

Now you should see that there is no match.

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 organisation(s) involved.

Review

In this part of the lab

- You have modified the smart contract to add the new transaction functions provided, and installed it on peers for all 3 organisations, as well as upgrading the running smart contract 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 on the ledger, to verify data shared for cross-verification purposes.
- You have worked with a **regulatorVerify** transaction, which enables the regulator **FinReg** to cross-verify or 'audit' private data evidence, ensuring it matches (once a hash is calculated) against the corresponding hash records in the private data hash stores.
- You have tested the **regulatorVerify** with a hash that does not match to check it shows the output where a mismatch has occurred.

5 Add Demo Data and Private Query Transaction

5.1 Introduction

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

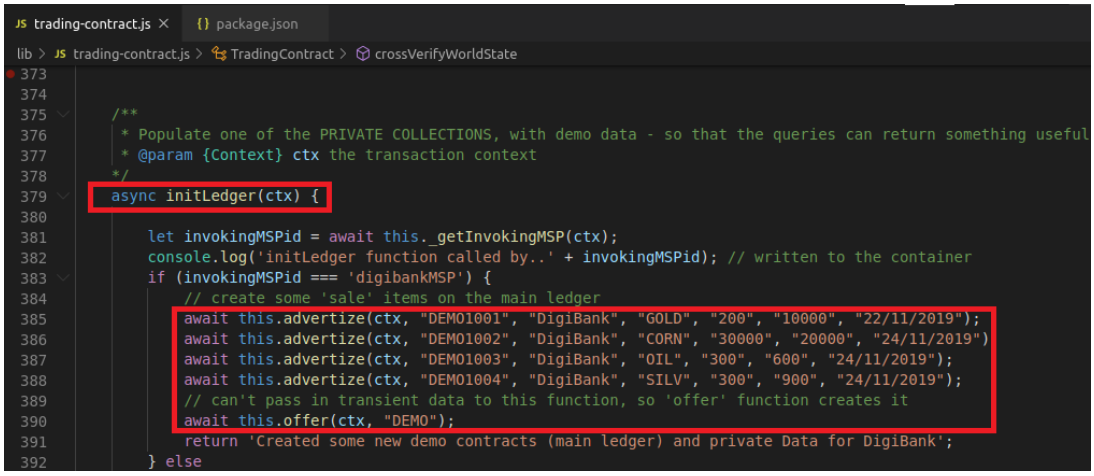
- Add sample data to two separate Private Data collections for query purposes – and
- Execute Private Data query transactions that allow you to perform 'rich queries' against the private collection data.

Note that the private data transaction functions described in this section, were added when the code block was copied into the smart contract and subsequently upgraded to v1.1.0 of the **exchangecontract.**

5.2 Review Demo Data and private data Rich Query transactions

Let's look at the transactions:

__ 162. Click the **Explorer** icon open **trading-contract.js** if it is not already displayed. Scroll to line 379 in the code.



The **initLedger** transaction creates four DEMO contracts that are advertised and – against those contract IDs, creates some offer and offer/accept sample data respectively, in both DigiBank's and EcoBank's private data collections.

Next, let's briefly review the two query transactions starting at line **411**.

The first is **privatequeryAdhoc** – this can take either a 'named query' as a parameter (example: a query named "pricequery" - find Contracts with a value > 1000 USD) or it takes an 'ad-hoc' query string – where you create a 'custom query' selector string (in 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 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.

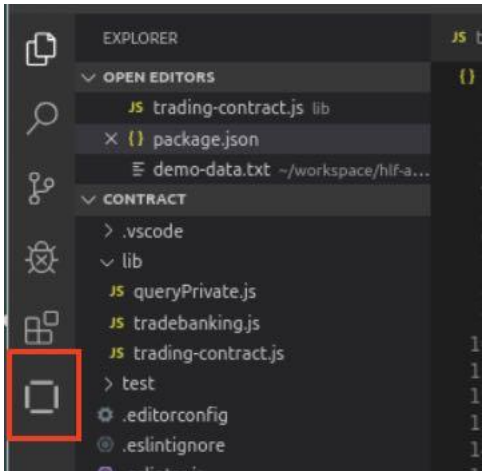
There is also a **privatequeryPartialKey** query transaction that does a private data collection query by partial key (e.g. by Asset namespace, to find all records for that namespace, in the collection name supplied).

5.3 Perform Demo Data creation for querying Private Data

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

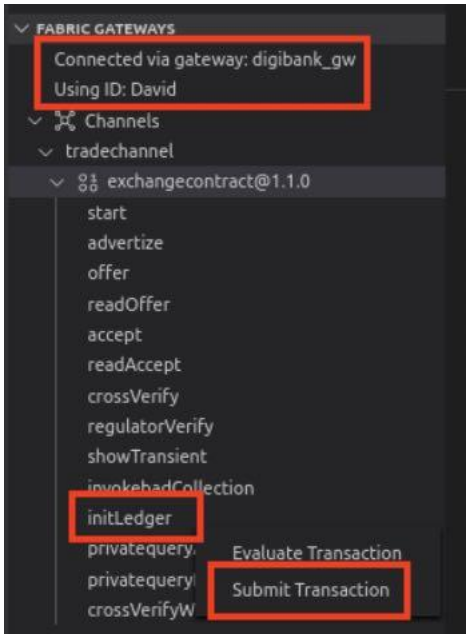
We need to return to the IBM Blockchain Platform extension.

__ **163.** Click on the IBM Blockchain Platform **extension icon**.



__ **164.** On the **Fabric Gateways** panel disconnect from FinReg's gateway and click **digibank_gw** and connect with identity **David.**

__ **165.** Expand the **exchangecontract@1.0.0** contract to get a transaction list, and right-click the **initLedger** transaction**.** Click **Submit Transaction**



__ **166.** There are no parameters, just press **enter** leaving the empty square brackets **[]**

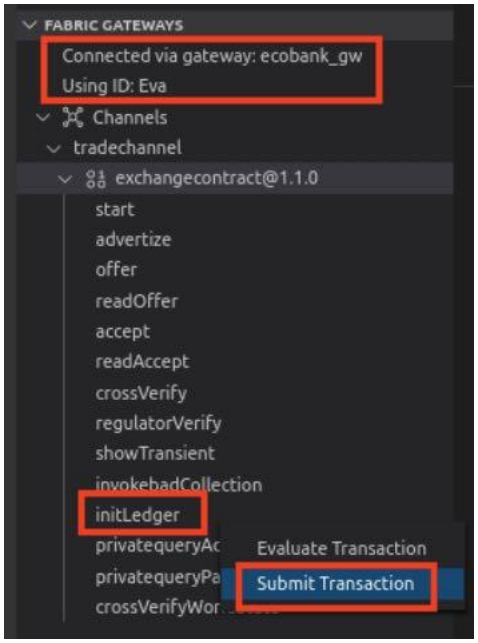
__ **167.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

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
```

__ **168.** On the **Fabric Gateways** panel disconnect from DigiBank's gateway and click **ecobank_gw** and connect with identity **Eva.**

__ **169.** Scroll down to the transaction **initLedger**, right click then click **Submit Transaction**



__ **170.** Once again, there are no parameters, just press **enter** leaving the empty square brackets **[]**

__ **171.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **172.** Review the messages in the output pane – it should indicate that the private demo data was created this time for EcoBank

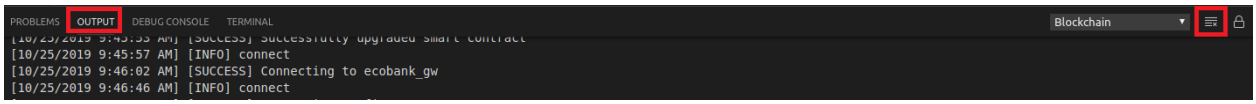
```
Returned value from initLedger: Create some private Data for EcoBank
```

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.

5.4 Perform Rich Queries against the Private Data Collections

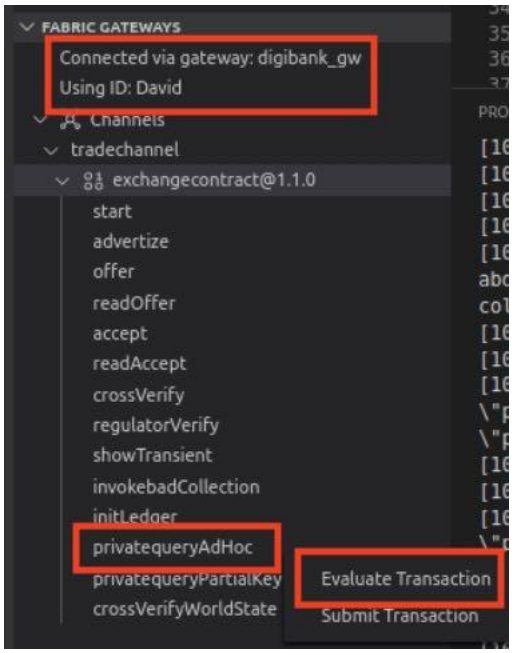
This section performs the execution of the queries we described earlier. So that we can more easily see what is happening, we're going to reset the Output pane in VS Code and make it larger.

__ **173.** 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.



__ **174.** On the **Fabric Gateways** panel disconnect from EcoBank's gateway and click **digibank_gw** and connect with identity **David.**

__ **175.** Expand the **exchangecontract** and right-click **privatequeryAdhoc.** Click **Evaluate Transaction**



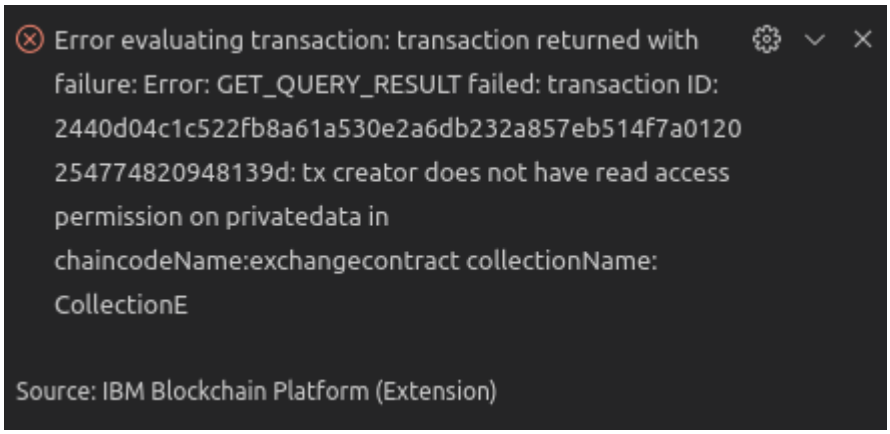
__ **176.** When prompted, **copy and paste** the following parameters, replacing the '[]' text offered with the following parameter list:

```
["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.

__ **177.** 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)



__ **178.** Close the error popup message, and clear the Output pane using the **Clear Output** icon, as shown previously.

__ **179.** Once again, right-click **privatequeryAdhoc** and click **Evaluate Transaction.**

__ **180.** 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

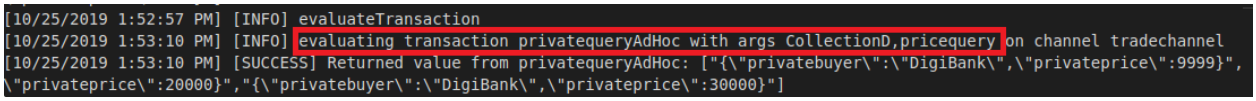
__ **181.** Once again, right-click **privatequeryAdhoc** and click **Evaluate Transaction**.

__ **182.** When prompted, paste in the following parameters, replacing the '[]' text offered with the following 'named' query, 'pricequery':

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

__ **183.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **184.** 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 are two records from the earlier listed results that are under 1000 USD. This is because out example pricequery is hard coded to show only values over 1000.



Next you will try some queries as EcoBank.

__ **185.** On the **Fabric Gateways** panel disconnect from DigiBank's gateway and click **ecobank_gw** and connect with identity **Eva.**

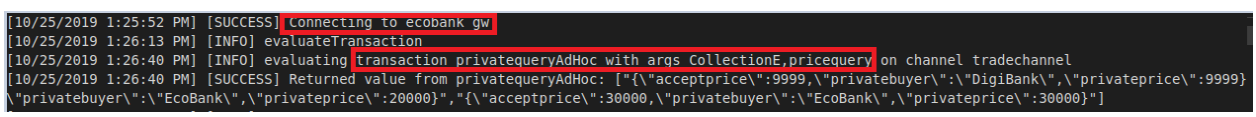
__ **186.** Expand the **exchangecontract** and right-click **privatequeryAdhoc**. Click **Evaluate Transaction**

__ **187.** When prompted, paste in the following parameters, replacing the '[]' text offered with the following – and press **enter**

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

__ **188.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **189.** 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.



Finally, lets 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.

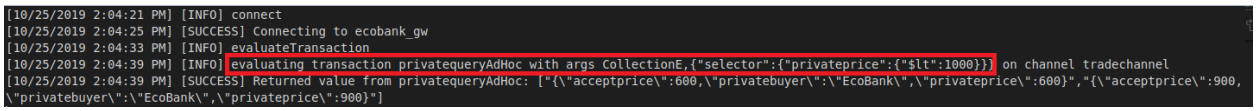
__ **190.** Once again, right-click **privatequeryAdhoc** and click **Evaluate Transaction**.

__ **191.** 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}}}"]
```

__ **192.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **193.** 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").



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

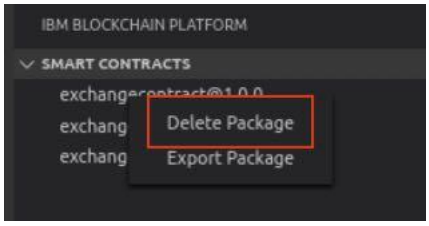
Review

You've now successfully:

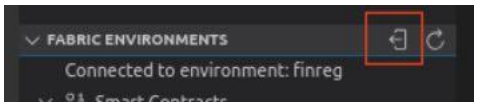
- created some additional sample data
- performed queries against private data collections
- understood more about using rich queries to filter result sets, by using different criteria.

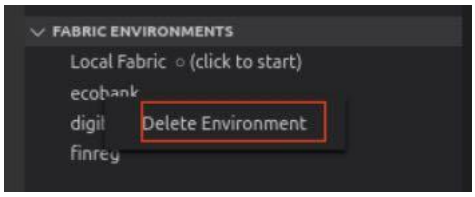
Finally, you need to clean up the virtual machine in preparation for the next lab. You will carry out the steps for this on the next page.

__ **194.** In the **Smart Contracts** panel **right-click** each package in turn and click **Delete Package**.

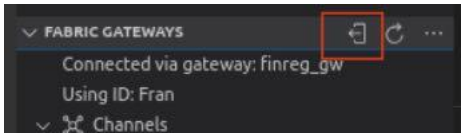


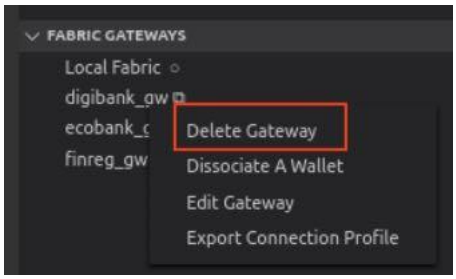
__ **195.** In the **Fabric Environments** panel **Disconnect** the current environment, and then **right-click** each environment and click **Delete Environment**.



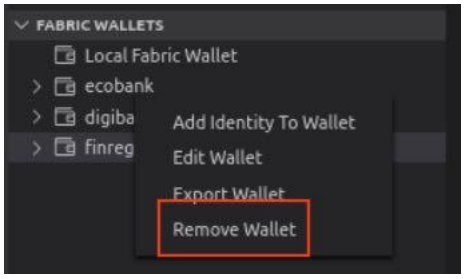


__ **196.** In the **Fabric Gateways** panel **Disconnect** the current gateway, and then **right-click** each gateway in turn and click **Delete Gateway**.

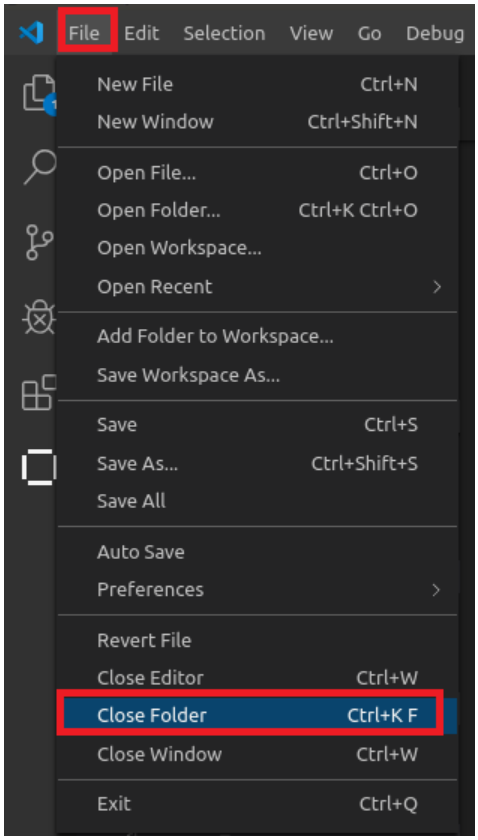




__ **197.** In the **Fabric Wallets** panel **right-click** each wallet in turn and click **Remove Wallet**.



__ **198.** Next, in VS Code, close the open smart contract project by clicking on the menu bar – select 'File ....Close Folder'



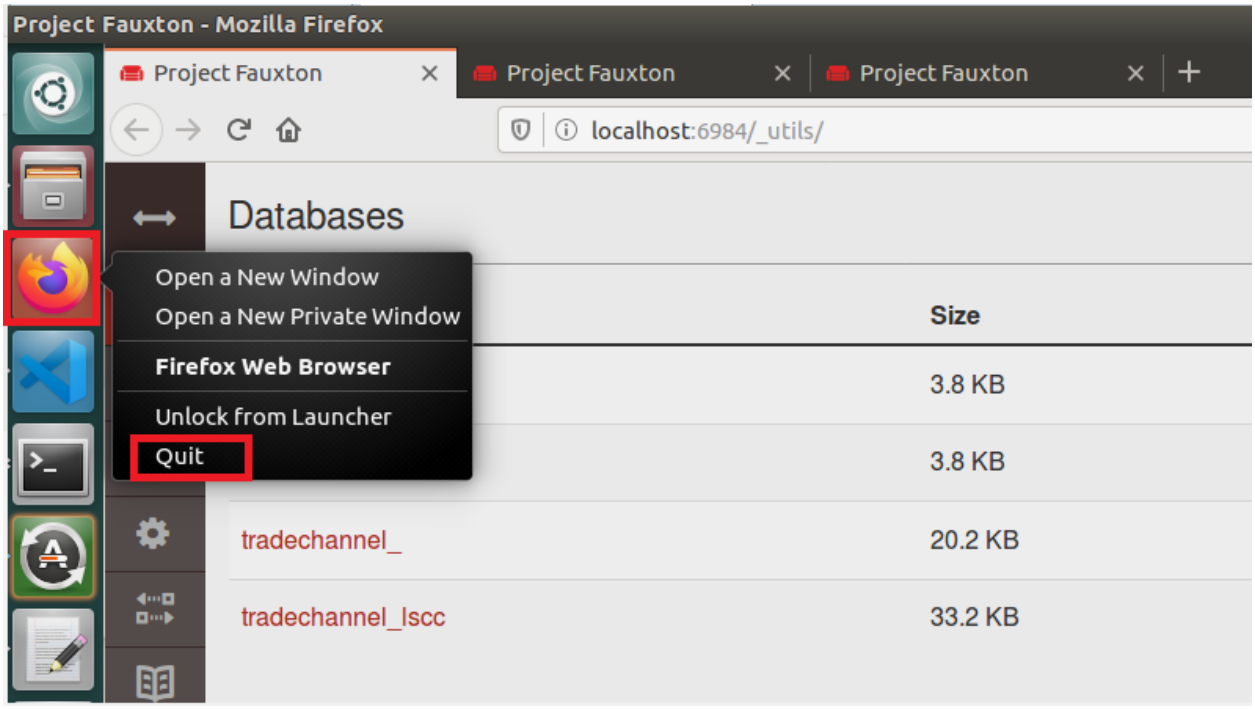
__ **199.** In the terminal window, use the following two commands to navigate to the **hlf-ansible-master** folder and teardown the custom network using a bash script:

```
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
```

__ **200.** From the task bar, right-click on the Firefox icon and select 'Quit' – confirm to close all tabs when prompted



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, { buyer, offerprice, 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, {buyer, offerprice, acceptprice, 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 or 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, simplex private data sets), 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 the hopes of finding a match against 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 clearup 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** …