Lab Center – Hands-On Lab

Session 1183

Developing for IBM Blockchain Platform

Part 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 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, and ibm.com 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.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
OpenShift is a trademark of Red Hat, Inc.
UNIX is a registered trademark of The Open Group in the United States and other countries.

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.

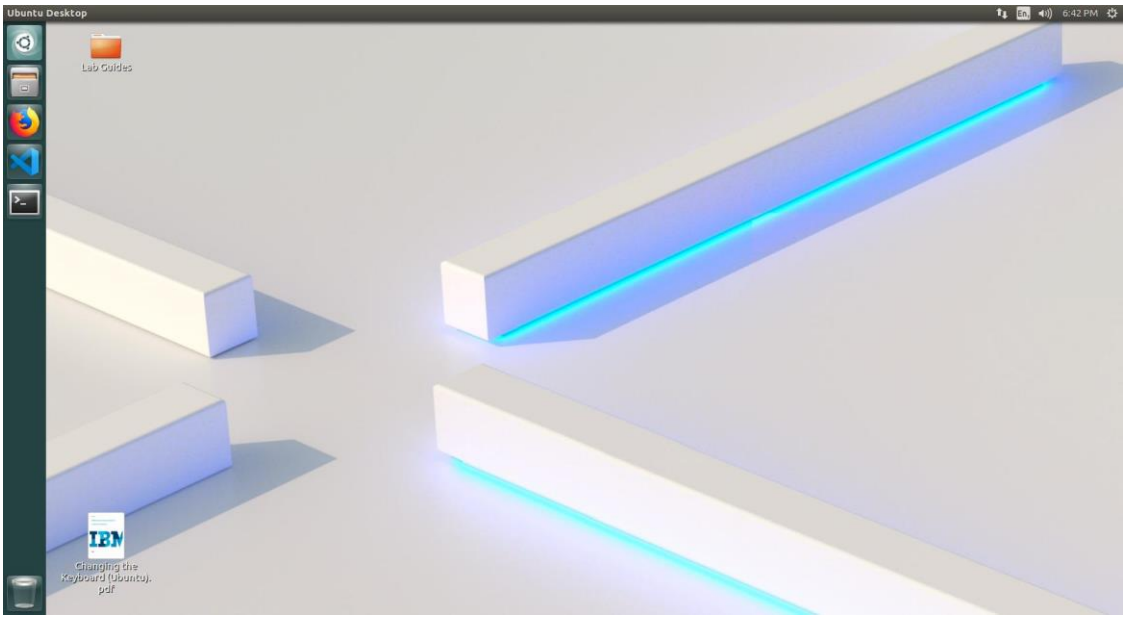
***Note**: The screenshots in this lab guide were taken using version **1.43.2** of **VS Code**, and version **1.0.26** of the **IBM Blockchain Platform** extension. If you use different versions, you may see differences from those shown in this guide.*

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**".

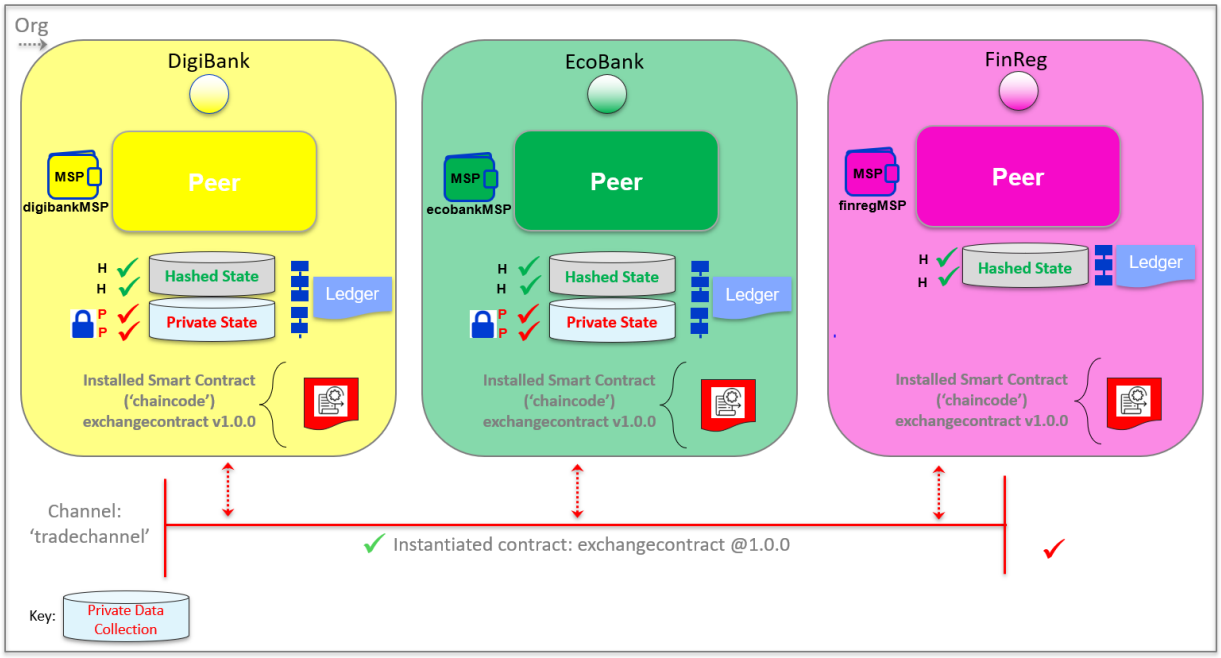


Simplified Commodity Trading Exchange

DigiBank and **EcoBank** participate in a Commodity Exchange business network – 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 counteroffers) 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 exercise we see that EcoBank and DigiBank have independent private data collections. 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.

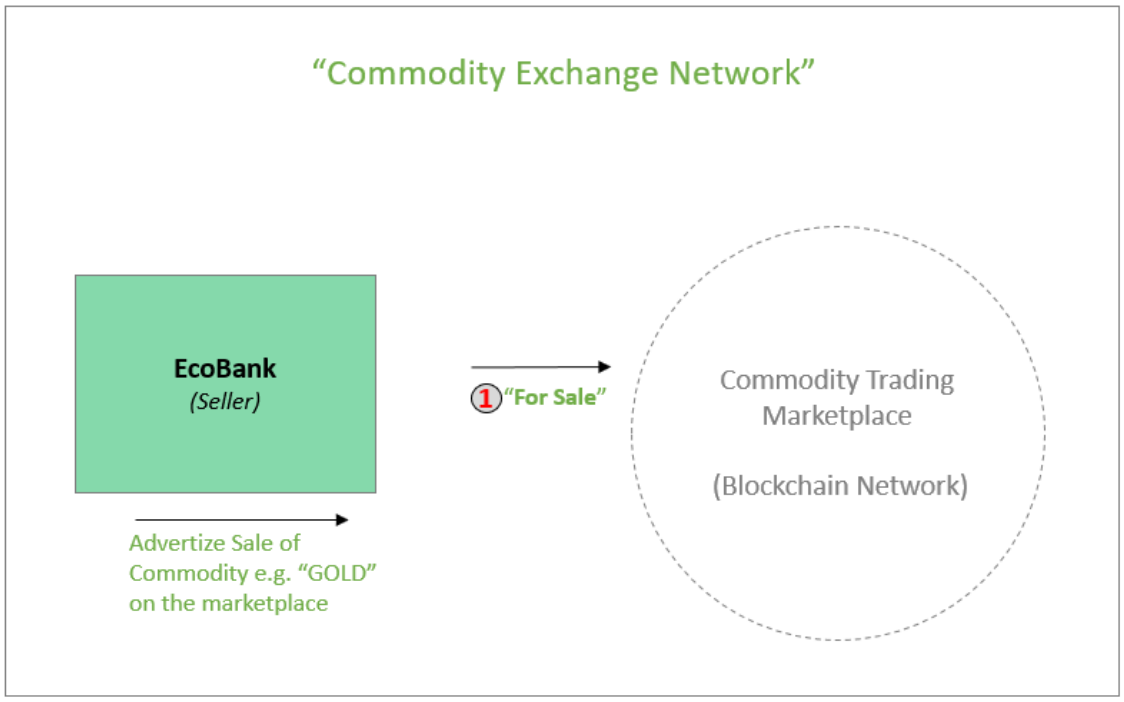
Commodity Trading Exchange: Overview



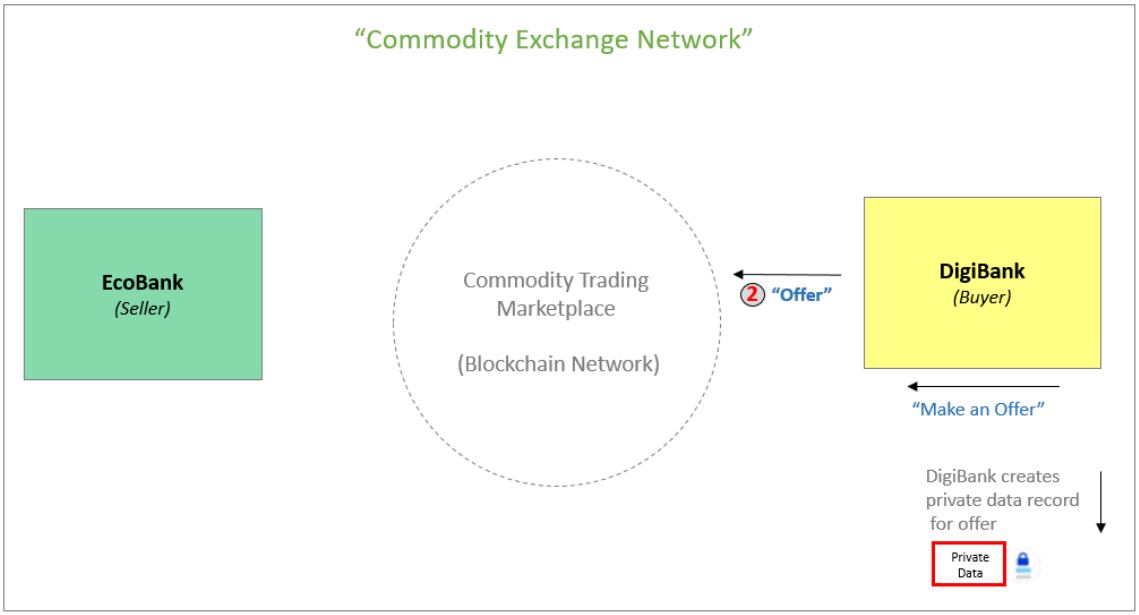
Notice in the above diagram, the regulator **FinReg** does not store any private data of its own. It does, however, have a copy of the ledger incl. the hashed state ledger of the private data. Its role is to audit records – e.g. third-party verification or dispute resolution of private data, shared with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

The smart contract reflects the transactions in this private data scenario – let's examine the transactions in the lab:

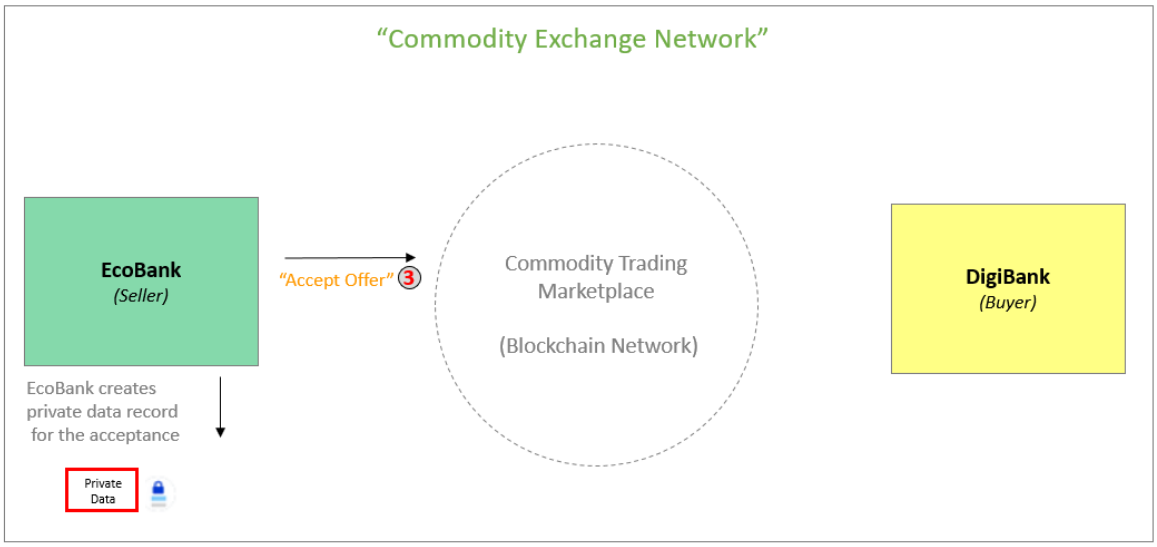
1. **EcoBank** advertises or 'broadcasts' a Commodity Contract for sale on the marketplace



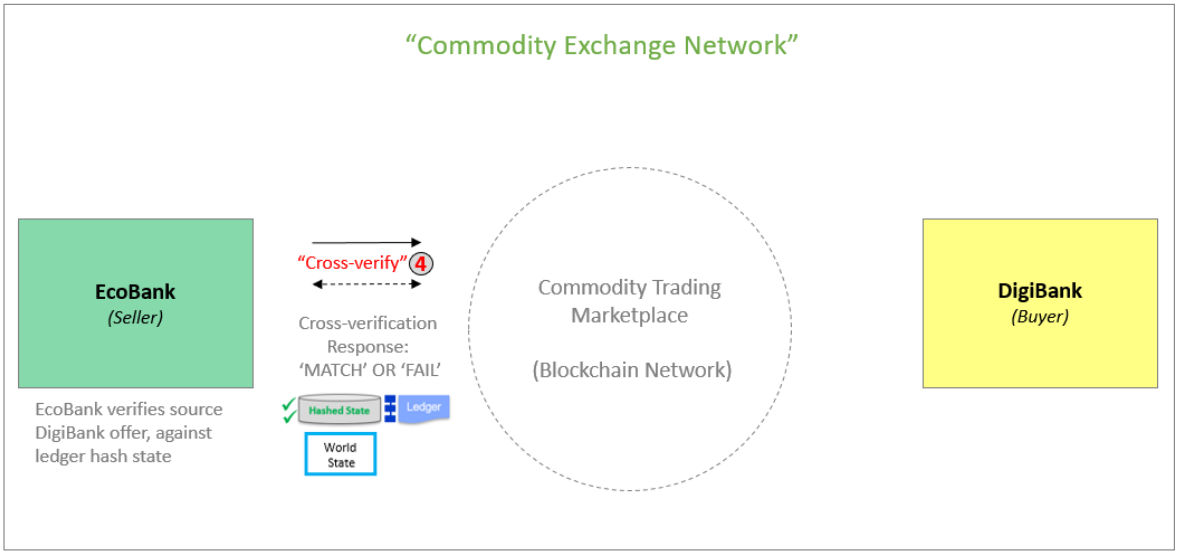
2. **DigiBank** makes an offer to EcoBank. Note that DigiBank writes a record of this offer, to its own private data store. A cryptographic 'hash' of the record is also recorded on the blockchain (channel world state).



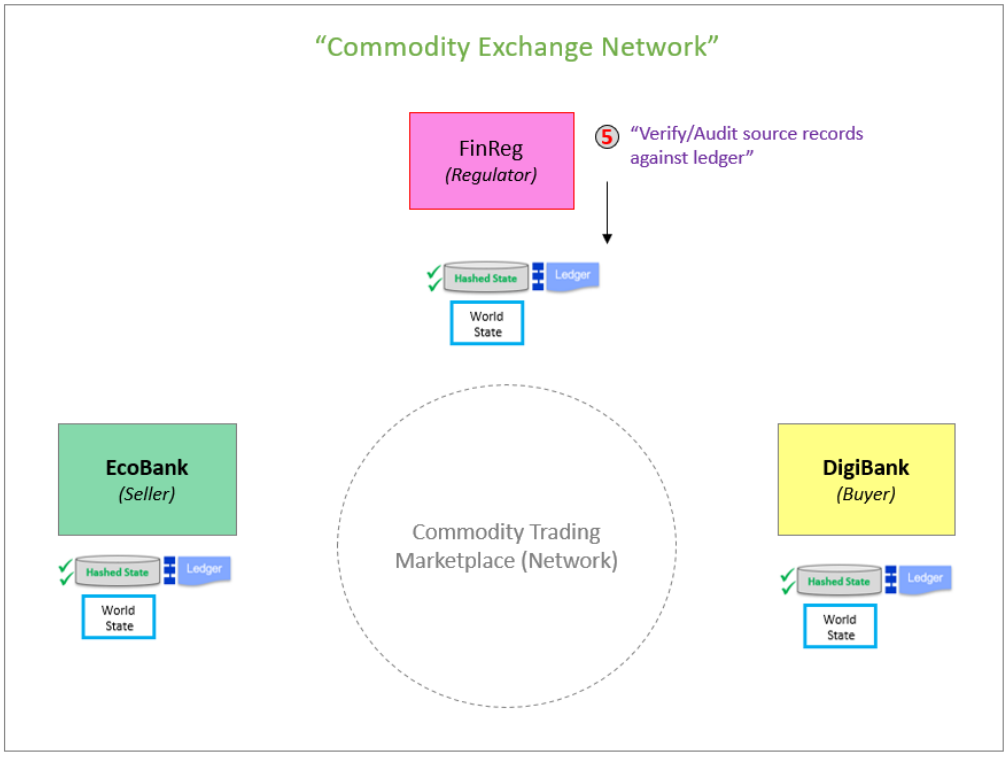
3. **EcoBank** receives the offer and informs DigiBank that it accepts the offer; EcoBank writes its own 'accept with offer' record to its own private data store – the offer on the ledger is, as of yet, UNVERIFIED – meaning, it needs to be cross-verified.



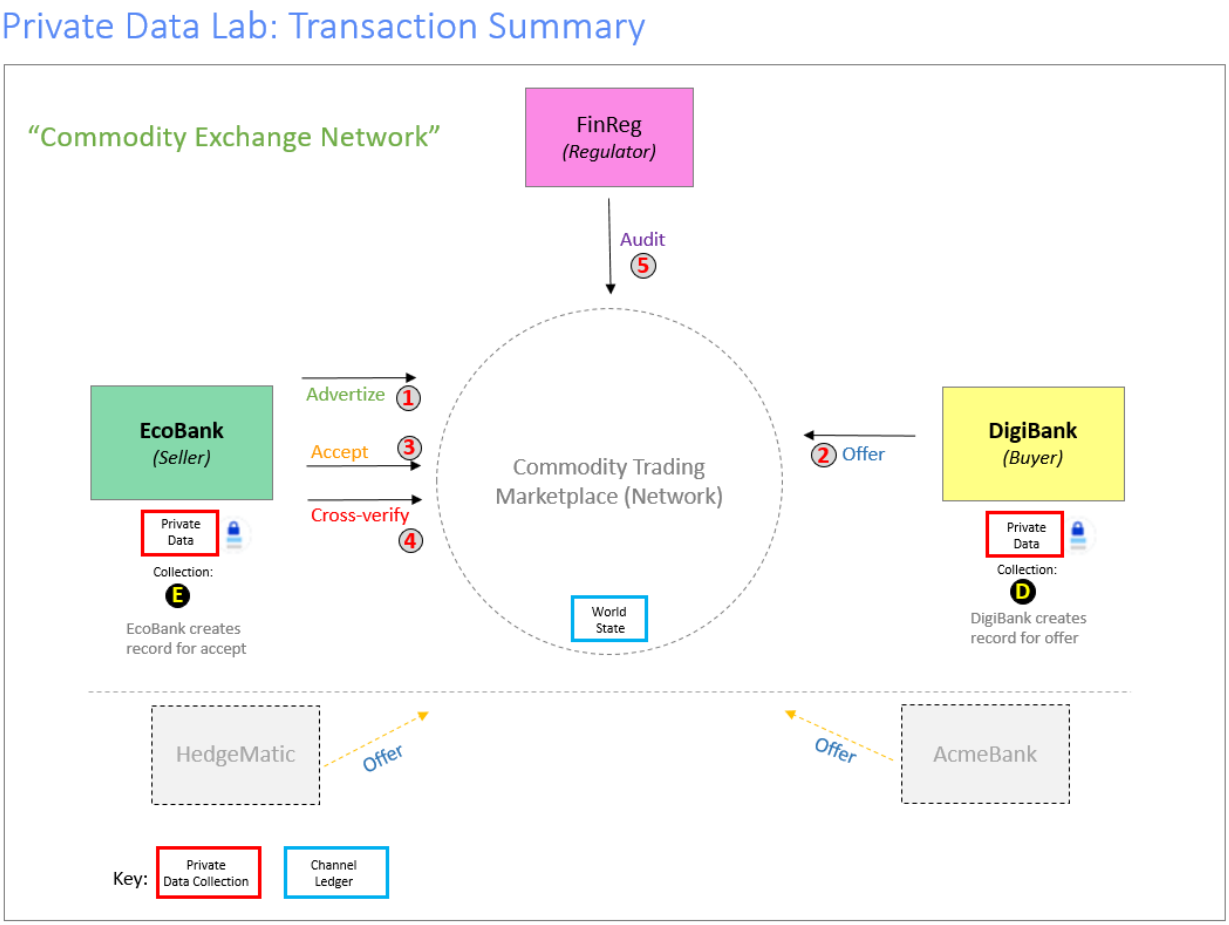
4. **EcoBank** calculates a hash of the source offer private data and verifies this hash, against the private data hash that was written by DigiBank in step 2. The smart contract compares the hashes and if they match, will update the accept/offer to 'CROSSVERIFIED' – this status attribute is not actually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



5. **FinReg**, a 3rd party regulator, carries out audit checks (sometime later) and takes source offer (or accept) data received out-of-band, and cross-verifies the data against the hashes originally written to the ledger.



The following diagram contains a consolidated transaction summary:



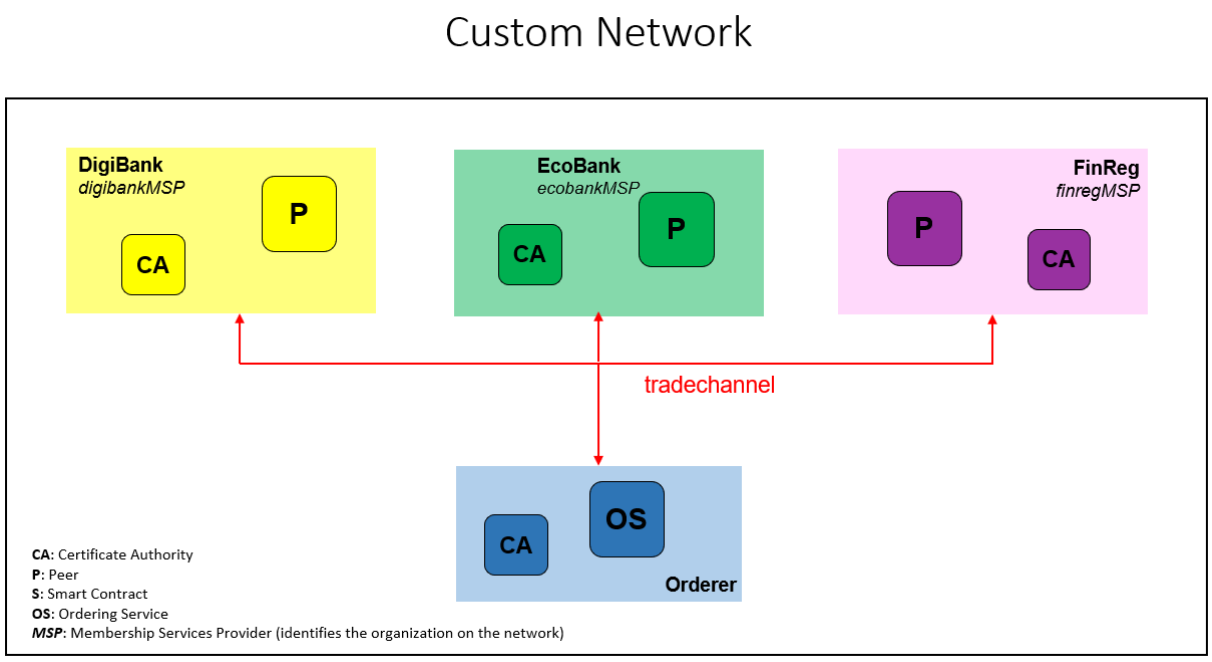
Note that we focus on 3 organisations in this lab exercise; but remember, there could eventually be more (reference the grey boxes in the above diagram) already joined the network and be on the same channel.

FinReg, the regulator will later be required 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. The network is built using Ansible, based on a template in the IBM Blockchain Github repo at: https://github.com/IBM-Blockchain/ansible-examples/tree/master/two-org-network. Ansible is an open source configuration and deployment automation tool. The network comprises a set of Docker containers built as a result of running the Ansible build script called `site.yml`. The Lab requires no knowledge of Ansible, just an awareness that it is being used.

The network uses one Fabric channel, **tradechannel.** Each Peer uses CouchDB as the state database used for rich queries in the lab. The IBM Blockchain Platform for VS Code extension has a built-in feature to import the ansible-built Fabric environment (i.e. all Fabric nodes, gateways and wallets/identities generated by Ansible).



Lab Structure

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

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

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

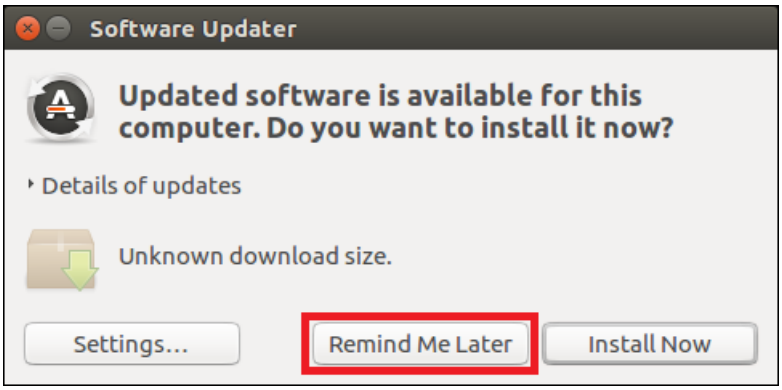
Part 3 of this lab will focus on the Offer and Accept transactions and the writing of the initial Private Data.

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

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

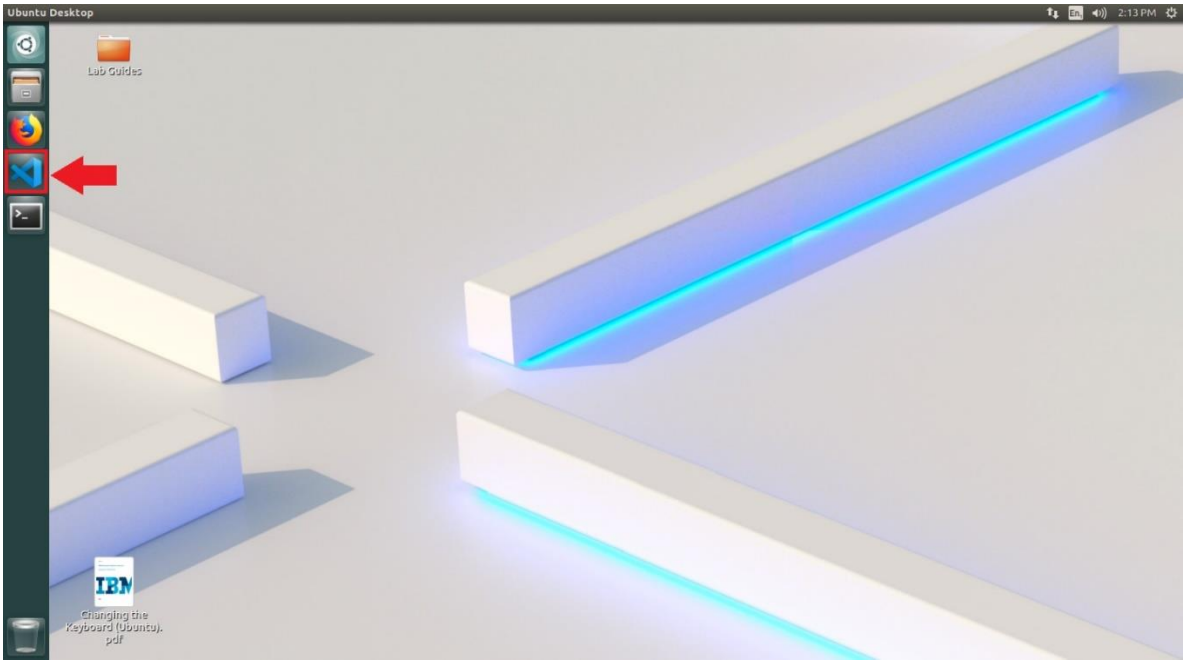
Part 6 adds an optional part to the lab: You will perform a custom Cross-Verify transaction, where a hash of the offer data is calculated and written to the world state record - and later, the stored value is cross-verified against a calculated hash

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

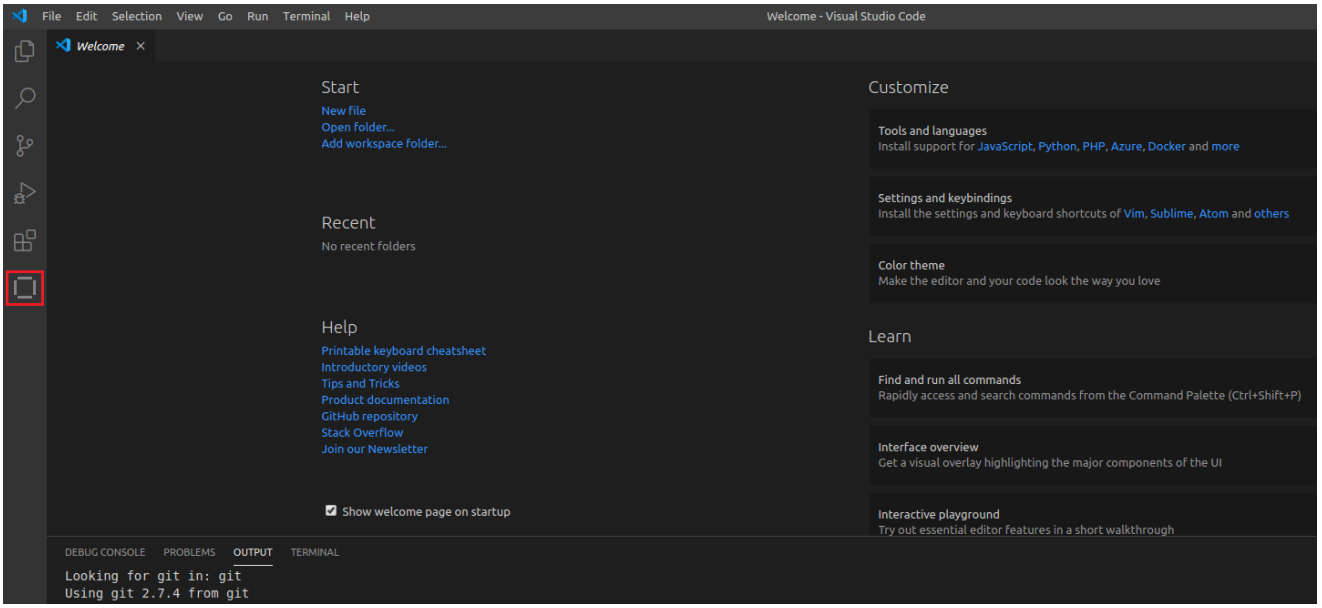


1 Build and Start Fabric for the 'Trade Network'

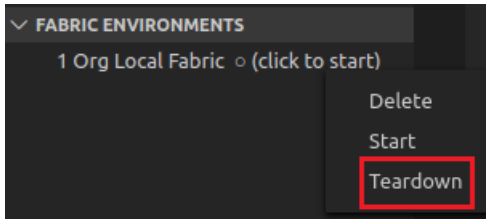
__ **3.** VS Code may already be running from a previous lab exercise, but if not, launch VS Code by clicking on the VS Code icon in the toolbar.



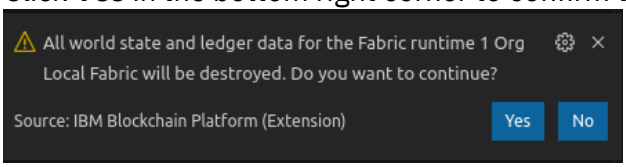
__ **4.** When VS Code opens, click on the IBM Blockchain Platform icon in the Activity Bar in VS Code as shown below.



__ 5. **Right click** on the **1 Org Local Fabric** environment and click the option the **Teardown Fabric Runtime:**



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

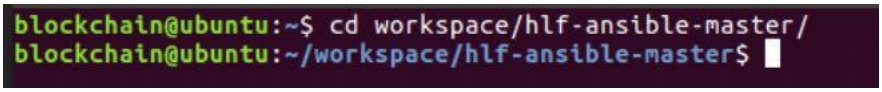


We are now going to start the custom Hyperledger Fabric network. This VM includes a script to automate the process.

__ 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 defines the network, as per the description in the ‘Trade Network’ section earlier.

Briefly examine the playbook in the terminal window.

__ **8.** Type the following command:

```
more site.yml
```

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

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

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

__ **9.** 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 – at the beginning of the output, there are some warnings, which you can ignore).

```
./deploy.sh
```

```
blockchain@ubuntu:~/workspace/hlf-ansible-master$ ./deploy.sh

PLAY [Deploy blockchain infrastructure NO smart contracts] *******

TASK [Gathering Facts] *********************************************
ok: [localhost]
```

Successful completion of this command will look like the following:

```
TASK [ibm.blockchain_platform_manager : Create gateway JSON file] **************
changed: [localhost]

TASK [ibm.blockchain_platform_manager : Create gateway JSON file] **************
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=340  changed=129  unreachable=0    failed=0    skipped=46   rescued=0    ignored=0


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

> ***Note***
>
> If you see a red error ending with “Bind for 0.0.0.0:17054 failed port is already allocated” or similar then the Local Fabric in VS Code is still present - you may not have performed a teardown, as described in Step 5.

__ **10.** 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:            0.0.0.0:17050->17050/tcp, 7050/tcp, 0.0.0.0:17060->17060/tcp
ca.orderer.example.com:         7054/tcp, 0.0.0.0:16054->16054/tcp
peer0.finreg.example.com:       0.0.0.0:19051-19053->19051-19053/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:     0.0.0.0:18051-18053->18051-18053/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:      0.0.0.0:17051-17053->17051-17053/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
blockchain@ubuntu:~/workspace/hlf-ansible-master$
```

The blockchain network for your 3 organisations is now running.

__ **11.** In order to have full access to some configuration files, file protections need to be modified. Run the following command to modify the file protection – if requested for the password enter ***blockchain***

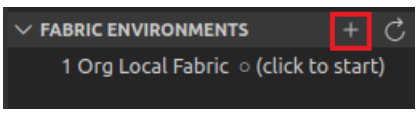
```
sudo chown -R blockchain:blockchain trade/
```

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

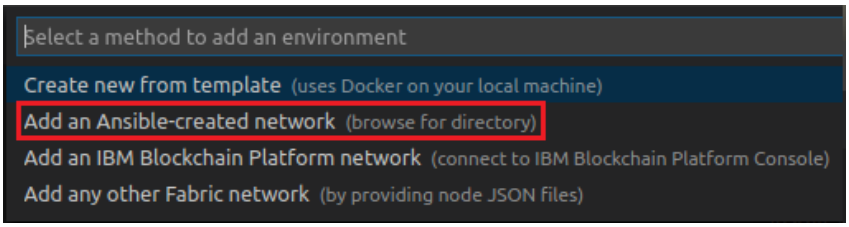
The next step is to import all the node, gateway and wallet JSON metadata files, using a recent feature added to the IBM Blockchain Platform extension (Import an Ansible created network). In one step, you can connect/interact as an organisational identity to execute the private data lab exercises. You simply point at a directory to do the import.

Once imported, the next step is to install the trade exchange smart contract on the three organisations' peers, then instantiate the smart contract on the channel **tradechannel**

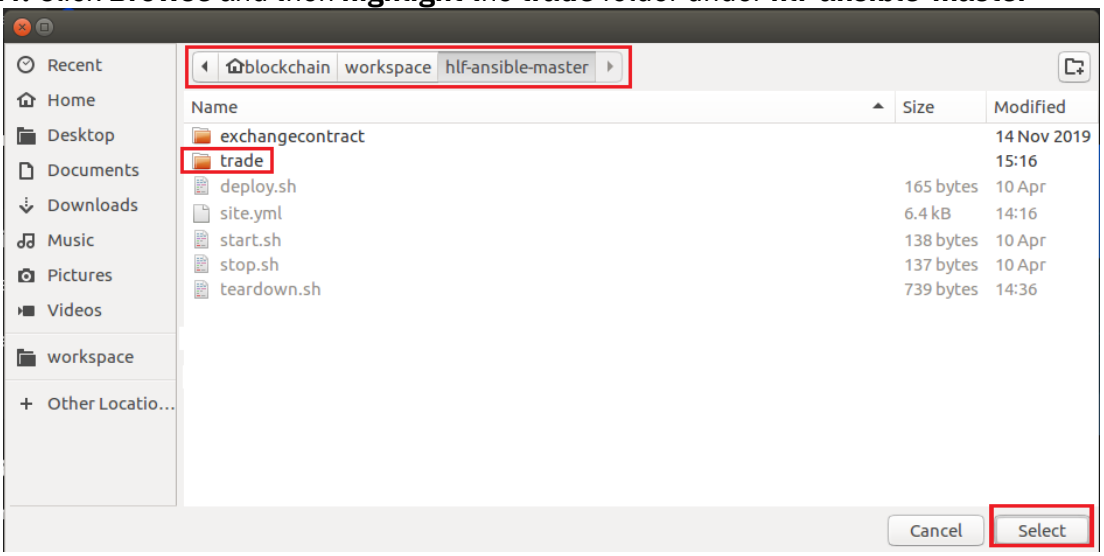
__ **12.** Back in the **IBP VS Code extension**, navigate to the **Fabric Environments** view and click on the '+' icon to add a new environment.



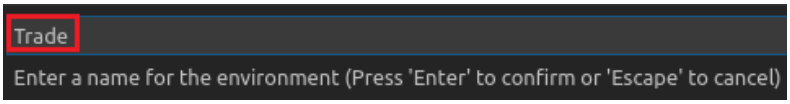
__ **13.** Select to **Add an Ansible-created network** from the list



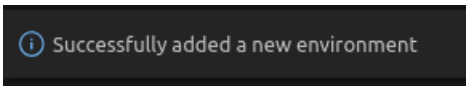
__ **14.** Click **Browse** and then **highlight** the **trade** folder under **hlf-ansible-master**



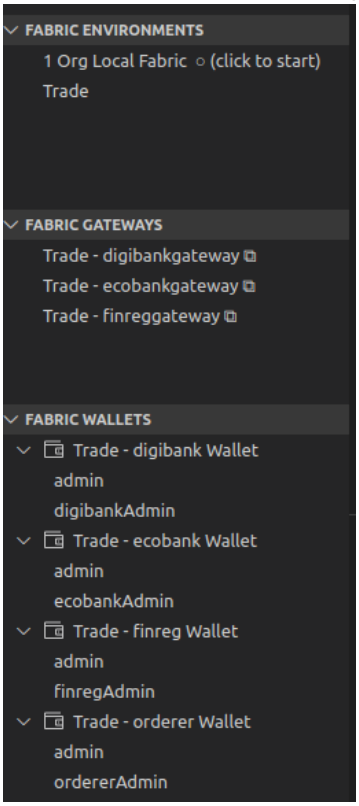
__ **15.** Provide a name of **Trade** for the Fabric Environment and hit **enter**



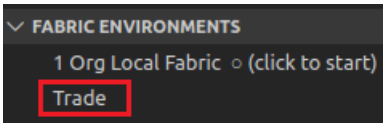
You should get a message (bottom right) it was added successfully, and the left sidebar is populated with a 'Trade' Fabric Environment, Gateways and Wallet names.



__ **16.** Verify that your **Fabric Environment, Gateway and Wallet views** show that the Ansible-created artefacts (for three organisations) are imported.



__ **17.** Connect to the **Trade Fabric Environment** by clicking on it once

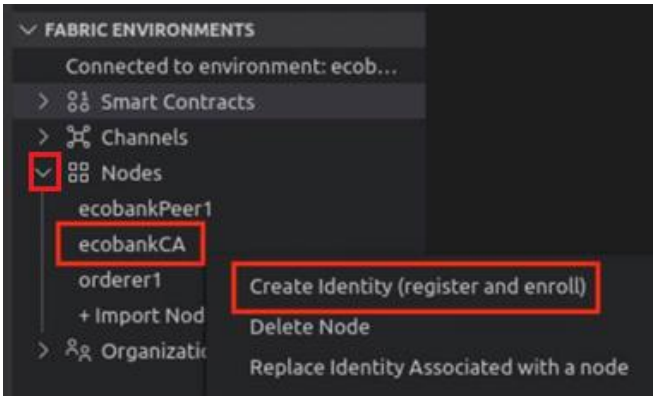


__ **18.** The three Certificate Authority (CA) nodes, under **Fabric Environments** will now be used to create additional (non-administrative) IDs, to allow more realistic testing of the smart contract These IDs are:

Eva for EcoBank organisation
David for DigiBank organisation and
Fran for FinReg organisation

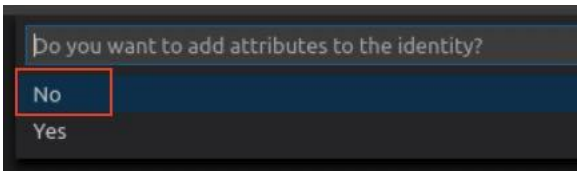
Start by creating the ID for **Eva**.

__ **19.** Expand the **Nodes** twisty, right-click on the **ecobankCA** and then click **Create Identity (register and enroll)**

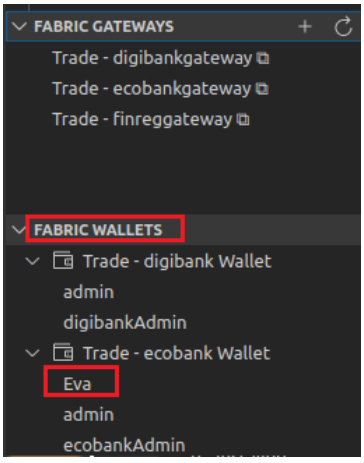


__ **20.** Specify **Eva** as the identity and hit **enter**

__ **21.** There are no attributes required for this lab – click **No**



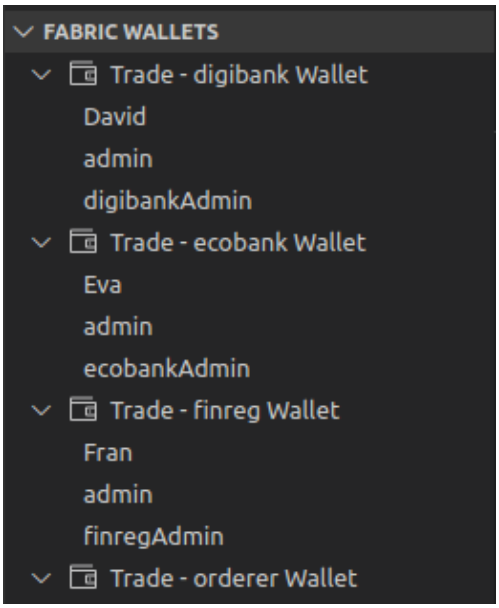
The Identity **Eva** has been successfully added to the **Trade - ecobank** wallet – check the **Fabric Wallets view** for its addition to Trade – ecobank wallet



__ **22.** Using steps *19 - 21* as a guide, connect to the **digibankCA node** under Fabric Environments and create and enroll the identity **David**

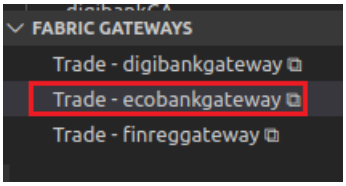
__ **23.** Using steps *19 - 21* as a guide, connect to the **finregCA node** under Fabric Environments and create and enroll the identity **Fran**

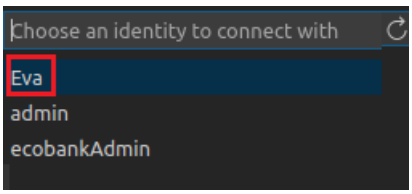
The complete list of wallets and identities for the 3 organisations is shown below



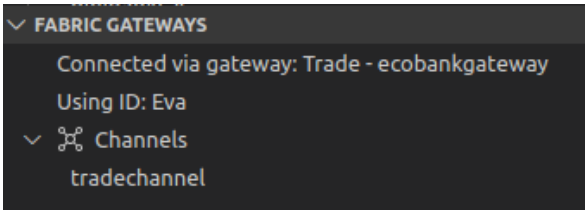
The identities can be now used with the ***Fabric Gateways*** for each organisation.

__ **24. Click** on the new gateway **Trade – ecobankgateway** and choose **Eva** as the identity to connect with

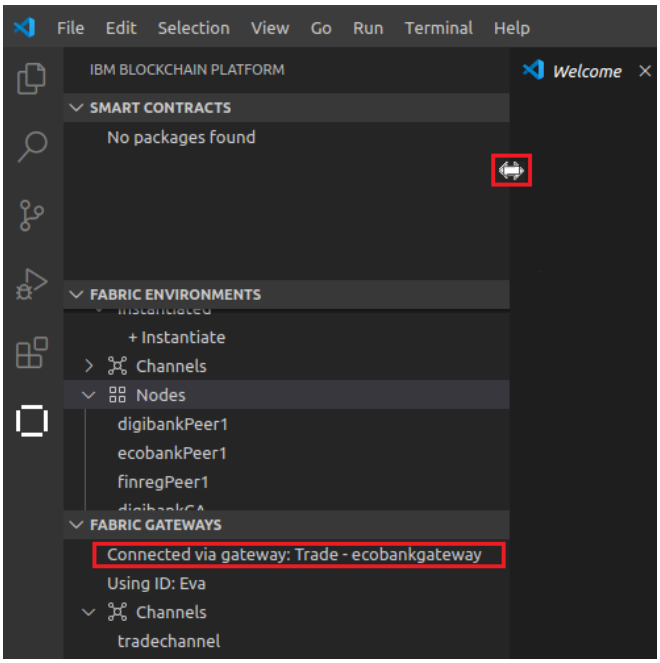




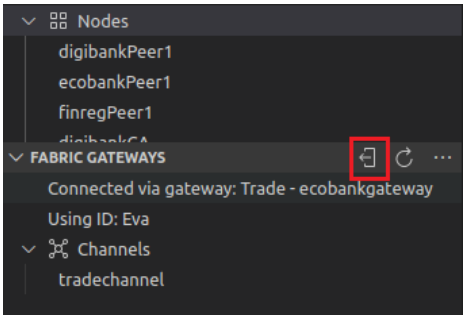
__ **25.** Verify that Eva can see the **tradechannel** channel



__ **26.** Expand the IBP extension sidebar view horizontally, by hovering then **dragging** the pane divider over to the right – ensure you can see the full 'Connected via gateway' message in **Fabric Gateways** view



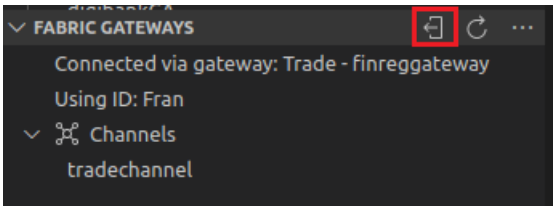
__ **27.** Hover over the Fabric Gateways view and **click** the **Disconnect** icon to disconnect from the **EcoBank** gateway



__ **28.** Using steps *24- 27* as a guide, connect to **Trade – digibankgateway** and verify that David can see the tradechannel channel.

__ **29.** Using steps *24- 27* as a guide, connect to **Trade – finreggateway** and verify that David can see the tradechannel channel.

__ **30.** As the final step in this section, hover over the Fabric Gateways view and **click** the **Disconnect** icon, to disconnect from the **Trade - finreggateway** gateway.



Review

In this section you have:

- Used a playbook (site.yml) to create a Fabric network
- Import Fabric nodes, wallets, gateways and administrative identities
- Tested connectivity from the IBP extension to the running Fabric network.
- Created additional non-administrative users for more realistic private data testing in subsequent sections.

2 Deploy the first version of the Smart Contract

2.1 Introduction

The Fabric network is ready and running, but needs the smart contract containing private data functionality deployed to the network. The process is to package the Smart Contract, install it on organisations' peers as a single step, and finally instantiate it once on the channel.

The smart contract contains code that uses private data features and APIs, 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 shared channel.

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 original transaction, i.e. for state validation or audit/compliance purposes.

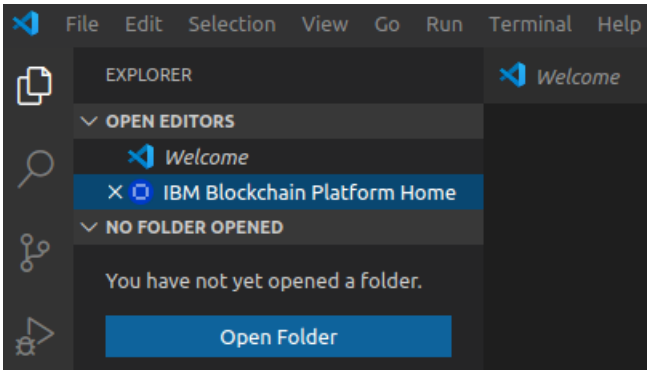
The second is **Private Data collection definition files**. These are policy-based definitions that describe which organisation(s) belong to a Private Data collection. It could be one, two or more organisations (as policy dictates), that govern who can access private data in each collection. It is commonplace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

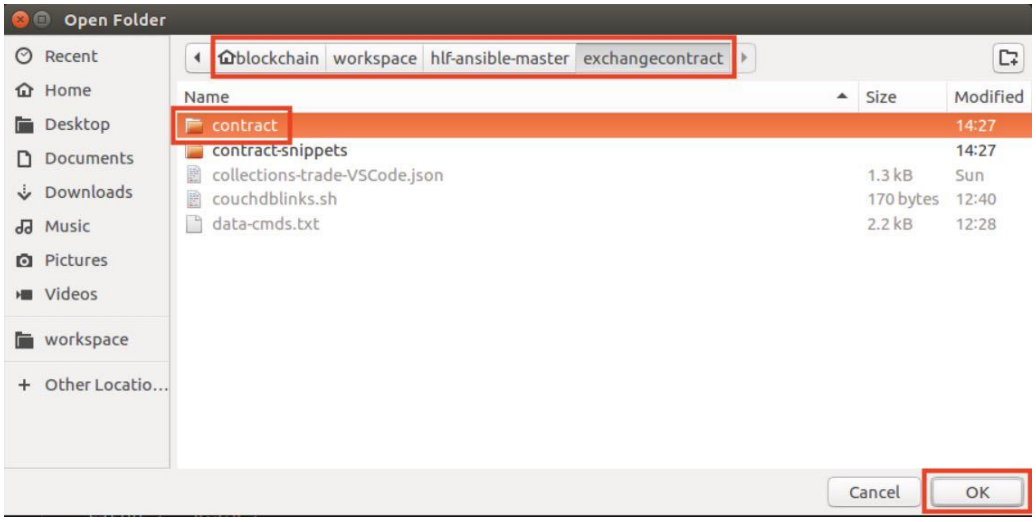
Complete details of the syntax and meaning of all the fields is available in the Fabric Documentation, but note that the two Collections defined in this lab exercise have collection names of **CollectionD** for DigiBank and **CollectionE** for EcoBank. (The "Policy" section in the JSON file is the same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

__ **31.** Click the Explorer icon in VS Code and **Open Folder**

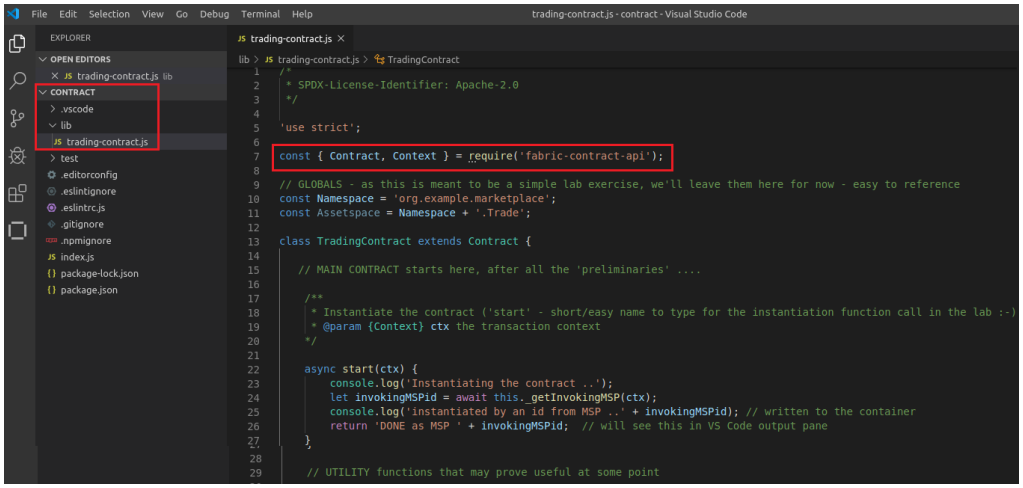


Navigate to the location:
"Home" > workspace > hlf-ansible-master > exchangecontract

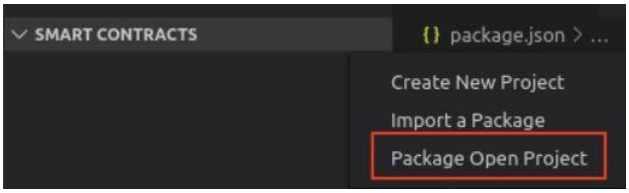
Click to highlight the **contract** folder and click **OK**



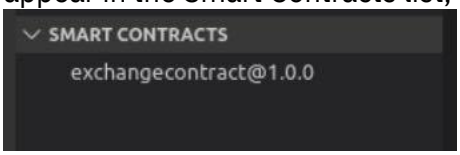
__ **32.** VS Code will display the **Contract folder** tree. Expand the twisty for the **lib** folder and click **trading-contract.js** to view it.



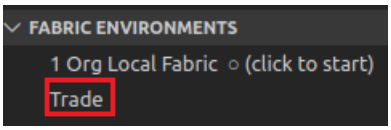
__ **33.** Click the icon  to return to the IBM Blockchain Platform extension, hover over the **Smart Contracts** view, click the Ellipses icon to select “More actions” and then select **Package Open Project**



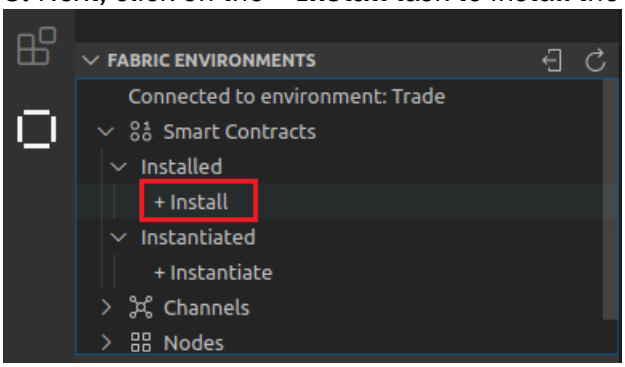
Confirmation that it is package is shown in a message popup, bottom right. This will package the contract into a CDS file that can be installed on a peer. The package will then appear in the Smart Contracts list, as shown.



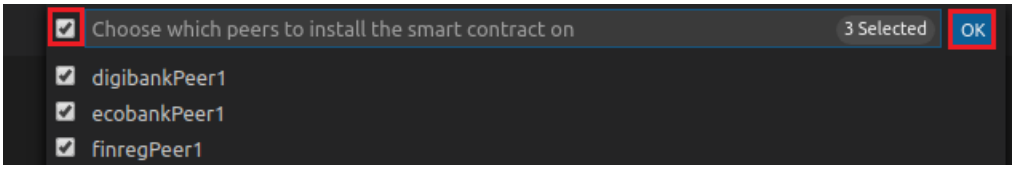
__ **34.** On the **Fabric Environments** view, click on the **Trade** environment to connect to that Fabric environment:



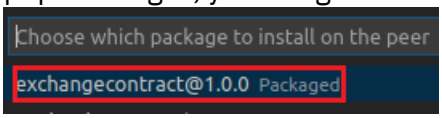
__ **35.** Next, click on the **+ Install** task to install the packaged smart contract:



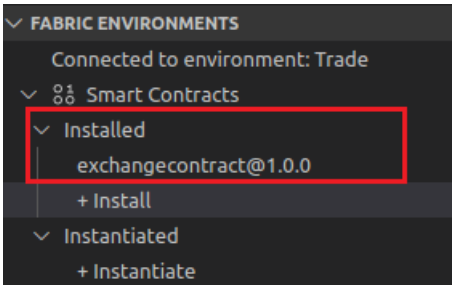
__ **36. Click** on the top left checkbox to select all 3 organisation peers to **install** the smart contract **exchangecontract@1.0.0** and **click OK** to install on all 3 peers in one step.



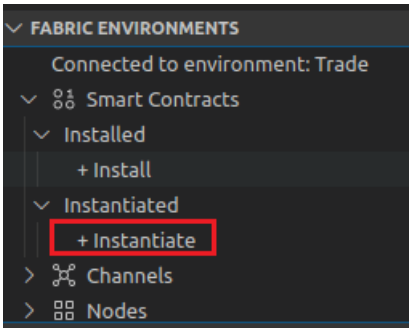
__ **37.** Select the package **exchangecontract@1.0.0** to install from the list – amongst the pop messages, you will get confirmation it was installed on all peers

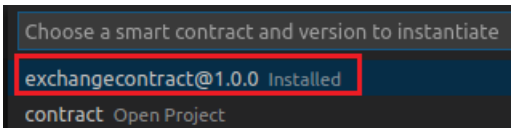


__ **38.** The contract will appear under the installed list:

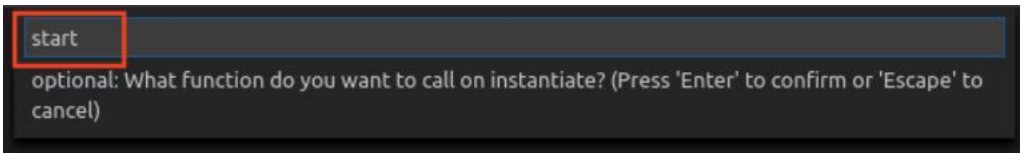


__ **39.** Now click on "**+ Instantiate"** to instantiate the contract on the **tradechannel** channel, and select **exchangecontract@1.0.0** when prompted

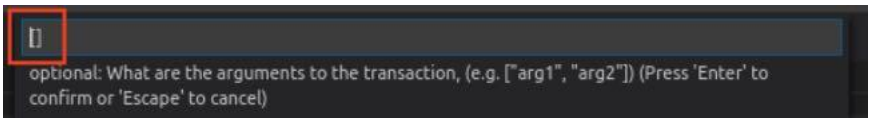




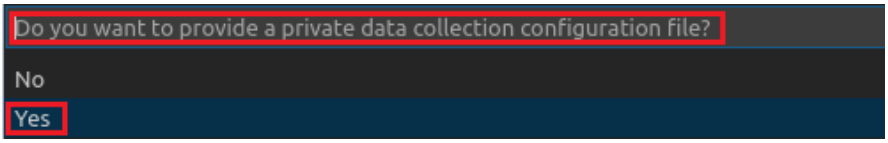
__ **40.** Specify **start** as the function to be used with instantiate



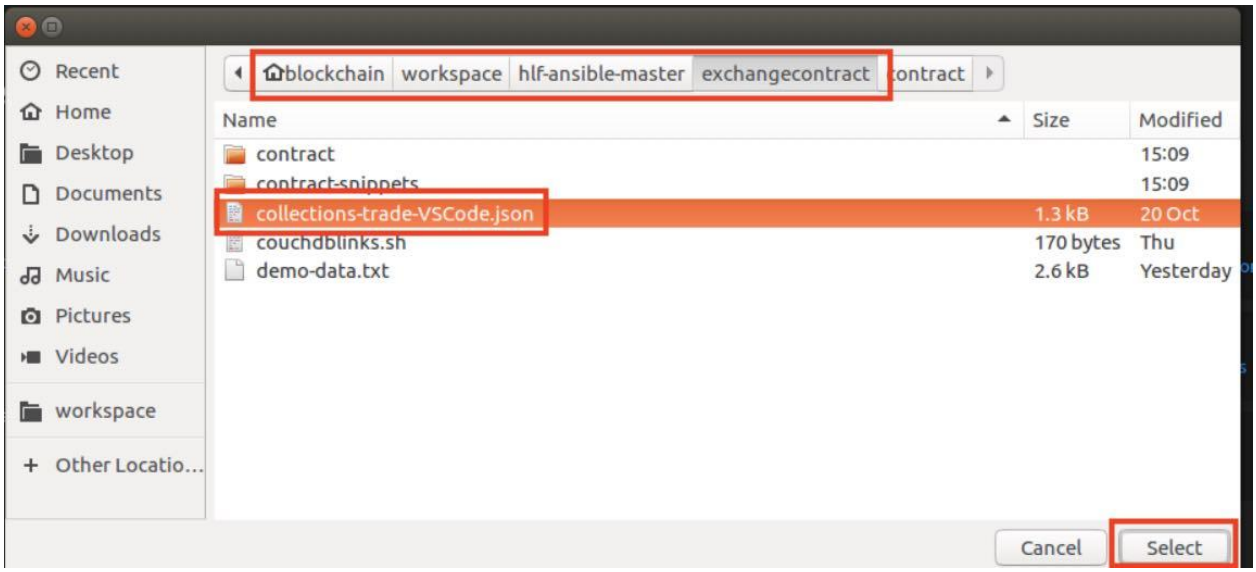
__ **41.** There are no parameters to pass to the `start` function – just press **enter** to accept the default empty array **[]**



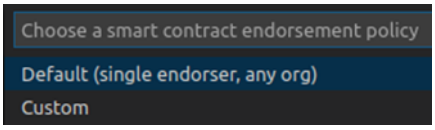
__ **42.** When asked if you 'want to provide a private data collection configuration file?' – select **Yes**



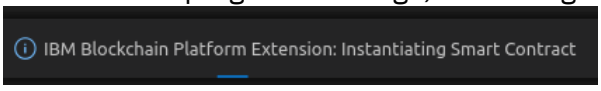
Click **Browse** and navigate to the folder:
"Home" > workspace > hlf-ansible-master > exchangecontract
and select the file **collections-trade-VSCode.json**.



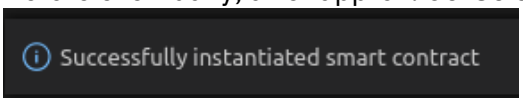
__ **43.** In the next dialogue that asks "Choose a smart contract endorsement policy" choose the default "Default (single endorser, any org)"



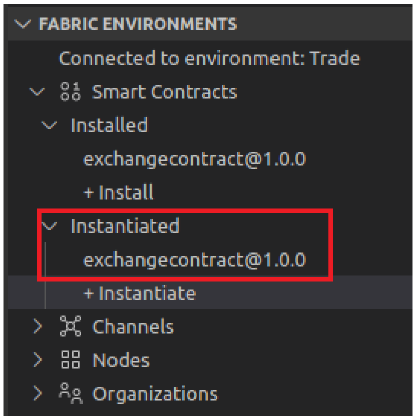
It will show a progress message, bottom right



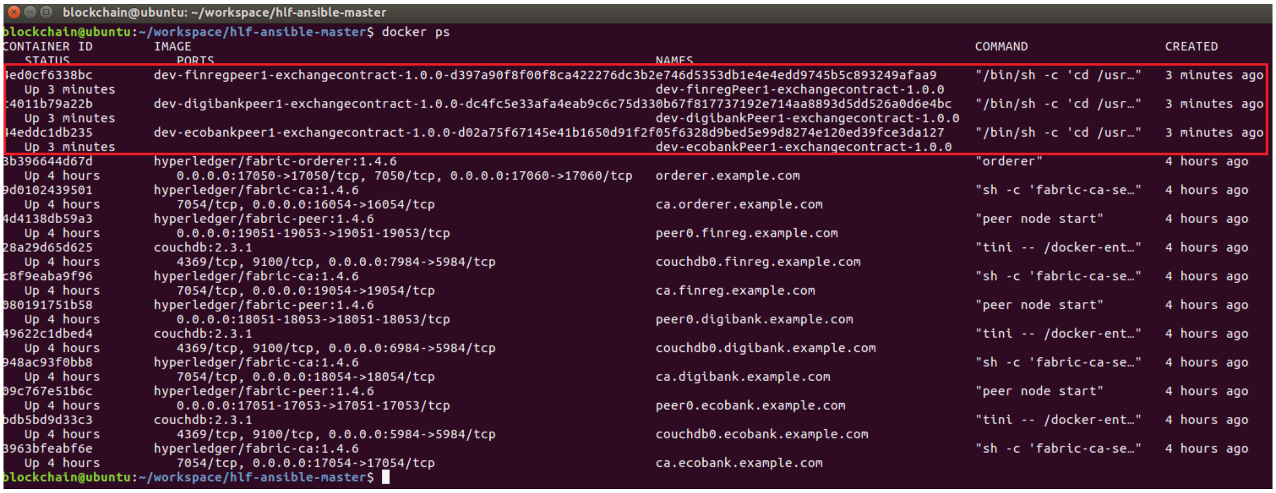
Before eventually, after approx. 60-80 seconds, successfully instantiating



When the contract is instantiated, a popup message will appear (bottom right) and it will appear under the Fabric Environments view under **'Instantiated'** as follows:



The smart contract has been instantiated on the **tradechannel** channel. Note that three organisations' containers are started when listed using `docker ps` from the terminal:



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 channel (starting a chaincode container)
- Instantiated the Smart Contract with the definition of the Private Data Collections

3 Private Data for the Offer and Accept Transactions

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 – this is the way to market a 'commodities contract' for sale on the blockchain network. (The commodities contract is a saleable asset - eg. value $5m - in this business network marketplace; and not to be confused with 'smart contract').

You will 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 transactions 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' by the client - in the VS Code extension you're prompted when you submit a transaction. Transient data is not persisted in the transaction that is stored on the channel ledger; this keeps the data confidential. Transient ('short-lived') data is passed as binary data to a smart contract transaction and then decoded/written in the function.

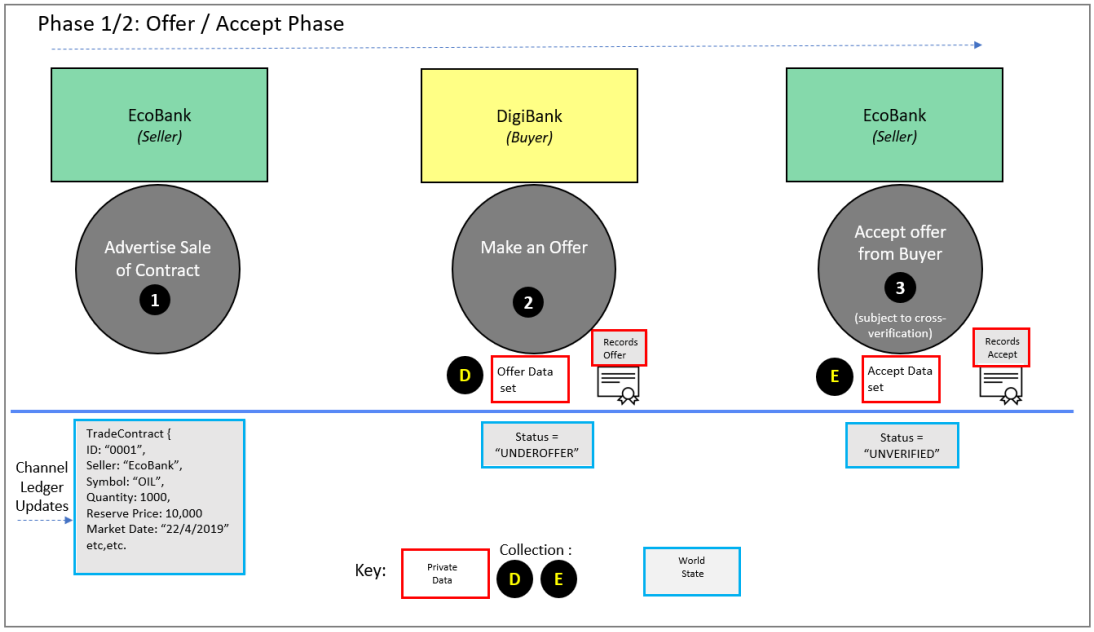
Note: It is common practice to 'salt' the transient data with a random element, as the hash of 'predictable' private data patterns could be matched via brute force. See the appendix 'Access Control considerations for Private data"' for more information.

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

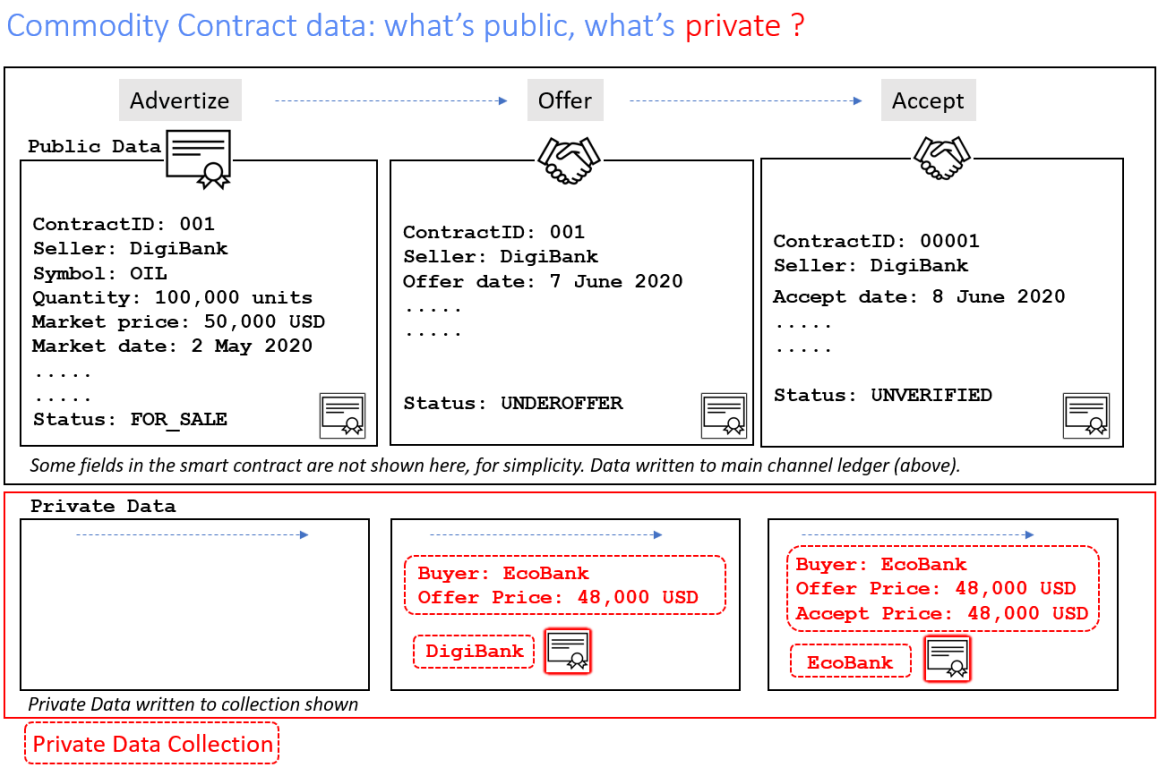
1. Seller **EcoBank** *advertizes* the nominal sale price / info on the marketplace and it is recorded on the blockchain - this is written to the channel-wide ledger and all would-be buyers get notified (via applications, out-of-band).

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:



__ **44.** During the lab exercise, you will see both private data and channel ledger updates generated, as transactions get invoked. From a labs execution standpoint, its also 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. Let's first open these CouchDB views.

3.2 Invoking the Offer and Accept transactions

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 long - and thus are more easily and efficiently sourced from a text file. We'll open these views and commands file, as a preparatory step.

__ **45.** 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$
```

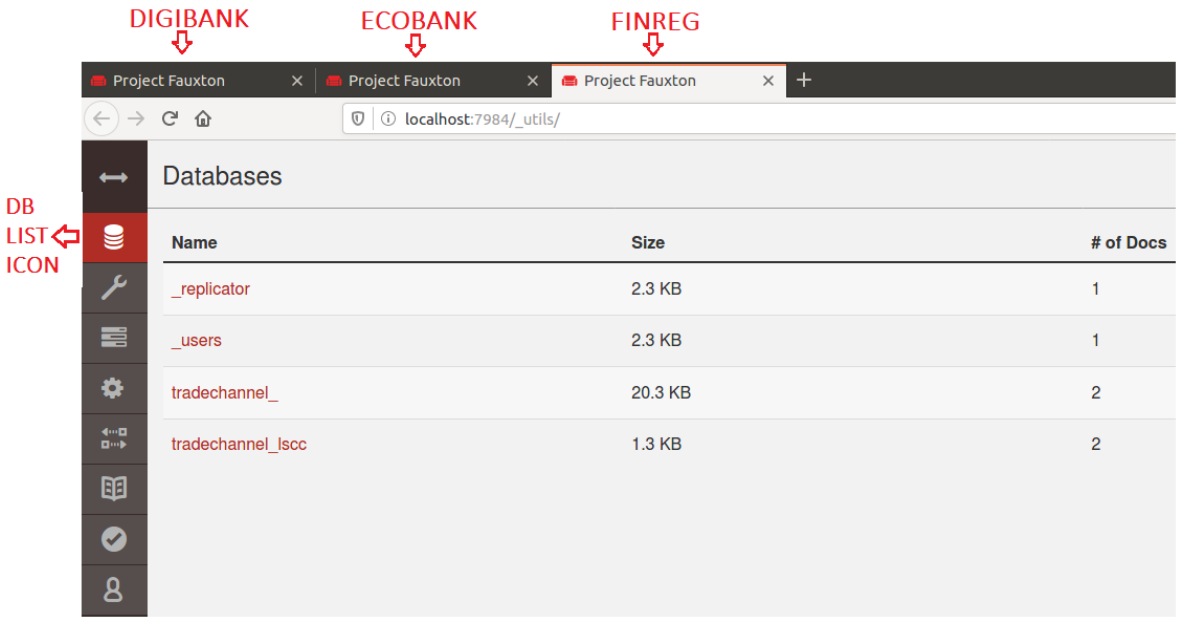
When the script is complete you should eventually see a Firefox browser window with **three tabs** displayed. (**Note:** that the script may take a couple of minutes to complete, and there will be a pause between tabs opening).

Note: If the tabs have not opened inside a minute, 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.

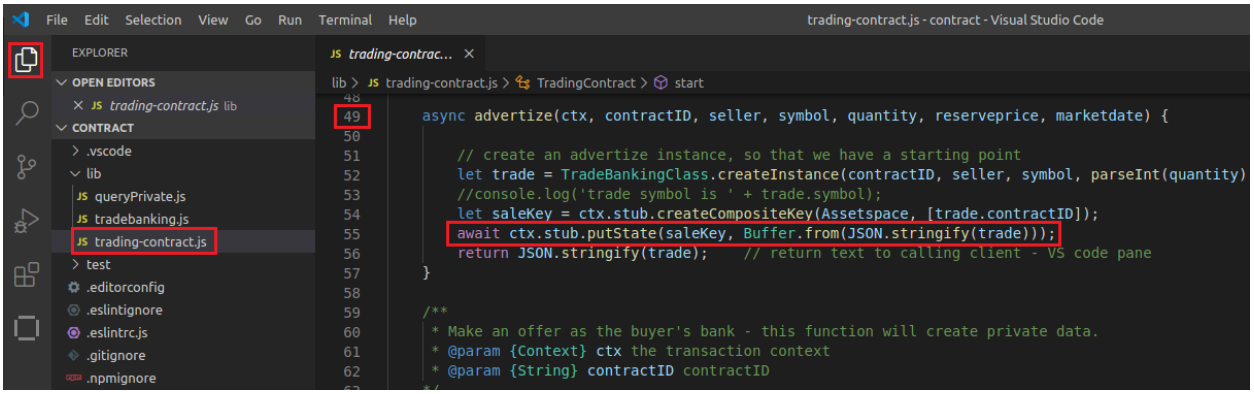


3.3 Review and Perform the 'advertize' transaction

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

__ **46.** In VS Code, click on the **Explorer** icon

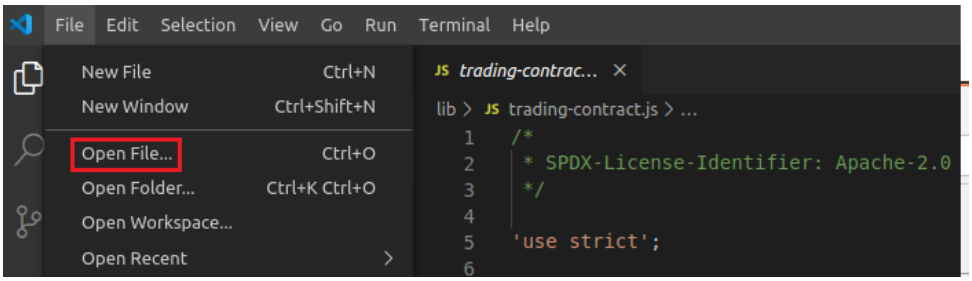
Click on the file **trading-contract.js** under the **lib** directory in Explorer and locate the function beginning with **async advertize** function at **line 49**.



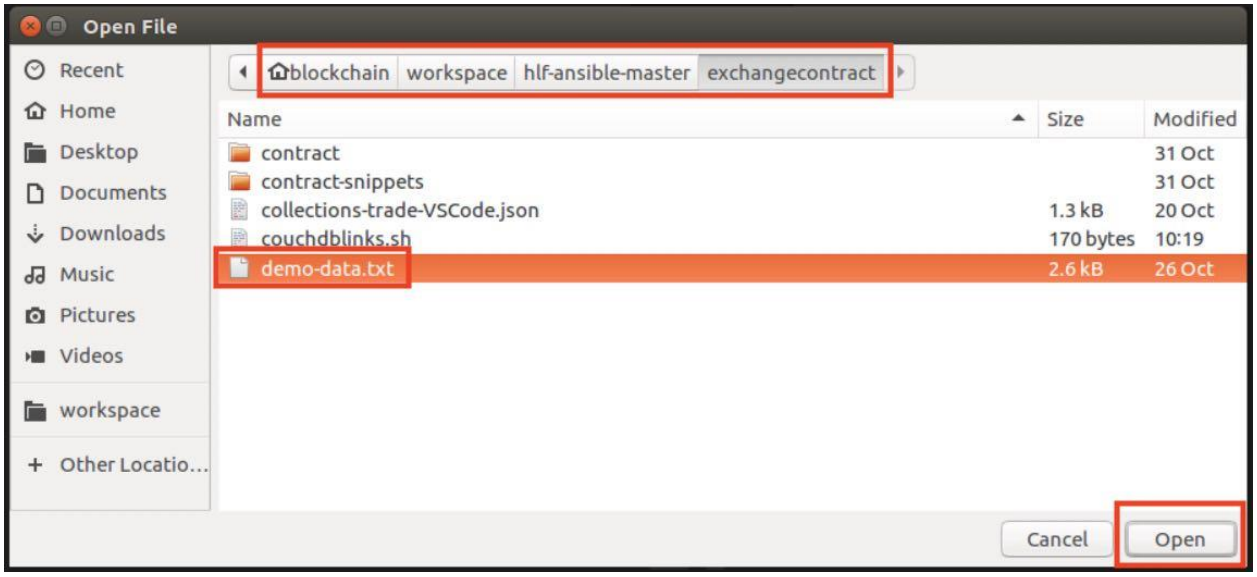
The **advertize** transaction creates a saleable Commodity contract (identified by a 'contract ID') on the channel ledger. For information, it uses the Fabric **putState API** to write non-private data to the ledger (no private data created just yet). This transaction function is called by someone from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

As mentioned, the lab also provides you with a helper file called **demo-data.txt** - from which you can copy and paste parameters, and command lines. It's located under the **workspace/hlf-ansible-master/exchangecontract** folder. We will now open it in VS Code.

__ **47.** In the VS Code editor, click on the menu option File > **Open File**

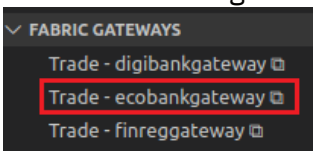


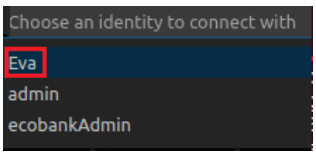
__ **48.** Browse to the folder:
"Home" > workspace > hlf-ansible-master > exchangecontract
and select the file **demo-data.txt**



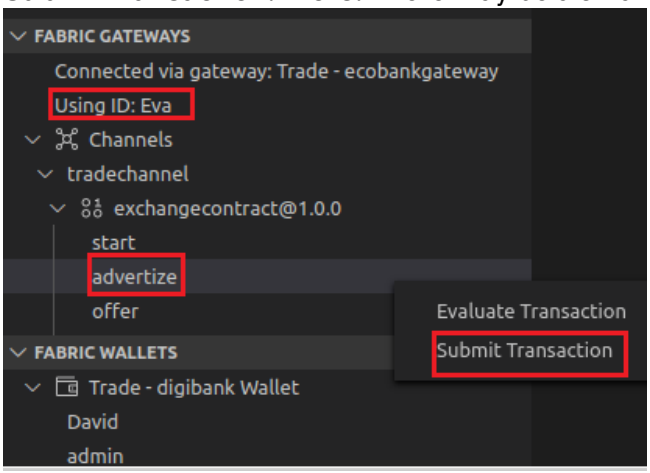
You can copy and paste from this text file in the background in the steps that follow.

__ **49. Switch back** to the IBM Blockchain Platform VS Code extension **by clicking the icon**, and in **the Fabric Gateways view** click the **Trade - ecobankgateway** gateway and connect to it using the identity **Eva**



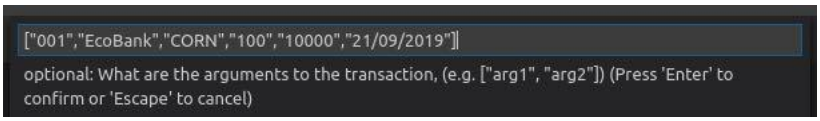


__ **50.** Expand Channels and the channel **tradechannel** - then expand the **exchangecontract@1.0.0** twisty, then right click on the **advertize** transaction and click **Submit transaction**. **Note**: There may be a small delay when expanding the contract.



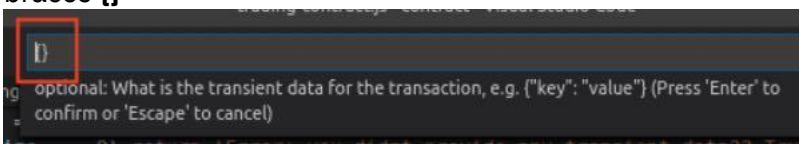
__ **51.** Next, enter the following parameters for arguments when prompted – popup with the parameter list copied from **demo-data.txt**

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

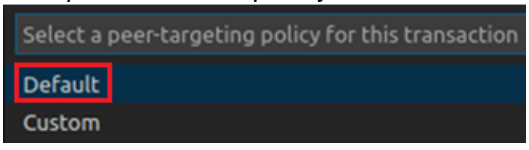


As you may recall from the advertise function in the contract, the parameters represent the **contract ID**, **seller**, **symbol**, **quantity**, **reserve price** and **market date** respectively.

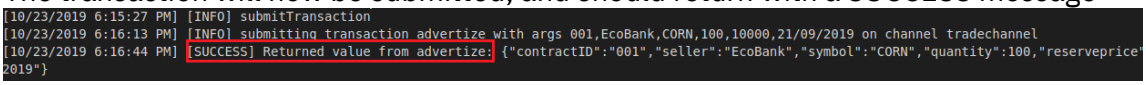
__ **52.** There is no transient data for this function, just press **enter** leaving the empty curly braces **{}**



__ **53.** When asked to "Select a peer targeting policy for this transaction" just press enter to accept the Default policy



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



3.4 Review and Perform the 'offer' transaction

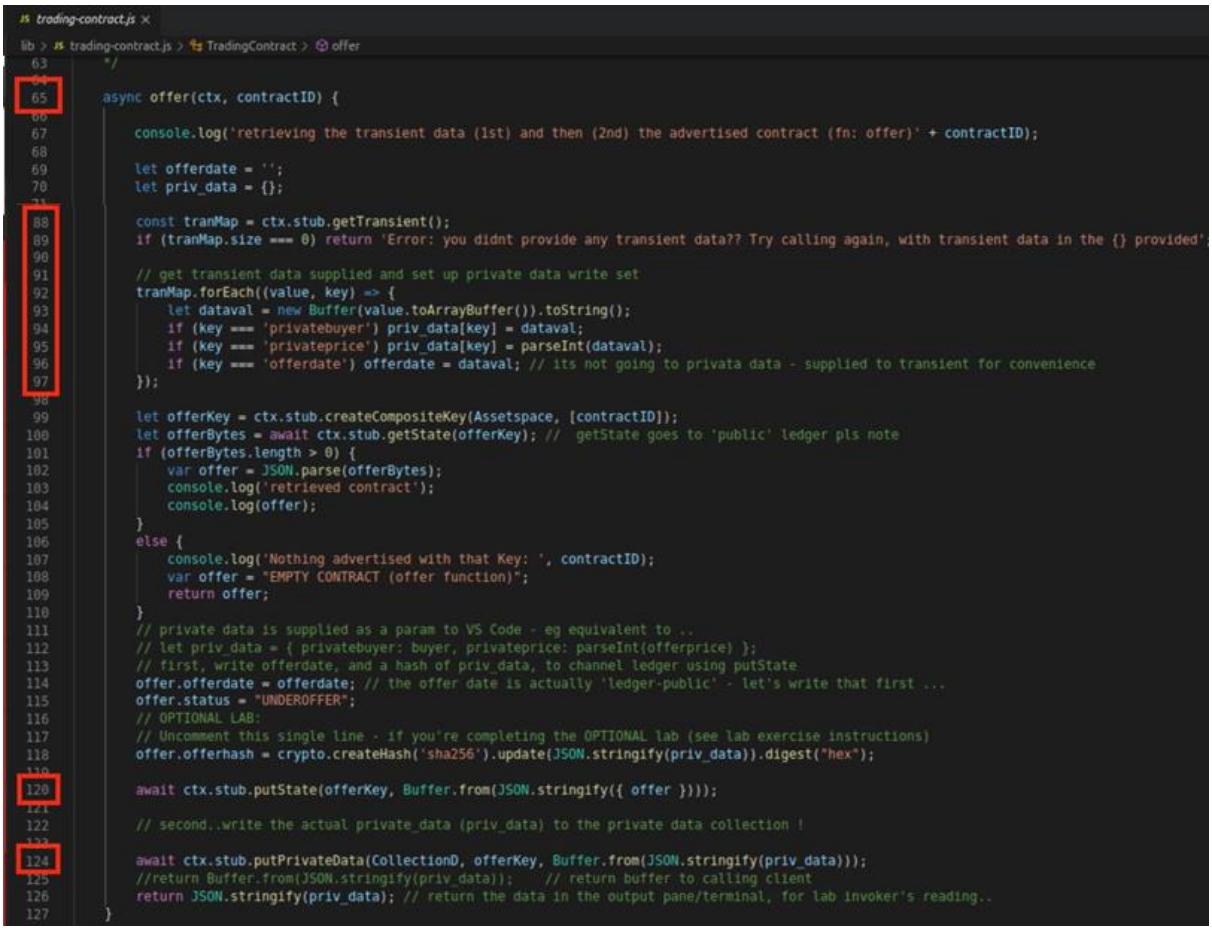
In this part of the lab, you will perform the offer transaction as DigiBank.

__ **54.** Back 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 – such data is supplied when invoked from the VS Code extension or using a client app via the Fabric SDK. FYI, transient data is passed as binary data and then decoded in the function using the Fabric API **getTransient** method.

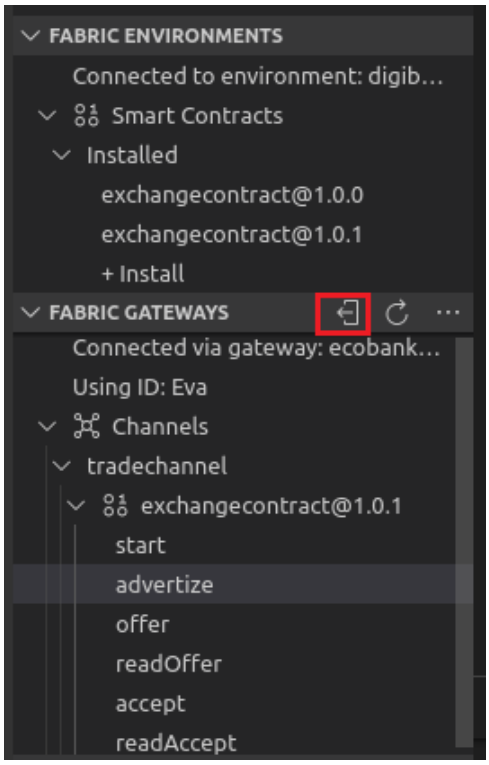
Lines 88-97 show the private data elements being committed to DigiBank's private collection using the Fabric API method **PutPrivateData** in **line 124**

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

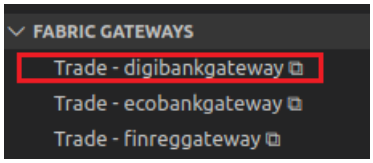


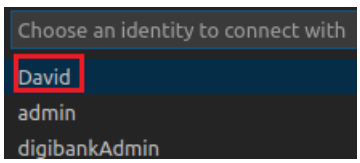
Now let's invoke an **offer** transaction for the advertised commodity contract (with Contract ID `"001"`) – then later, verify the private data was added, using a **readAccept** smart contract transaction.

__ **55.** The **offer** transaction needs to be done **as a DigiBank trader** – first, we **must** disconnect from the **EcoBank** gateway – click on the **disconnect** icon

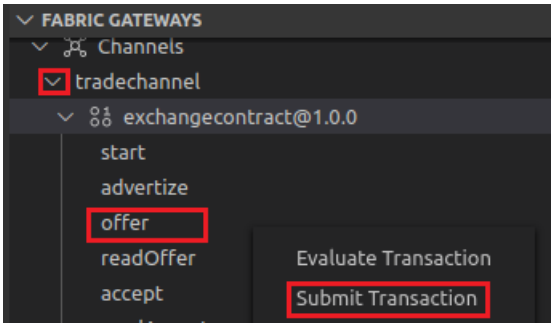


__ **56.** Next, click on the DigiBank gateway **Trade - digibankgateway** and select **David** as the identity to connect with.



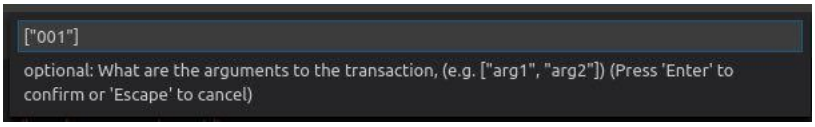


__ **57.** Right-click on the **offer** transaction and click **Submit Transaction**



__ **58.** When prompted for arguments, enter the following parameter (overwriting any existing '[]' data) – and press **enter**

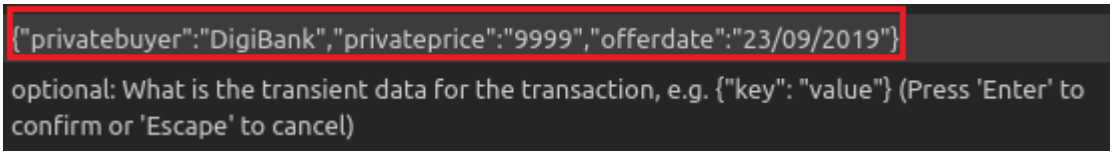
["001"]



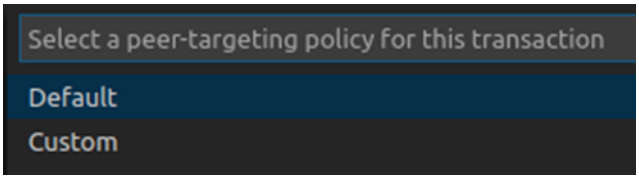
__ **59.** Next, you will be prompted to provide 'transient data'. Our private offer is confidential and so we want to use this 'transient' field to supply this information so that it is not captured on the ledger. Copy the entire parameter list below – and/or paste from 'demo-data.txt' commands file - into the parameter list, overwriting the existing '{}'.

Note: that the data is wrapped in curly brackets this time – this is required when passing transient data! Also ensure there are no trailing spaces.

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



__ **60.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy:



__ **61.** 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 – obviously you would not do this in a live application!

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

__ **62.** Still in the VS Code Explorer, review the additional helper function **readOffer** (line 134) in the contract. The **readOffer** transaction enables you to verify what was written to private data earlier by the offer transaction:

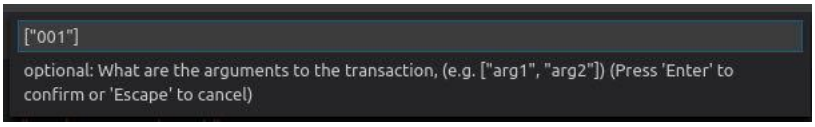
```
130     * Read the offer - can be done by someone from the buyer's bank - this function will read private data from the collection
131     * @param {Context} ctx the transaction context
132     * @param {String} contractID contractID
133     */
134    async readOffer(ctx, contractID) {
135
136        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**.

__ **63.** Right-click on transaction **readOffer** and click **Evaluate Transaction**

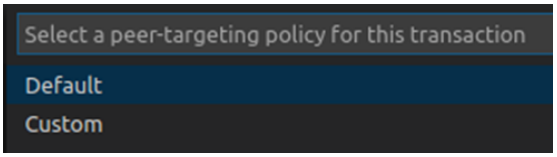
__ **64.** When prompted for parameters, enter the following:

["001"]



__ **65.** There is no transient data for this function, just press **enter** leaving the empty curly braces **{}**

__ **66.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy:



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
```

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

To consolidate your learning, you will check out the 'private data footprints' created in CouchDB directly, to get some context. Earlier you launched three Firefox tabbed sessions each containing a CouchDB view and a set of additional databases - you can click through each tab, to explore the individual records we've created, following the **advertize**/**offer** transactions that created channel-wide and private respectively.

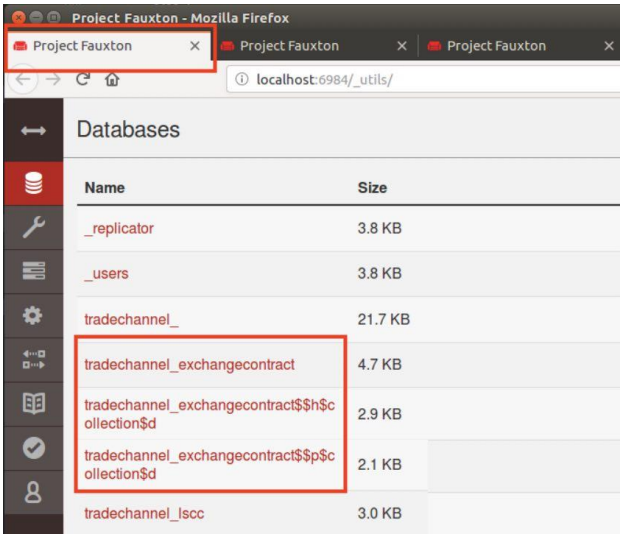
__ **67.** Switch to the **Firefox** window in your VM

__ **68.** 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.

__ **69.** Review the databases created following the 'offer' transaction unit of work:
tradechannel_exchangecontract is the world state database for the exchangecontract on the tradechannel

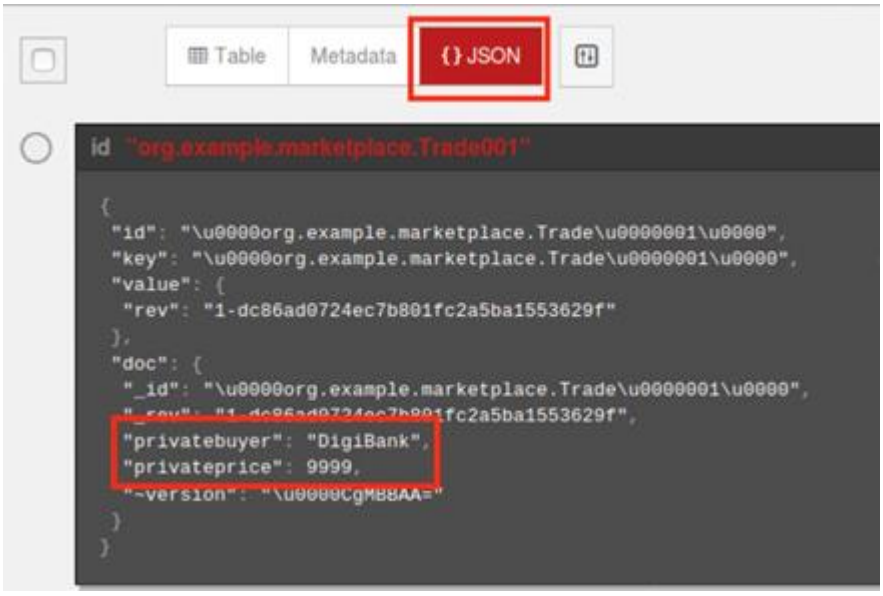
tradechannel_exchangecontract$$h$collection$d is the private data collection **hashstore** for 'CollectionD'

tradechannel_exchangecontract$$p$collection$d is the private data collection for 'CollectionD' (which is only available in DigiBank's CouchDB)

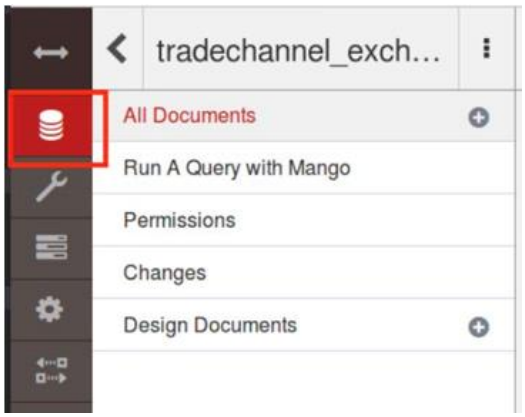


__ **70.** Click on the private collection tradechannel_exchangecontract$$p$collection$d

__ **71.** 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".

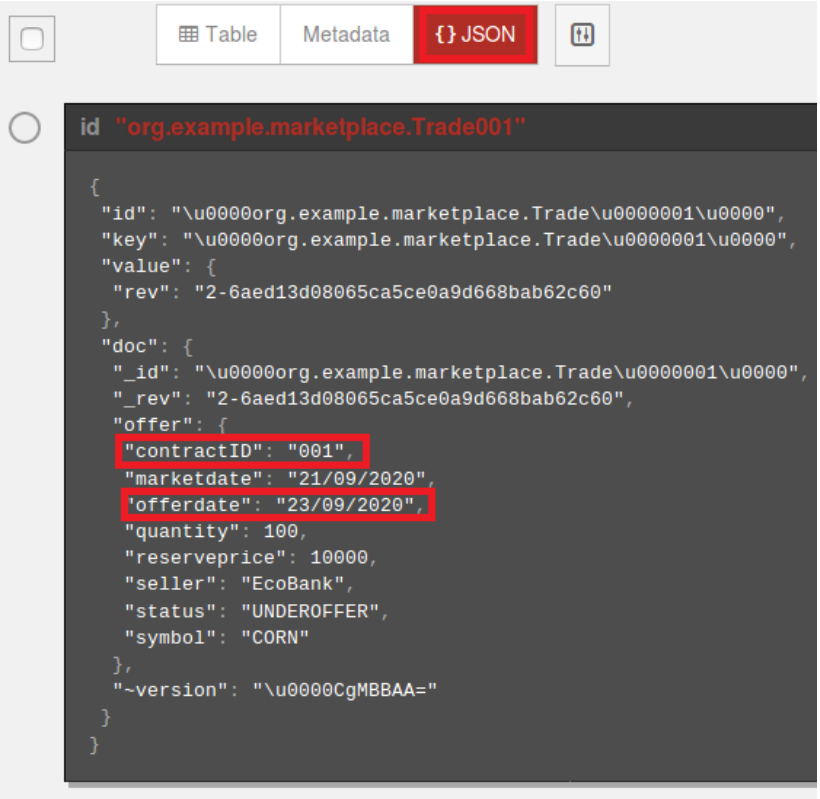


When you have reviewed the data, click the **All DBs** icon to return.



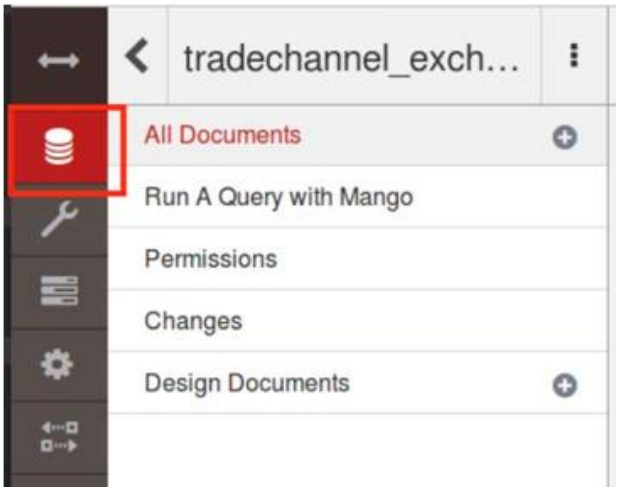
__ **72.** Back in the main database list – click on the world state database called **tradechannel_exchangecontract**

__ **73.** 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

__ **74.** When you have reviewed the data, click the **All DBs** icon to return.



__ **75.** Still using the **first tab** in Firefox (the **DigiBank** CouchDB), open the collection hash database **tradechannel_exchangecontract$$h$collection$d** and verify that the key and value is unreadable.

__ **76.** Using the **second** tab for **EcoBank's** CouchDB, look at the database list and notice that EcoBank does not have the private database called **tradechannel_exchangecontract$$p$collection$d** but note that it does have a copy of the collection hash database **tradechannel_exchangecontract$$h$collection$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.

__ **77.** In **trading-contract.js** in **VS Code Explorer**, scroll to the implementation of the **accept** transaction function at **Line 157** and examine it.

You will see that the **accept** transaction has similar private data logic to the **offer** transaction, 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 **readAccept** transaction 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.

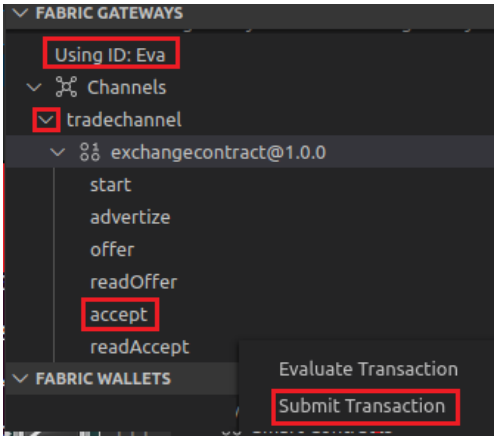
Now let's perform the actions of the **accept** transaction function – after, we will verify the private data was added using the **readAccept** transaction.

__ **78.** Return to the VS Code editor and the **Fabric Gateways** view, click the Disconnect icon to disconnect from DigiBank's **Trade – digibankgateway** gateway.

__ **79.** In the Fabric Gateways view, connect to **Trade- ecobankgateway** and choose **Eva** as the identity – we will perform the **accept transaction** as the **Ecobank** organisation

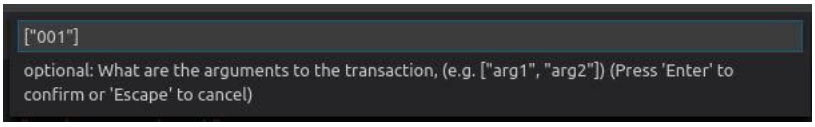
__ **80.** Expand the transaction list as before, and **right click** on the **accept** transaction and click **Submit Transaction.**

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



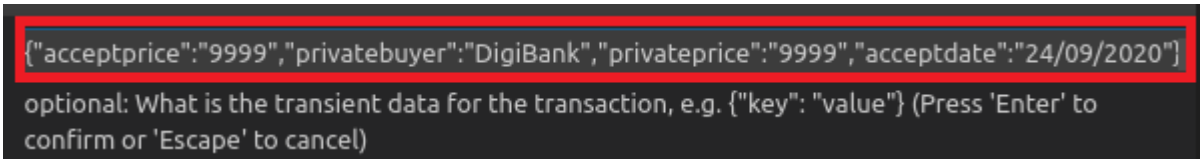
When prompted to enter arguments enter the following **Contract ID** parameter:

["001"]

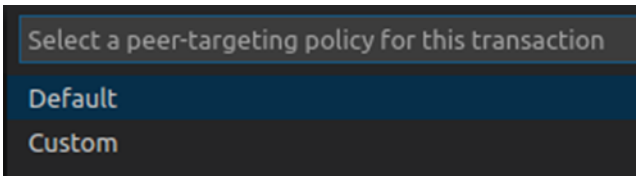


__ **81.** Enter or paste in from the file 'demo-data.txt' (still open in VS Code) the following transient data for this transaction, replacing the '{}' curly brackets offered, with the following one-liner and hit **enter**:

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

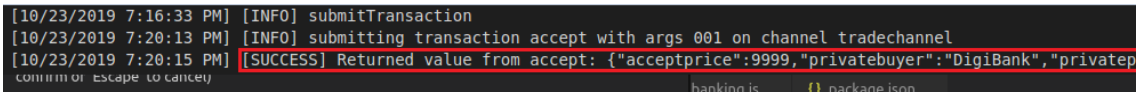


__ **82.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy:



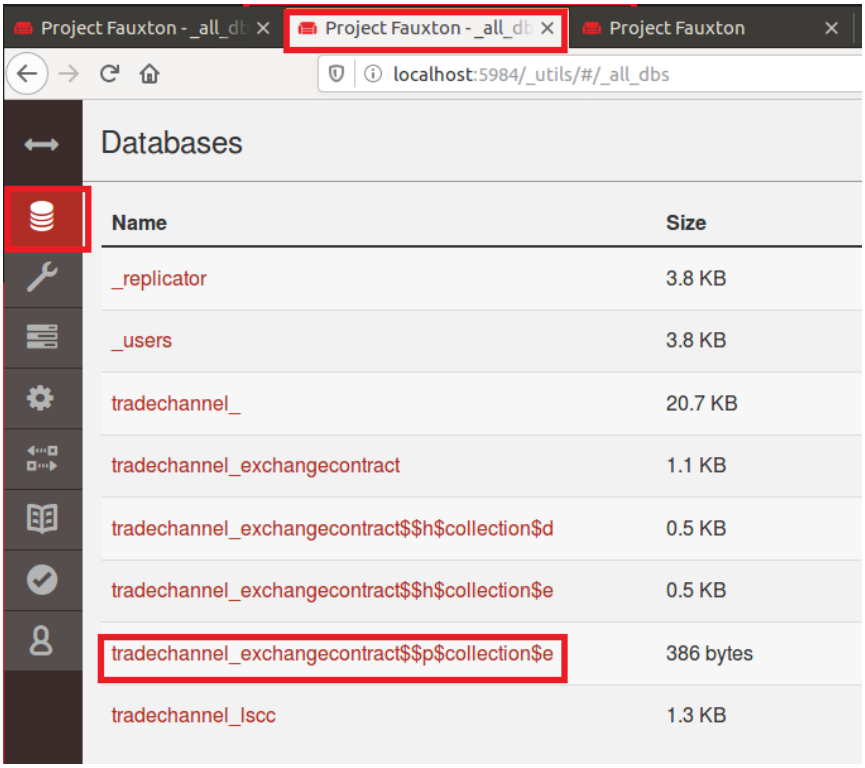
The transaction is now submitted.

__ **83.** Review the output pane at the bottom for results – you should see a SUCCESS message like this one containing the private data being written:



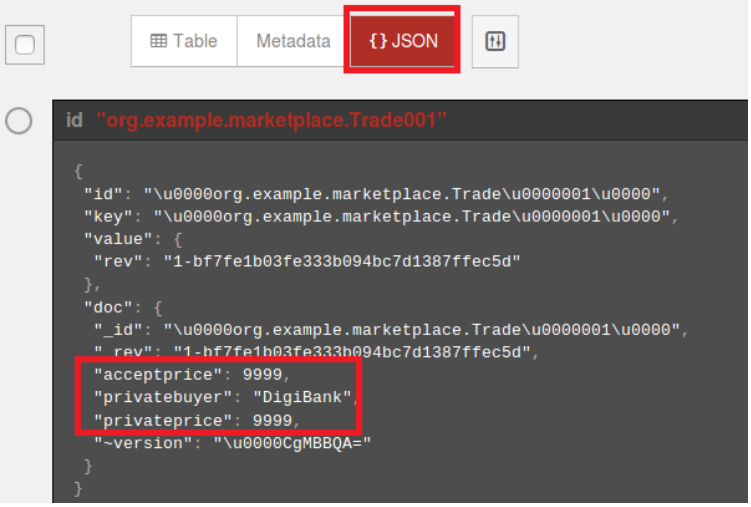
Notice that the **'acceptdate'** value is not part of this SUCCESS message. This is because 'acceptdate' is not private,and written to the channel-wide ledger – the output only reflects the private data written by the function, to the collection.

__ **84.** Return to Firefox to review activity in CouchDB on the **second tab** for the **EcoBank CouchDB,** running on **Port 5984** – Refresh the Firefox Tab and click the **All DBs** icon (on left) – click on the database for the private collection CollectionE called **tradechannel_exchangecontract$$p$collection$e**

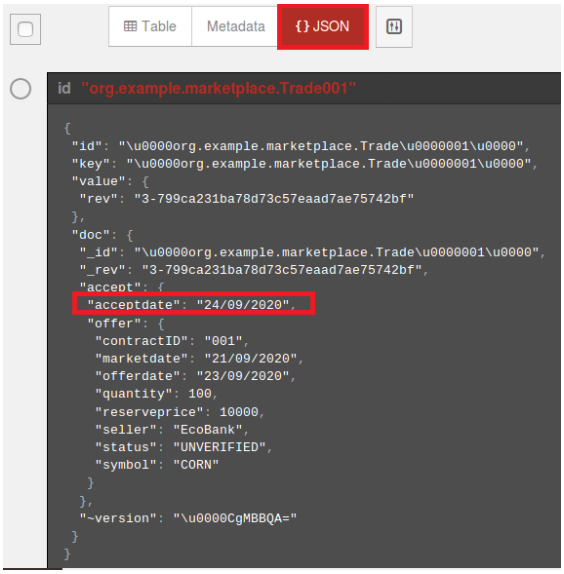


__ **85.** Click on the **{} JSON** format icon alongside

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



__ **86.** 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 Identity** - which of course, should be denied - proving that the data is confidential to EcoBank.

__ **87.** Return to VS Code and in the Fabric Gateways view click the **Disconnect** icon to disconnect from the current gateway.

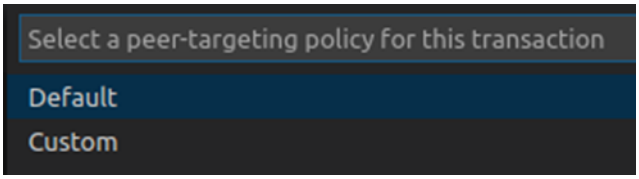
__ **88.** Connect to **Trade – digibankgateway** gateway with the identity **David.**

__ **89.** Right-click on the **readAccept** transaction and click **Evaluate Transaction.** When prompted, enter the following in the parameters in [ ] brackets:

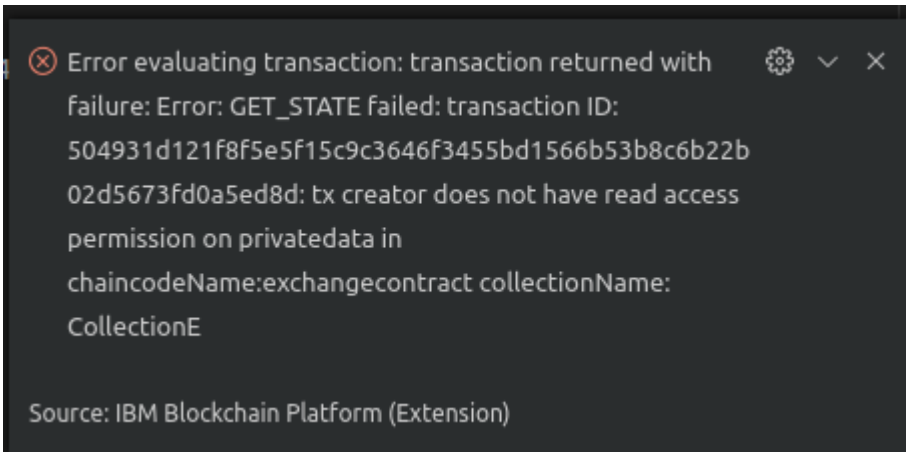
["001"]

__ **90.** There is no transient data for this function, just press **enter** leaving the empty curly braces **{}**

__ **91.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy



__ **92.** The evaluate should return an error, shown in the informational message on the bottom right in VS Code. This is because the DigiBank identity used to invoke the transaction cannot access the private collection.



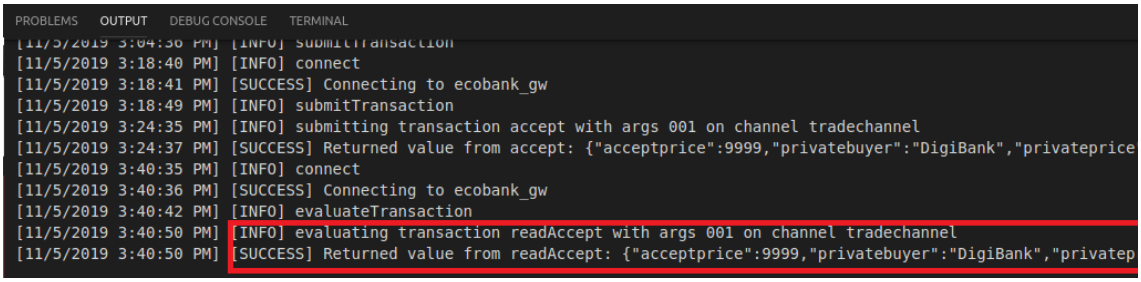
__ **93.** Now **disconnect** from the **DigiBank Gateway** in the VS Code extension.

__ **94. Connect** to EcoBank's Gateway as **Eva**

__ **95.** Highlight the transaction **readAccept** and do a right-click '**Evaluate Transaction**' and at the prompt, replace the existing '[]' string entirely with the following:

["001"]

__ **96.** Review in the **output pane** that you can now read the accept private data as **Eva**



__ **97.** As a final step, **disconnect** from **Trade - 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 have used the offer/accept functions to create private data in different private data collections
- You have seen, through CouchDB views, that the transactions also create a private data hash store (containing the public hash) of a corresponding private data collection.
- You've verified 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

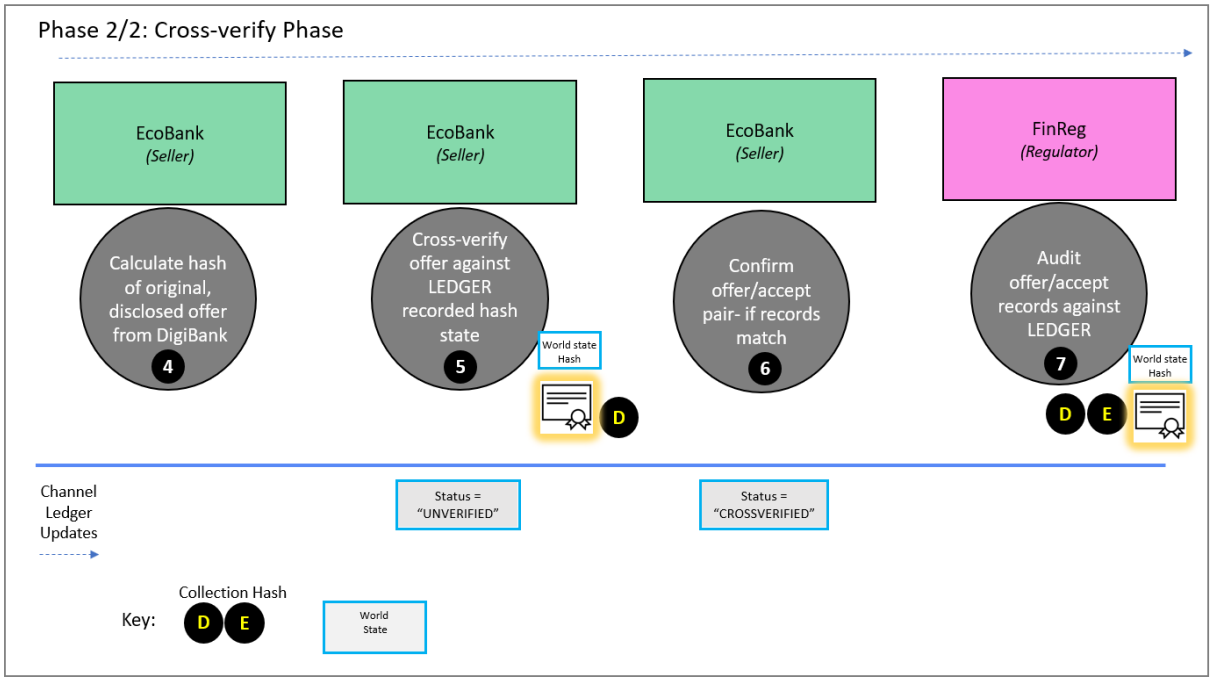
4.1 Introduction

In this section we will upgrade our smart contract to include cross-verification functionality. This will enable the smart contract to verify what's written as private data, cryptographically matches data provided to a verifier 'out-of-band' (e.g. seller, regulator). If they match, for a particular commodities contract asset, the channel ledger record gets a status of 'CROSSVERIFIED'.
An identity from a proving organisation (for example, the seller) submits this cross-verify function which checks whether the two hashes match, returning a MATCH / NO MATCH response to the verifier.
It's important to note that the verifying party only has access to the channel-wide private data hash store, and a set of data it asks to verify – it does not have access to the private data collection containing the private data.

Here is an outline of the cross-verify process:

1. **EcoBank** calculates a hash of the accepted offer data set, that they originally agreed to
2. **EcoBank** cross-verifies this calculated hash against the channel-wide ledger hash of the original offer for the contract ID in question – it was recorded on that ledger, when the original private data transaction was executed.
3. **EcoBank** cross-verifies they match.
4. If there is a match, the offer/accept pair is verified and the channel ledger record (for the commodities contract record) has its status set to 'CROSSVERIFIED' - At this point, the exchange can be completed, and the resultant sale transfer can progress.
5. Later, the regulator **FinReg** wishes to cross-check the original offer and accept transactions recorded to each organisation's private data store, against the private data hash for this, stored in the channel-wide 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 the Smart Contract

The next phase is to modify the smart contract source to add the cross-verification transactions. Various transaction functions are added next: namely, creating sample data, adding Private Data query functionality and adding an alternative, custom cross-verify lab. A new smart contract package is then installed on all peers in the lab and the contract upgraded. Adding the code is done as a single task below.

__ **98.** Click the **Explorer icon** in the VS Code editor and return to the **trading-contract.js** file under the **lib** folder.

__ **99.** Scroll down to **line 239 -** you will see the following code comment:

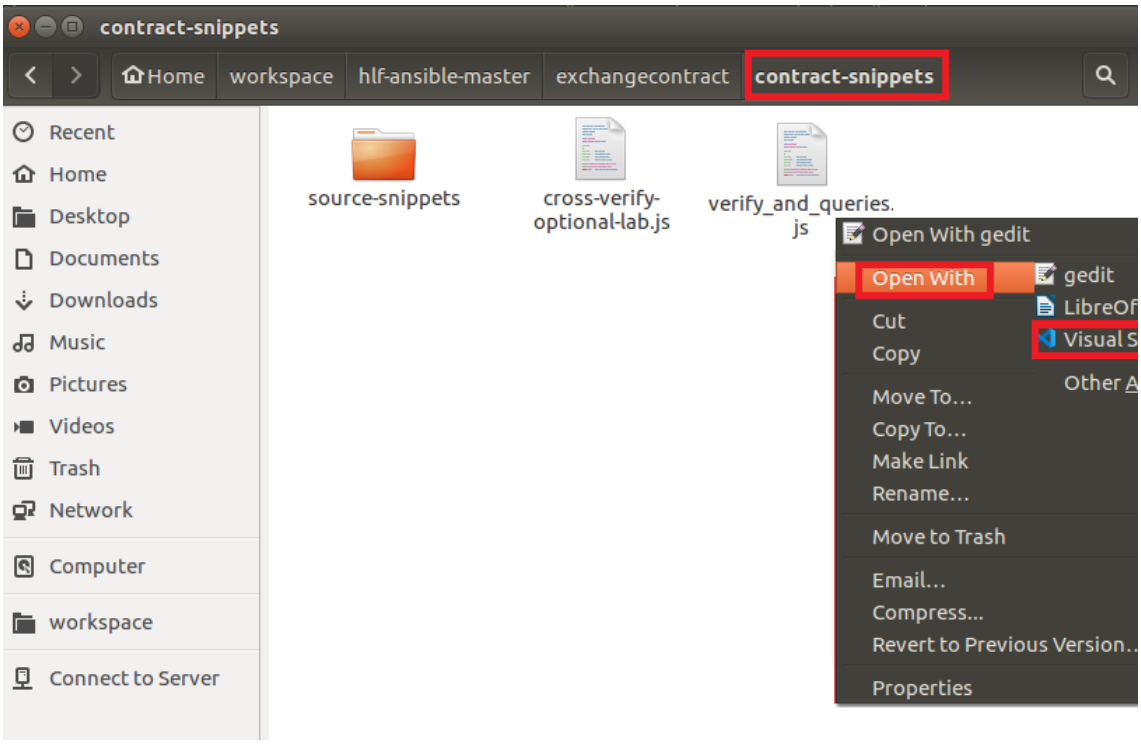
```
// UTILITY functions that may prove useful at some point
```

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

__ **100.** Next, position the cursor on **line 239** – this is where we will paste in the code snippet

__ **101.** Open the **Ubuntu File Explorer**

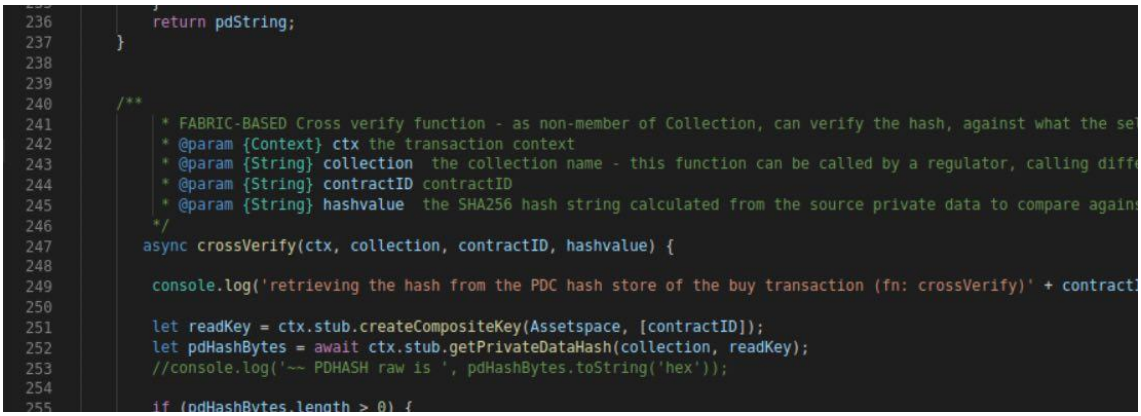
__ **102.** Navigate to the following folder:
Home > workspace > hlf-ansible-master > exchangecontract > contract-snippets
Right click on `verify_and_queries.js` - select **Open With** ... **Visual Studio Code**



__ **103.** In VS Code, highlight all the code by pressing **CTRL+A** to **select all**, then press **CTRL+C** to copy the selection to the clipboard

__ **104.** Back in the **trading-contract.js** file in VS Code, with **the cursor positioned at line 239 - paste in the contents** you copied to the clipboard using **CTRL+V**.

Scroll back up to **Line 239** to check it was pasted OK.

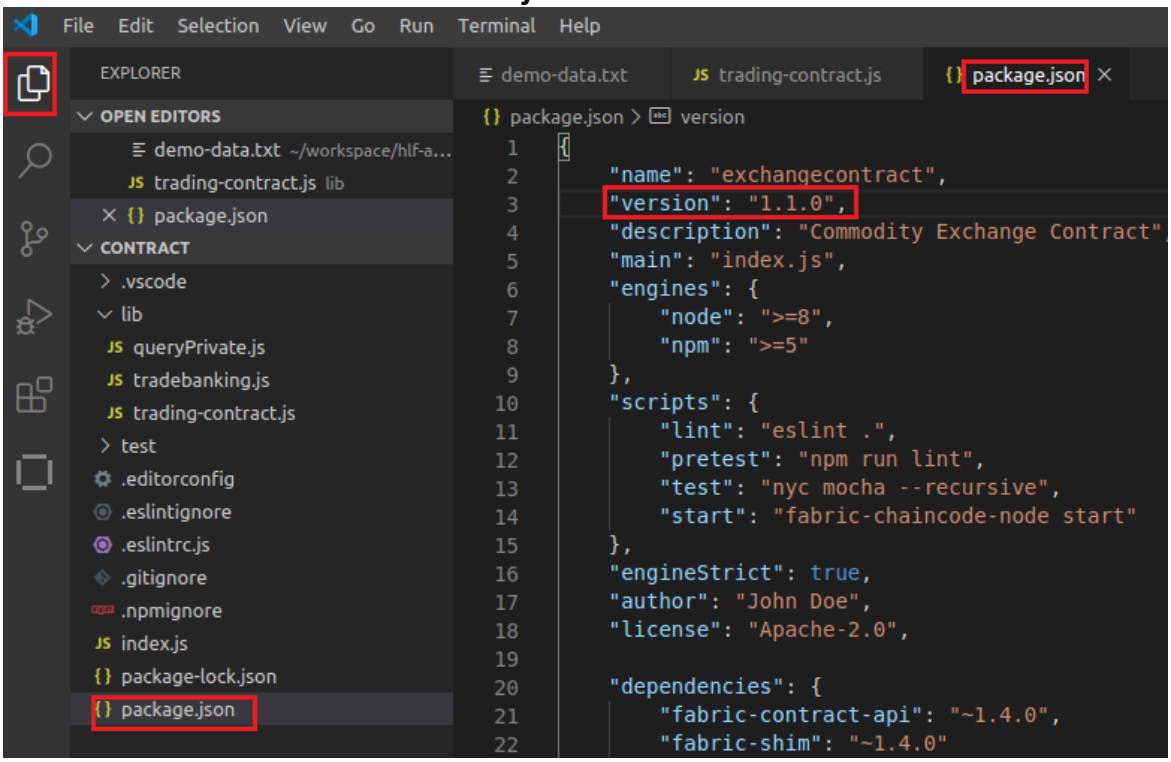


__ **105.** **Right-click** from your current line in VS Code and select the option **Format Document** to format the code and fix the indentation.

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

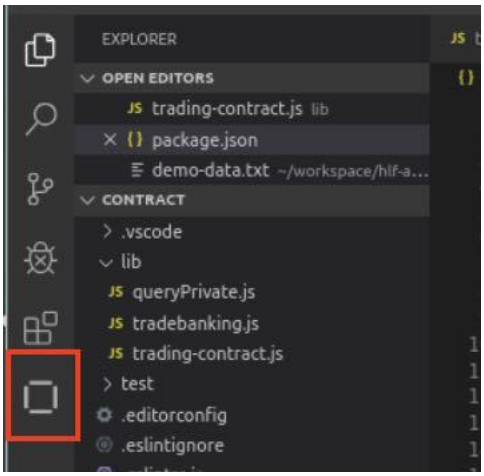
__ **106.** Save the **trading-contract.js** file by pressing **CTRL+S**.

__ **107.** Click on the VS Code Explorer icon, then click on the **package.json** file and change the version number to "**1.1.0". Ensure you hit CTRL+S** to save the file.

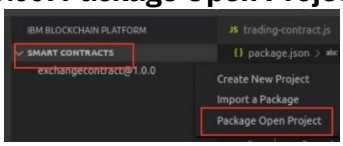


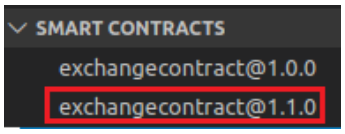
This will allow us to create a smart contract package with a new version number and enable us to upgrade the current contract instantiated on the channel.

__ **108.** Click on the **IBM Blockchain Platform extension icon** on the left in VS Code.

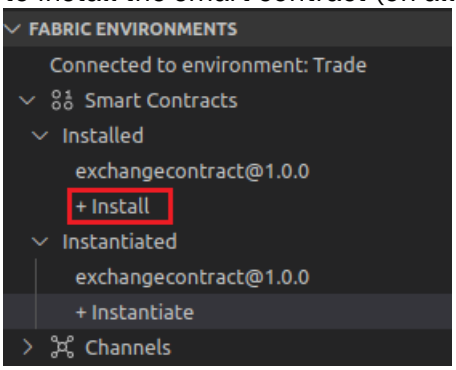


__ **109.** Hover over the **Smart Contract** view and click on the **ellipsis** (...) to the right - select **Package Open Project**.

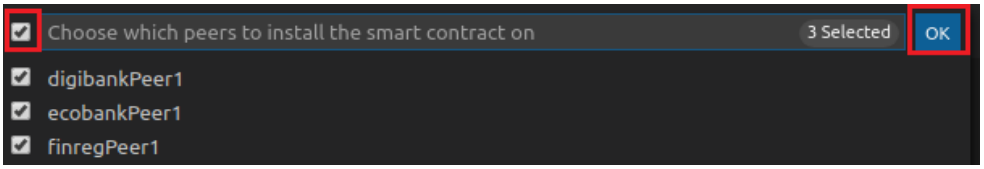




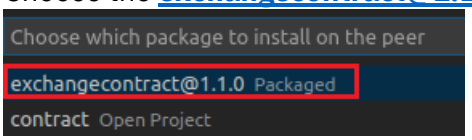
__ **110.** In the **Fabric Environments view**, still connected to the **Trade** environment, choose to install the smart contract (on all peers, in this Fabric network). Click on **Install +**



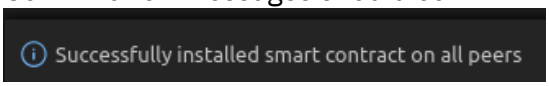
__ **111.** When prompted, click the checkbox (top left) to install on all peers and click **OK**



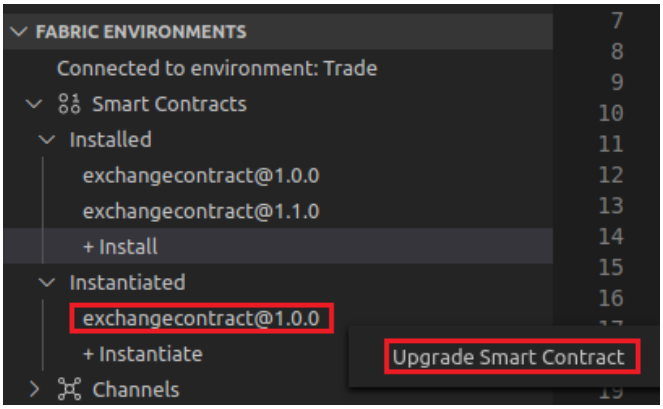
Choose the **exchangecontract@1.1.0** contract when prompted



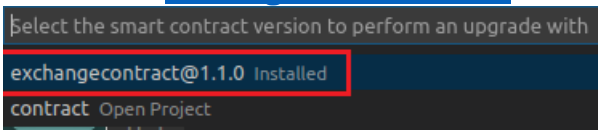
Confirmation messages should confirm it is installed on all peers



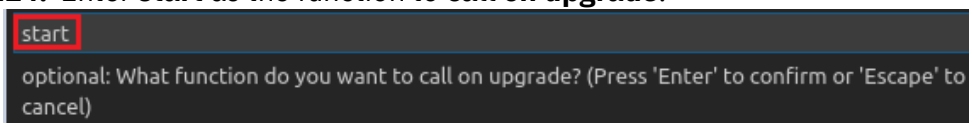
__ **112.** Under **Instantiated** contracts upgrade the running contract – choose **right-click**...**Upgrade Smart Contract** on the contract **exchangecontract@1.0.0**



__ **113.** Choose **exchangecontract@1.1.0** as the contract to upgrade with

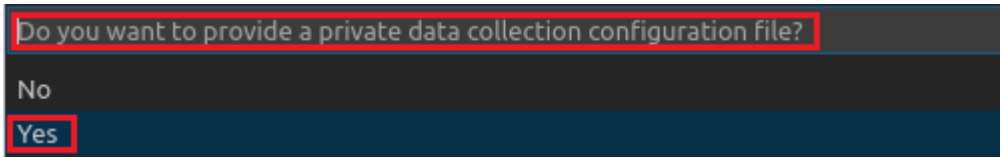


__ **114.** Enter **start** as the function to **call on upgrade**.

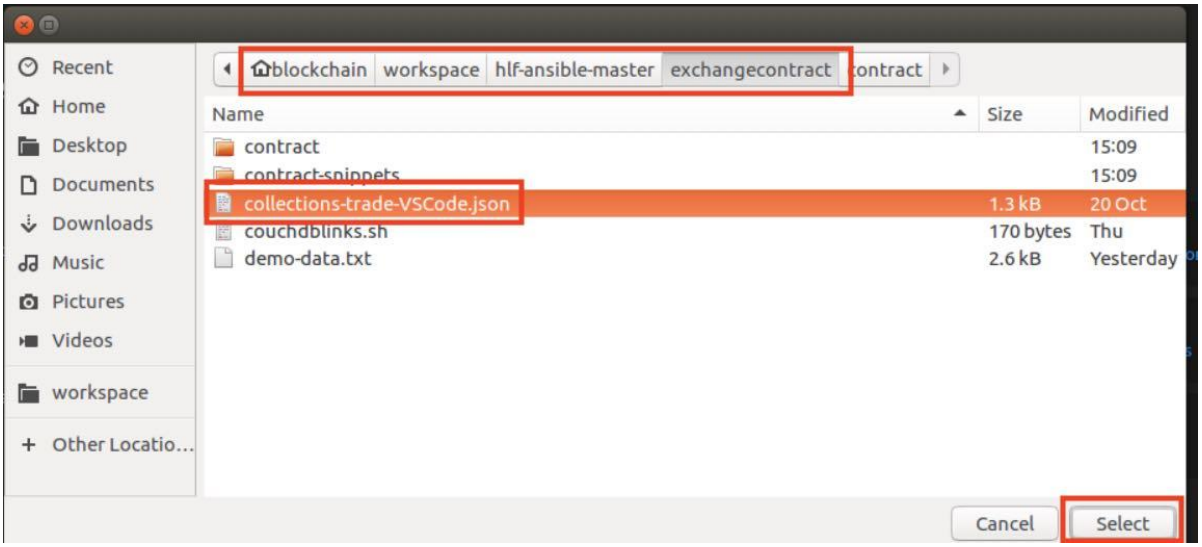


__ **115.** Press **enter** to accept the default 'no arguments' to the function (there are none).

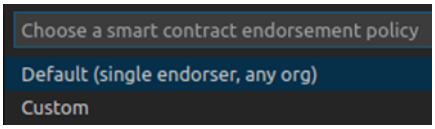
__ **116.** When asked if you 'want to provide a private data collection configuration file' – select 'Yes'



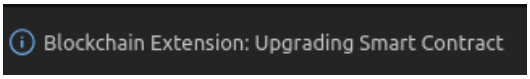
__ **117.** Click **Browse** and navigate to the folder Home > workspace > hlf-ansible-master > exchangecontract and select the file **collections-trade-VSCode.json**.



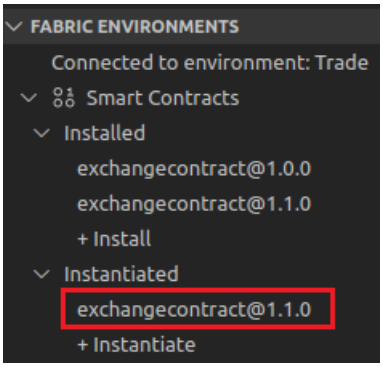
__ **118.** In the next dialogue that asks “Choose a smart contract endorsement policy” choose the default “Default (single endorser, any org)”



Upgrading may take a minute or so.



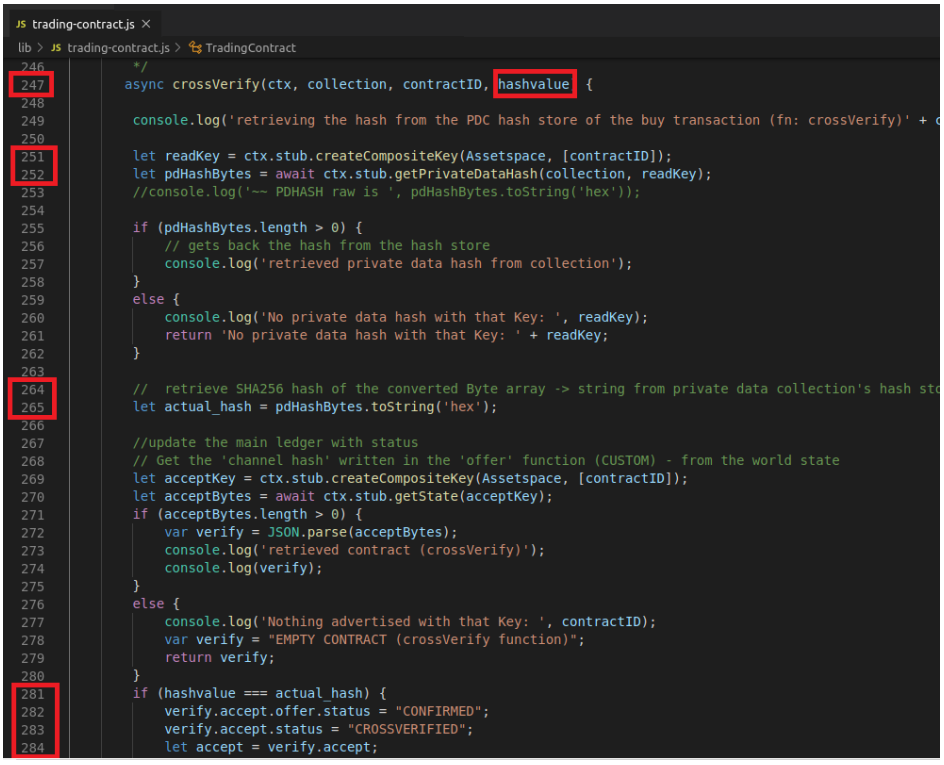
When the process has completed it will show in **Fabric Environments** view as follows:



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

__ **119.** Return to the **trading-contract.js** source in VS Code Explorer and scroll to **line 247**.

Let's look at the first transaction – **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 (hash value) – to the converted hash from line **265**

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

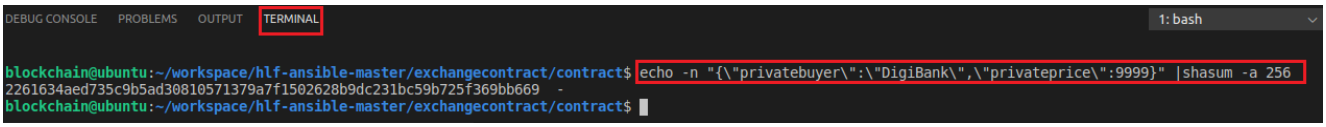
4.3 Perform the Cross Verification of Private Data

The next step is to try out the cross-verification: you will recall that EcoBank had earlier accepted the offer details provided by DigiBank, but now want to verify it, against the channel-wide hash record (ie of what DigiBank had originally created in their own private collection).

__ **120.** In the VS Code session, click the **terminal** pane at the bottom of the screen. (If you cannot see this, click on the VS Code menu and select "View->Terminal".)

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

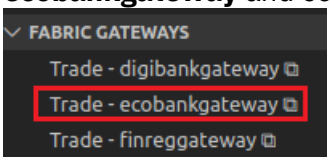
```
echo -n "{\"privatebuyer\":\"DigiBank\",\"privateprice\":9999}" |shasum -a 256
```



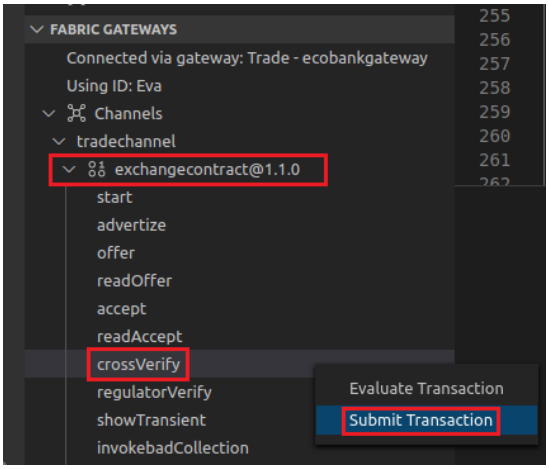
This will calculate and output, a SHA 256 hash value of the source data EcoBank had agreed with DigiBank.

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

__ **121.** In the Fabric Gateways view, if not already connected, click on **Trade - ecobankgateway** and connect with identity **Eva**



__ **122.** Expand the Channels/exchangecontract@1.1.0. Right-click **crossVerify** and click **Submit Transaction.**

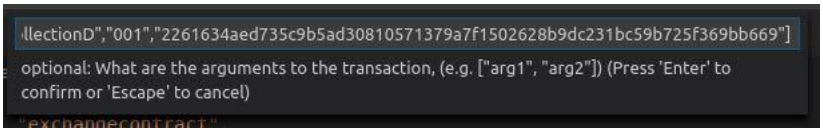


Note that we are running 'Submit Transaction' and not 'Evaluate Transaction' because the function is updating a status on the channel ledger for contract "001".

__ **123.** At the `arguments` prompt, paste in the argument/parameter list below **as one line**, in the VS Code parameter field – replacing the existing '[]' text (from here, or `demo-data.txt`)

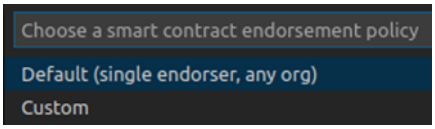
WARNING: You should copy/paste the parameter list from the file 'demo-data.txt' supplied in the **exchangecontract** subdirectory – also ensure there are no trailing spaces after the closing ']' square bracket. If you copy/paste the parameter list from the PDF document, you will NOT get a MATCH. This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

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



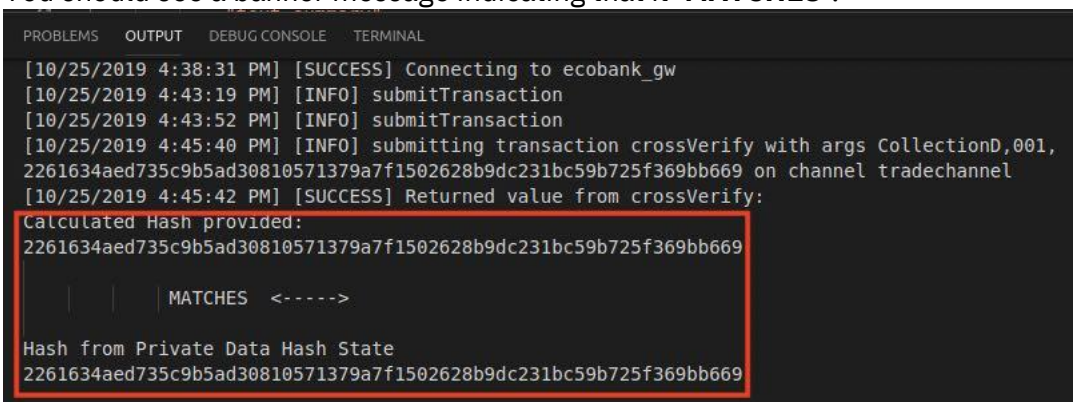
__ **124.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **125.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy



__ **126.** Wait for the transaction to complete – it should switch to the **Output** pane

You should see a banner message indicating that it '**MATCHES**':

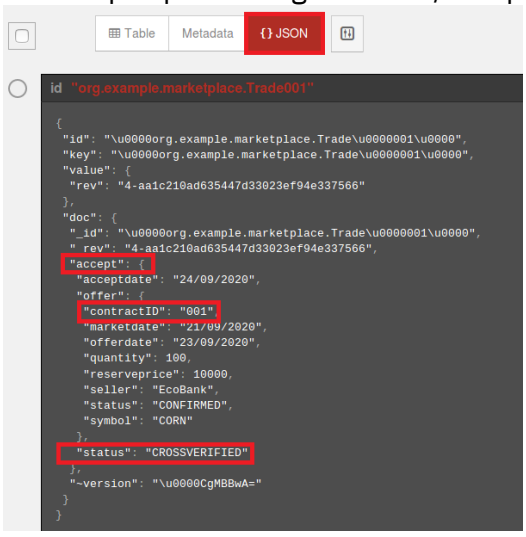


Having verified they match, we need to check that the *channel ledger record* itself, has a status of 'CROSSVERIFIED' (because it's updated by the **crossVerify** function).

__ **127.** Return to Firefox and click on the middle tab, which is **EcoBank's** CouchDB

__ **128.** As you've done before, click on the link to the channel ledger called **tradechannel_exchangecontract** in CouchDB

__ **129.** 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 the 'offer/accept' pair - the status is available to all organisations.

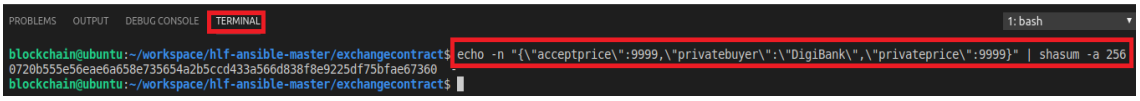


Note that equally, **DigiBank, can also take on the role of verifier** . That is, DigiBank can verify that the corresponding record in the EcoBank hash store ('CollectionE') matches what was agreed by DigiBank (and calculated as a hash) – let's try this out.

__ **130.** Switch back to the VS Code extension and under **Fabric Gateways, disconnect** as Eva and connect to the **DigiBank** gateway as **David**

__ **131.** First, return to the **terminal pane in VS Code**. When the prompt appears, paste in the following line exactly as shown (including the escape characters) on ONE line :

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



__ **132.** This will calculate and output another SHA 256 hash value of the source data. It returns a hash value beginning **0720b**. For convenience, we will provide this exact 'accept' verification hash string, in the text file `demo-data.txt`.

__ **133.** Right-click on the **crossVerify** transaction and choose **Submit Transaction**

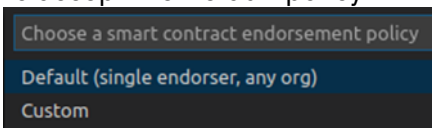
__ **134.** When prompted in the VS Code extension, paste in the following parameter list as a one-liner from the file 'demo-data.txt' in your VM image folder. Ensure you replace the existing '[]' text:

WARNING: 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. 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.

["CollectionE","001","0720b555e56eae6a658e735654a2b5ccd433a566d838f8e9225df75bfae67360"]

__ **135.** There is no transient data for this transaction, just press **enter** when prompted, leaving the empty curly braces **{}**

__ **136.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy



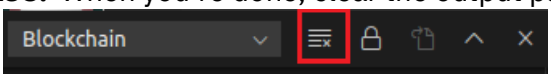
__ **137.** Wait for the transaction to complete, then select the **Output** pane and review the messages. It should provide a match of the EcoBank data hash (that you calculated) against the private data hash store record for Contract "001" – a '**MATCHES'** banner should appear.

```
[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 Collect
[11/5/2019 4:36:36 PM] [SUCCESS] Returned value from crossVerify:
Calculated Hash provided:
0720b555e56eae6a658e735654a2b5ccd433a566d838f8e9225df75bfae67360

        MATCHES  <----->

Hash from Private Data Hash State
0720b555e56eae6a658e735654a2b5ccd433a566d838f8e9225df75bfae67360
```

__ **138.** When you're done, clear the output pane by click on the 'clear' icon (far right)



4.4 Review the Regulator Verification function

Another transaction function in the contract is **regulatorVerify.** The code for this was also added earlier. 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.

__ **139.** Back in **trading-contract.**js, 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
329         else
330             return 'Could not match the Private Data Hash State: ' + actual_hash;
331
332     }
276         else {
```

We see the function's parameters are the collection name, the contract ID and a calculated hash value (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.

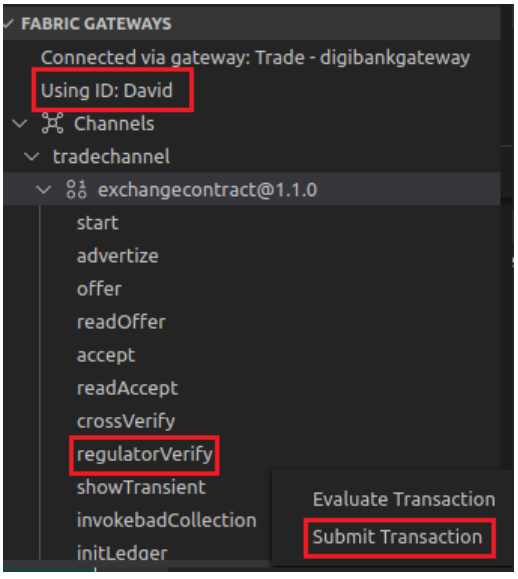
Lines 326-328 show what actions are taken dependent on whether there is a MATCH or failed MATCH between the hashes.

Further transactions added, were helper transactions like **ShowTransient** (show inputs for a given Transient set), and an **initLedger** transaction that is called once as **David** (DigiBank), and once as **Eva** (EcoBank) to create some sample/demo private data in their respective collections (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.

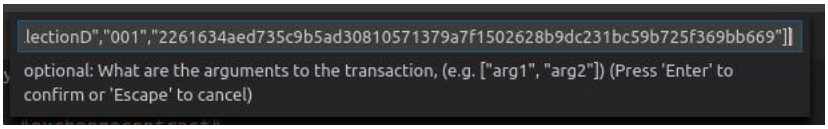
__ **140.** Still connected as the identity 'David' from DigiBank, highlight the **regulatorVerify** transaction, **right-click** on **regulatorVerify** and click **Evaluate Transaction**



__ **141.** Provide the following one-liner parameter list below as arguments when prompted, replacing the existing '[]' text:

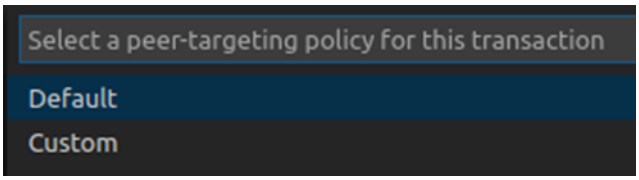
WARNING: You should copy/paste the parameter list from the file '`demo-data.txt`' supplied in the **exchangecontract** subdirectory – also ensure there are no trailing spaces after the closing ']' square bracket. If you copy/paste the parameter list 'as is' from the PDF document, it is likely you **will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

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

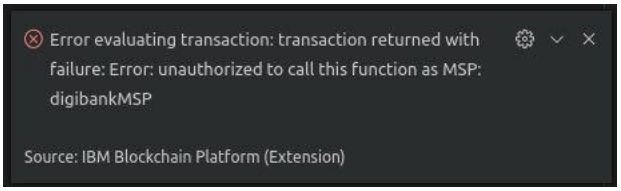


__ **142.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **143.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy



__ **144.** 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

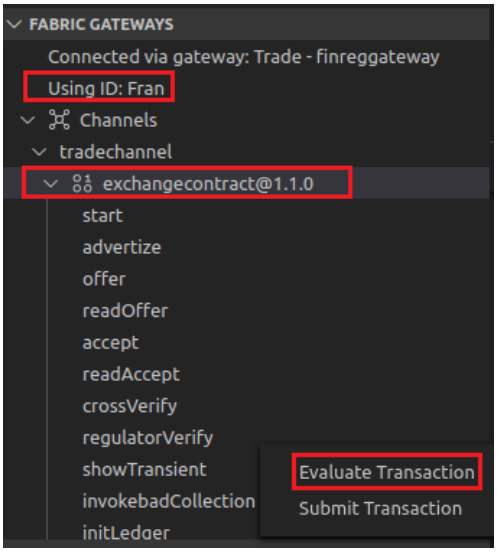


Close this popup message by clicking on the **'X'** to close. Next, you will switch to an identity from **FinReg** to call the transaction with an authorized identity.

__ **145.** Hover over the **Fabric Gateways** view and disconnect from DigiBank's gateway **Trade - digibankgateway**

__ **146.** Click on FinReg's gateway **Trade – finreggateway**. Connect with identity **Fran**.

__ **147.** Once again, **right-click** on the **regulatorVerify** transaction from the transaction list and click **Evaluate Transaction**

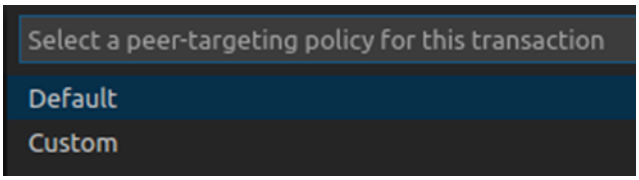


__ **148.** Provide the same parameter list as before when prompted:

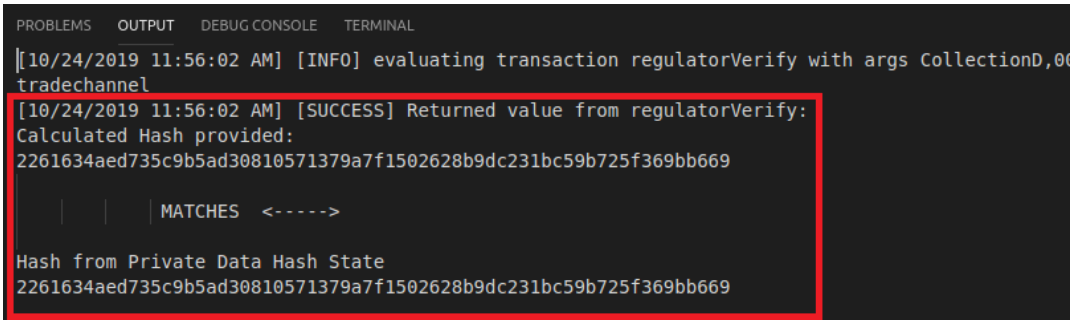
["CollectionD","001","2261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669"]

__ **149.** Again, there is no transient data for this transaction so just press **enter** leaving the empty curly braces **{}**

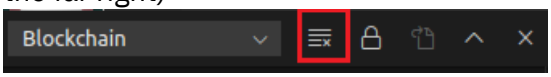
__ **150.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy



Now you should see that the evaluation is successful this time and the hashes match.

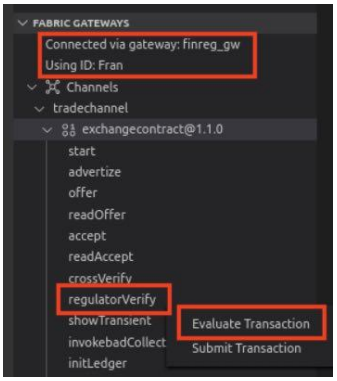


__ **151.** Once again, when you're done, clear the output pane by click on the 'clear' icon (on the far right)



Up to now, all the cross-verify and regulator verify transactions have shown a MATCH. So let's deliberately mis-MATCH a regulator verify - just to see what output we get.

__ **152.** Once again as **Fran**, **right-click** the **regulatorVerify** transaction from the transaction list and click **Evaluate Transaction**

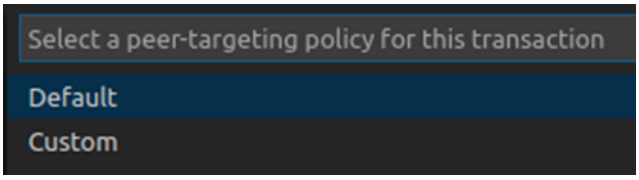


__ **153.** Copy and paste the following parameter list - this time the parameter list has been modified - the first digit of the hash is now a **9** instead of a 2:

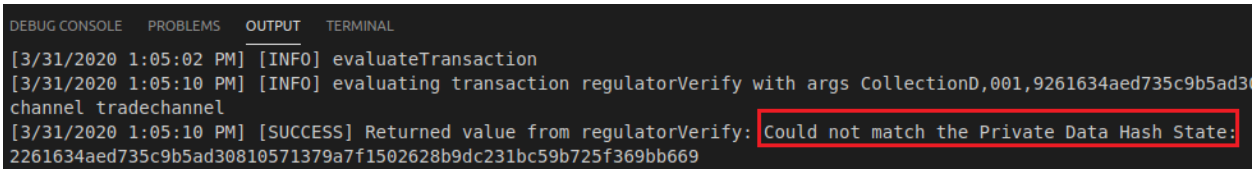
["CollectionD","001","9261634aed735c9b5ad30810571379a7f1502628b9dc231bc59b725f369bb669"]

__ **154.** Again, there is no transient data for this transaction so just press **enter** leaving the empty curly braces **{}**

__ **155.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

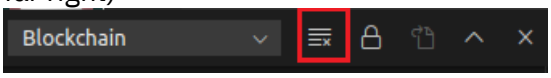


Now you should see that there is no match.



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 an investigation with the organisations involve.

__ **156.** Finally, when you're done, clear the output pane by click on the 'clear' icon (on the far right)



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:

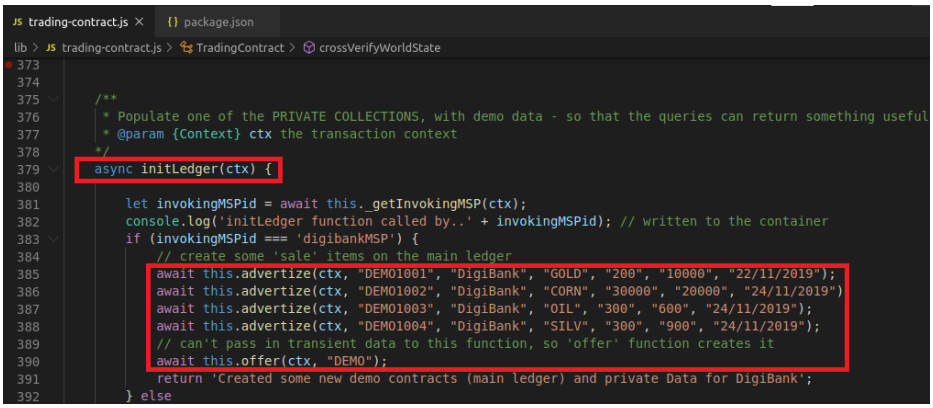
- Add sample data to two separate Private Data collections for query purposes – and
- Execute Private Data query transactions that allow you to perform 'rich queries' against the private collection data.

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** smart contract**.**

5.2 Review Demo Data and private data Rich Query transactions

Let's look at the transactions:

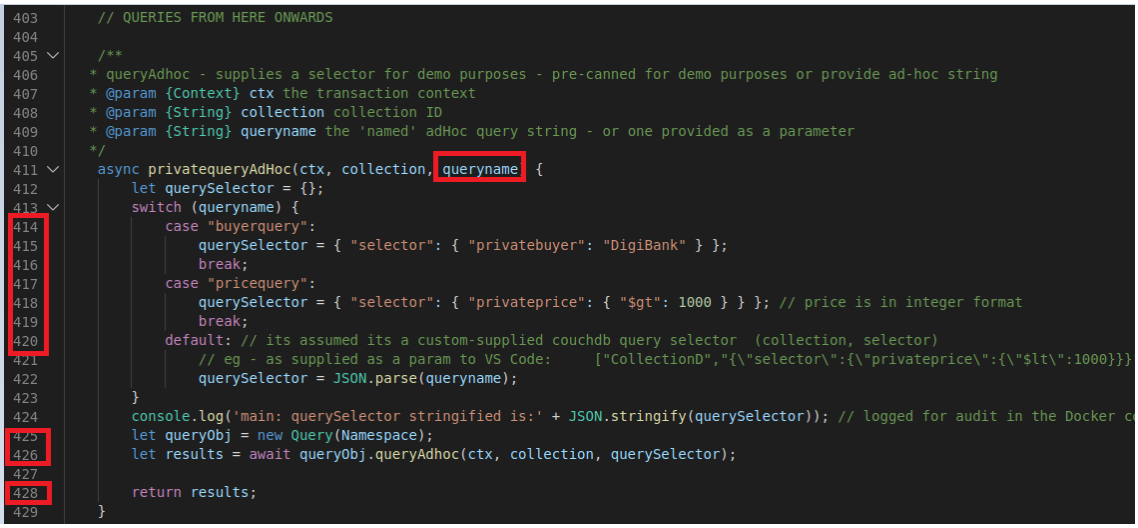
__ **157.** Click the **Explorer** icon and open **trading-contract.js** if it is not already displayed. Scroll to line 379 in the code.



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 query transactions starting at line **411**.

__ **158.** 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 c
425         let queryObj = new Query(Namespace);
426         let results = await queryObj.queryAdhoc(ctx, collection, querySelector);
427
428         return results;
429     }
```

Line 411 accepts the named query or ad-hoc as the queryname parameter, along with the collection name. Depending on the query, the selector is a pre-built query string, or the invoker of the query supplies a querystring themselves – you will see an example of using both later.

Line 425-426 sets up the query object by calling a helper transaction from the **queryPrivate.js** source (where its class is defined) and then iterates through the query results, before returning the query results to the VS Code output pane in this case on **line 428**.

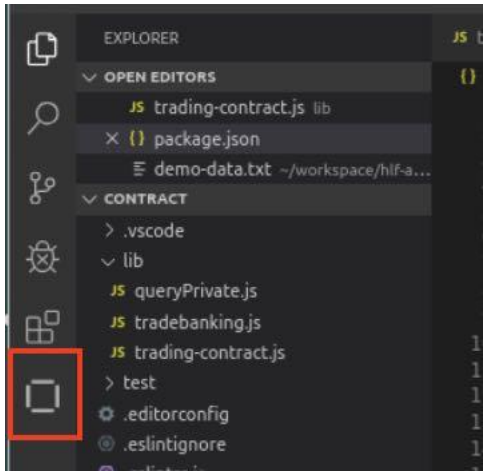
There is also a **privatequeryPartialKey** query transaction that does a private data collection query by partial key (e.g. by Asset namespace, to find all records for that namespace, in the collection name supplied).

5.3 Create the Demo Data 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.

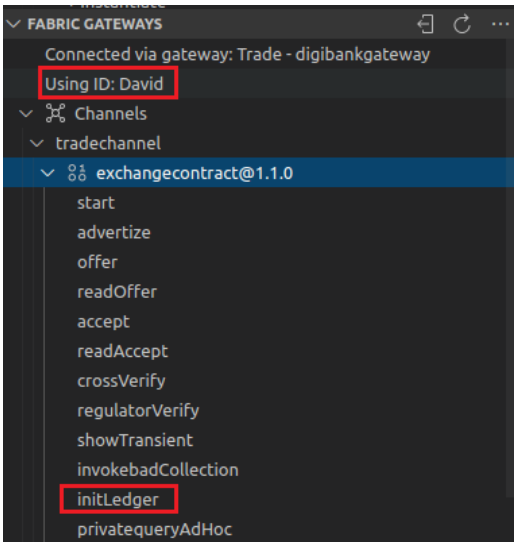
We need to return to the IBM Blockchain Platform extension.

__ **159.** Click on the IBM Blockchain Platform extension icon to return.



__ **160.** On the **Fabric Gateways** view, **disconnect** from FinReg's gateway and click on the **Trade - digibankgateway** and connect with identity **David.**

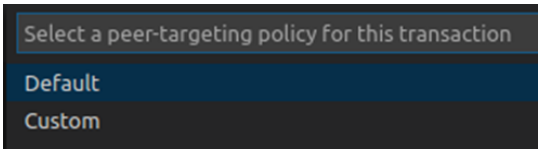
__ **161.** Expand the **exchangecontract@1.1.0** contract to get a transaction list, and right-click the **initLedger** transaction**.** Click **Submit Transaction**



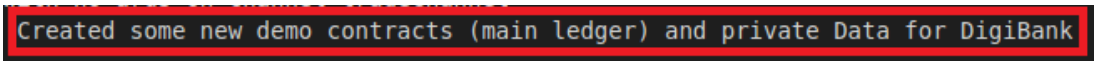
__ **162.** There are no parameters, just press **enter** leaving the empty square brackets **[]**

__ **163.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **164.** When asked to "**Select a peer targeting policy for this transaction**" just **press enter** to accept the Default policy

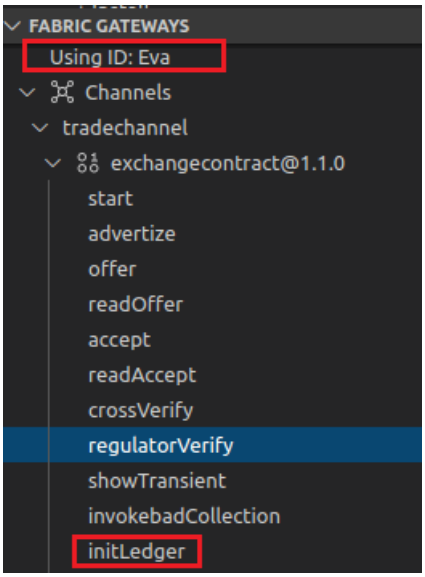


The transaction should return a 'success' message in the output pane, indicating the demo data was created:



__ **165.** On the **Fabric Gateways** view **disconnect** from DigiBank's gateway and click to **EcoBank's gateway** and **connect** with identity **Eva.**

__ **166.** Scroll down to the transaction **initLedger**, right click then click **Submit Transaction**



__ **167.** Once again, there are no parameters, just press **enter** leaving the empty square brackets **[]**

__ **168.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **169.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

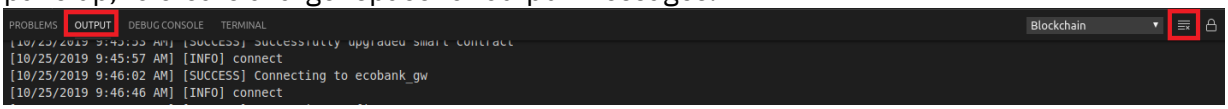
__ **170.** Review the messages in the output pane – it should indicate that the private demo data was created this time for EcoBank

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 Performing Rich Queries against the Private Data Collections

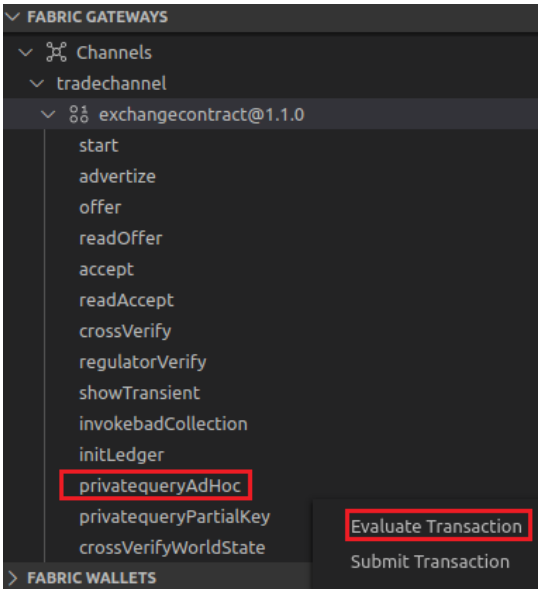
This section performs the execution of the queries we described earlier. So that we can more easily see what is happening, we're going to reset the Output pane in VS Code and make it larger.

__ **171.** 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.



__ **172.** On the **Fabric Gateways** view disconnect from EcoBank's gateway and connect to **DigiBank's gateway** and connect with identity **David.**

__ **173.** Expand the **exchangecontract** and right click **privatequeryAdhoc**. Click **Evaluate Transaction**



__ **174.** When prompted, **copy and paste** the following parameters, replacing the '[]' text offered with the following parameter list:

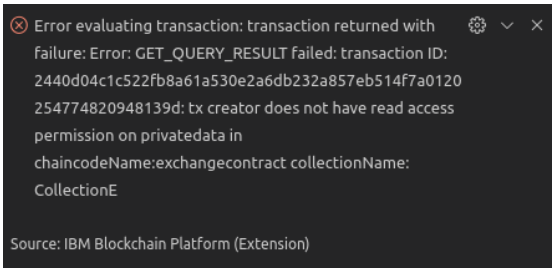
```
["CollectionE","buyerquery"]
```

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.

__ **175.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **176.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

Notice that we provided **`CollectionE`** 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)



__ **177.** Close the error popup message and clear the Output pane using the **Clear Output** icon, as shown previously.

__ **178.** Once again, right-click **privatequeryAdhoc** and click **Evaluate Transaction**.

__ **179.** When prompted, paste in the following parameters, replacing the '[]' text offered with the following 'named' query called 'buyerquery' (and no trailing spaces)

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

__ **180.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **181.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

__ **182.** Review the output pane – this time, it should show the expected query results.

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

__ **183.** Once again, right-click **privatequeryAdhoc** and click **Evaluate Transaction**.

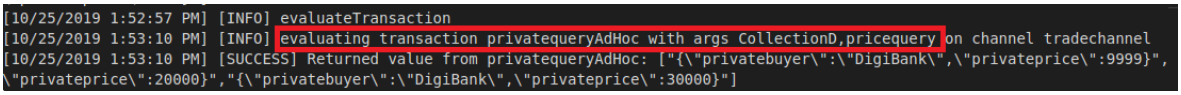
__ **184.** When prompted, paste in the following parameters, replacing the '[]' text offered with the following 'named' query, 'pricequery':

["CollectionD","pricequery"]

__ **185.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

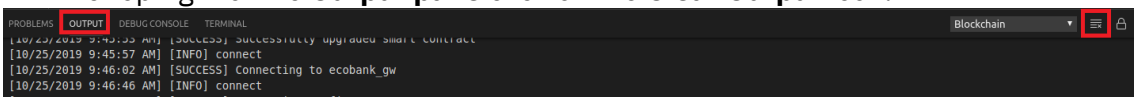
__ **186.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

__ **187.** Review the output pane for messages – you should see the query results – a screenshot of sample output is shown below – you will see different results. Missing from these results are two records from the earlier results: less than price of 1000 USD. Its because query `pricequery` inside the smart contract, only shows prices over 1000.



Next you will try some queries as EcoBank. First clear the output pane.

At the top right of the **Output pane** click on the **Clear Output** icon.



__ **188.** On the **Fabric Gateways** view disconnect from DigiBank's gateway and click **ecobank_gw** and connect with identity **Eva.**

__ **189.** Expand the exchangecontract and right click **privatequeryAdhoc**. Click **Evaluate Transaction**

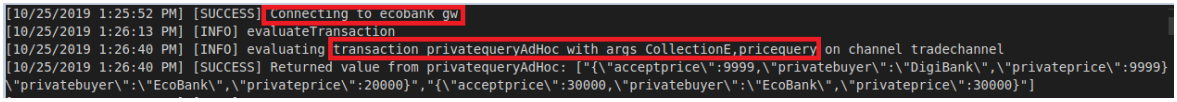
__ **190.** When prompted, paste in the following parameters, replacing the '[]' text offered with the following – and press **enter**

["CollectionE","pricequery"]

__ **191.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **192.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

__ **193.** 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 actual 'ad-hoc' query string to the **privatequeryAdhoc** transaction – the transaction allows a user to supply a custom query – eg 'show me any accept/offer record pairs, whose values are less than 1000 USD'.

__ **194.** Once again as **Eva**, right-click **privatequeryAdhoc** and click **Evaluate Transaction**.

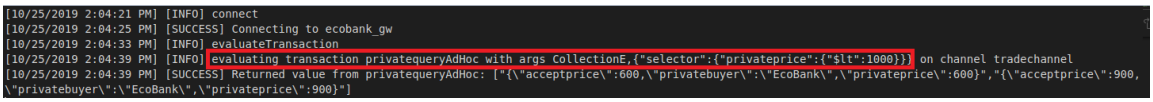
__ **195.** When prompted, paste in the following parameters, replacing the '[]' text offered, with the following string, pasted in exactly as shown – and press **enter**:

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

__ **196.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **197.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

__ **198.** Review the messages in the output pane – you should see a different set of results, that is, there are the two records that match the criteria we provided ("less than 1000").



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

Review

You've now successfully:

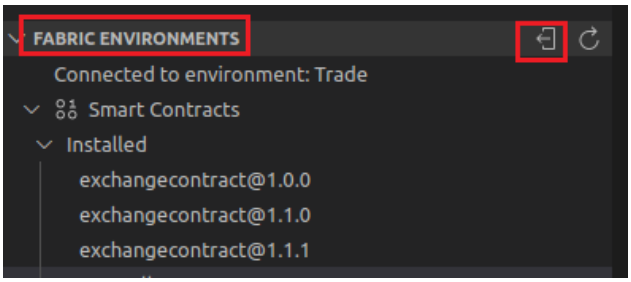
- created some additional sample data
- performed queries against private data collections
- understood more about using rich queries to filter result sets, by using different criteria.

Finally, you need to **clean up the virtual machine** in preparation for the next VM based lab – instructions are shown below. However, **you may wish to perform the optional lab exercise first** – i.e. section 6: "Executing a '**custom**' (and alternative) Cross Verify lab". If you do have time to complete the lab exercise in section 6 , go straight to **section 6** . After completing section 6 - return to the clean-up exercises shown below.

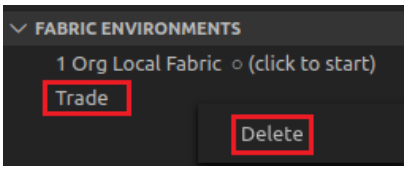
Lab Clean-up Steps:

__ **199.** **Disconnect** from any active gateway connected under **Fabric Gateways**

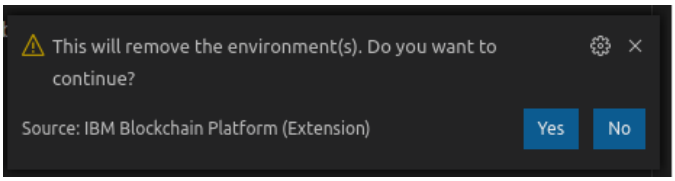
__ **200.** Return to the **Fabric Environments** view and **disconnect** from the current **Trade** Fabric environment:



__ **201.** In the **Fabric Environments** view **right-click** on the **Trade** environment and click on **Delete** to delete the environment

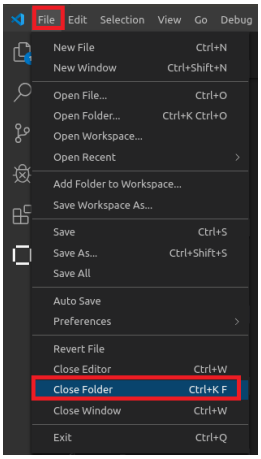


__ **202.** On the bottom right, you will see a popup to **confirm removal** of the environment – click **Yes** to continue -



You should now see that all (except Local Fabric) environments, gateways, wallets and nodes are gone. There are three remaining steps – closing out the contract folder, tearing down the Fabric docker container environment and closing the CouchDB views.

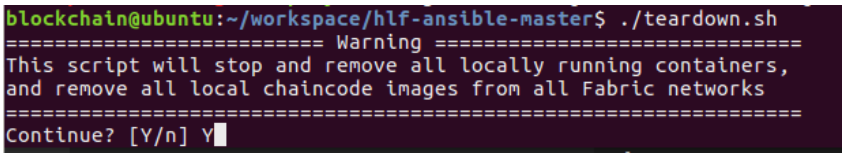
__ **203.** Next, in VS Code, close the open smart contract project by clicking on the menu bar – select 'File ....Close Folder'



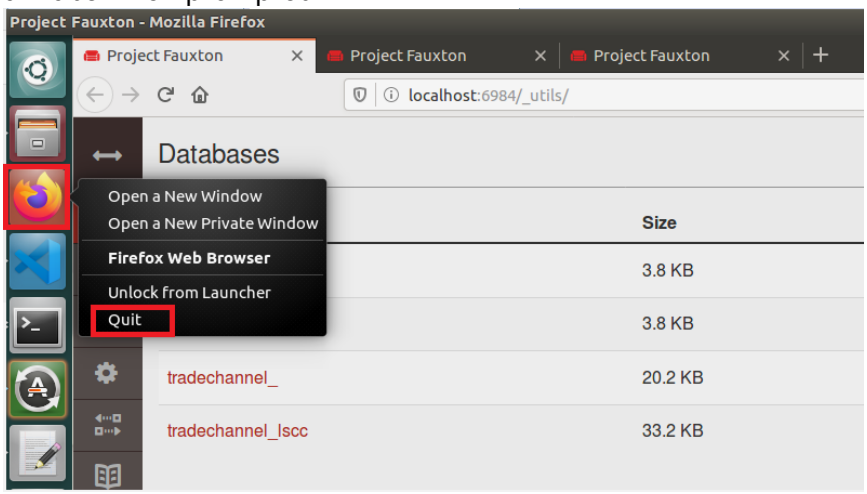
__ **204.** In the terminal window, use the following two commands to navigate to the **hlf-ansible-master** folder and teardown the custom network using a bash script – confirm **'Y'** to continue:

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

```
./teardown.sh
```



__ **205.** From the task bar, right-click on the Firefox icon and select 'Quit' – confirm to close all tabs when prompted



This completes the teardown section of the lab.

6 Optional Lab: Executing a 'custom' cross-verify method

6.1 Introduction

In Part 4 of this lab guide, you completed the Fabric Private Data collections 'cross-verify' pattern.

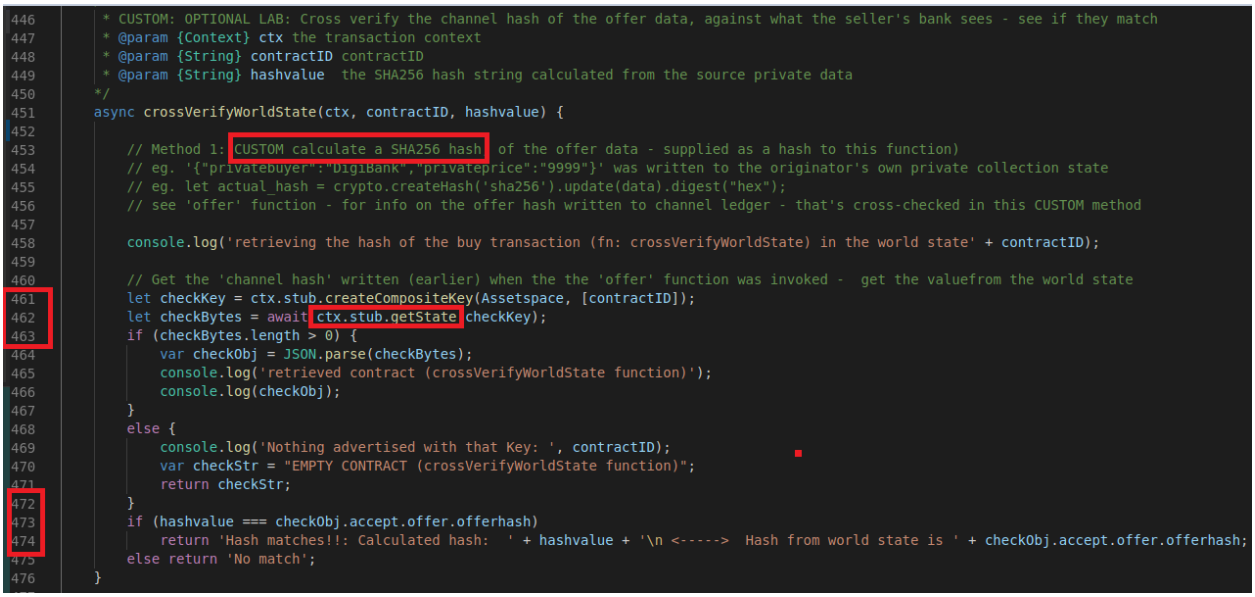
In this optional lab, you will work with a 'custom' cross-verify method, where the cross-verification is performed against an SHA256 hash of an offer stored on the channel ledger world state (and not linked to a private data collection).

The cross-verify process is the same as before: the verifier calculates a hash of a given set of source data, and then calls the cross-verify transaction to check, providing the calculated hash as a parameter, whether it matches the corresponding world state hash value.

Note that the **crossVerifyWorldState** function has already been added to the code but is not enabled. To enable, you will need to uncomment one line, as described in the lab below.

The next steps are to review the custom code the lab and implement it.

6.2 Review and Upgrade to add the Custom Cross-Verify function



```
446     * CUSTOM: OPTIONAL LAB: Cross verify the channel hash of the offer data, against what the seller's bank sees - see if they match
447     * @param {Context} ctx the transaction context
448     * @param {String} contractID contractID
449     * @param {String} hashvalue  the SHA256 hash string calculated from the source private data
450     */
451    async crossVerifyWorldState(ctx, contractID, hashvalue) {
452
453        // Method 1: CUSTOM calculate a SHA256 hash of the offer data - supplied as a hash to this function)
454        // eg. '{"privatebuyer":"DigiBank","privateprice":"9999"}' was written to the originator's own private collection state
455        // eg. let actual_hash = crypto.createHash('sha256').update(data).digest("hex");
456        // see 'offer' function - for info on the offer hash written to channel ledger - that's cross-checked in this CUSTOM method
457
458        console.log('retrieving the hash of the buy transaction (fn: crossVerifyWorldState) in the world state' + contractID);
459
460        // Get the 'channel hash' written (earlier) when the the 'offer' function was invoked -  get the valuefrom the world state
461        let checkKey = ctx.stub.createCompositeKey(Assetspace, [contractID]);
462        let checkBytes = await ctx.stub.getState(checkKey);
463        if (checkBytes.length > 0) {
464            var checkObj = JSON.parse(checkBytes);
465            console.log('retrieved contract (crossVerifyWorldState function)');
466            console.log(checkObj);
467        }
468        else {
469            console.log('Nothing advertised with that Key: ', contractID);
470            var checkStr = "EMPTY CONTRACT (crossVerifyWorldState function)";
471            return checkStr;
472        }
473        if (hashvalue === checkObj.accept.offer.offerhash)
474            return 'Hash matches!!: Calculated hash:  ' + hashvalue + '\n <-----> Hash from world state is ' + checkObj.accept.offer.offerhash;
475        else return 'No match';
476    }
```

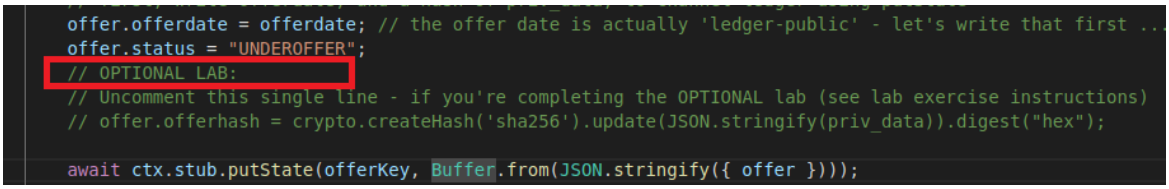
Line 453 (per screenshot above) mentions that a calculated hash of the offer data (in this case) is provided to this function as a parameter. The hash attribute in the world state is a line of code, that's currently commented out - inside the 'offer' transaction (see Part 3) – you will uncomment this later.

Line 461-463 shows that a **getState** to get the world state record and part of this is the offerhash field.

Line 472-474 shows a check between the calculated value and the offerhash stored in the world state. It will return a message indicating whether they MATCH or there is no MATCH.

__ **206.** In VS Code in the file **trading-contract.js**, **scroll** up to **line 116** approx. – inside the 'offer' transaction function and locate the following comment line that begins:

// OPTIONAL LAB



```
offer.offerdate = offerdate; // the offer date is actually 'ledger-public' - let's write that first ...
offer.status = "UNDEROFFER";
// OPTIONAL LAB:
// Uncomment this single line - if you're completing the OPTIONAL lab (see lab exercise instructions)
// offer.offerhash = crypto.createHash('sha256').update(JSON.stringify(priv_data)).digest("hex");

await ctx.stub.putState(offerKey, Buffer.from(JSON.stringify({ offer })));
```

__ **207.** Uncomment **line 118** that begins with: **// offer.offerhash**

Remove the leading '//' only - so that it now reads something like:

offer.offerhash = crypto.createHash **//** <.....continued...>

```
115     offer.status = "UNDEROFFER";
116     // OPTIONAL LAB:
117     // Uncomment this single line - if you're completing the OPTIONAL lab (see lab exercise instructions)
118     offer.offerhash = crypto.createHash('sha256').update(JSON.stringify(priv_data)).digest("hex");
119
120     await ctx.stub.putState(offerKey, Buffer.from(JSON.stringify({ offer })));
```

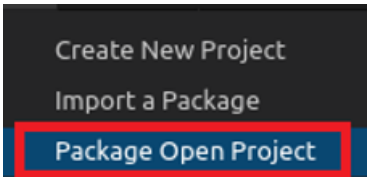
__ **208.** This now enables the previously implemented 'offer' transaction to write a SHA256 sum of the private data (the variable priv_data in the code) to the ledger world state.

Ensure you save this contract file **trading-contract.js** by pressing **CTRL+S.**

__ **209.** Switch to the file package**.json** file in VS Code Explorer - change the version number to the next increment (**1.1.1)** and ensure you save the file using **CTRL+S.**

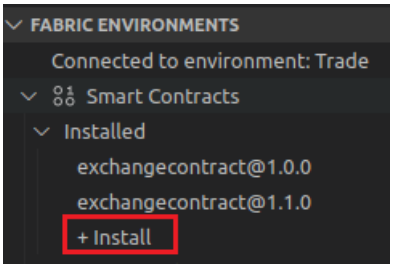
```
{} package.json > {} scripts
1   {
2       "name": "exchangecontract",
3       "version": "1.1.1",
4       "description": "Commodity Exchange Contract",
5       "main": "index.js",
6       "engines": {
7           "node": ">=8",
8           "npm": ">=5"
9       },
```

__ **210.** Click on the IBM Blockchain Platform extension icon and choose to **Package Open project** as before.



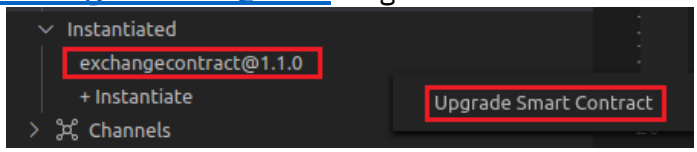
__ **211.** As before, in the Fabric Environments view, click on **Trade** environment to connect to the Fabric Environment.

__ **212.** Click on **+Install** to install the smart contract package on the peer – choose the version **exchangecontract@1.1.1** when prompted.

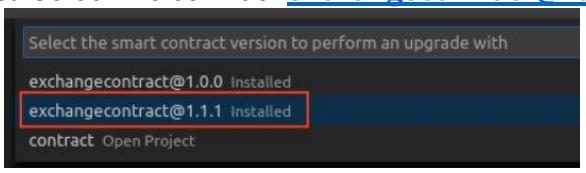


__ **213.** As you had done previously with **exchangecontract@1.1.0** – select all peers checkbox, to **install on all peers**

__ **214.** Similar to a step you had done previously, highlight the existing, instantiated **exchangecontract@1.1.0** …right-click and choose to **Upgrade Smart Contract**



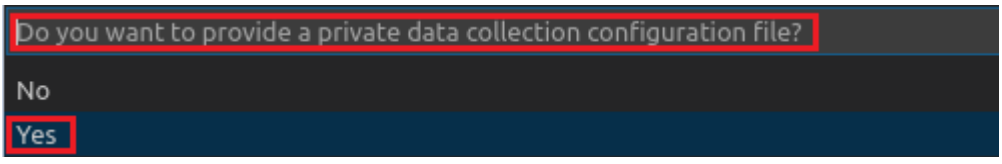
__ **215.** Select the contract **exchangecontract@1.1.1** as the contract to upgrade with



__ **216.** Provide a function name of **start** when prompted.

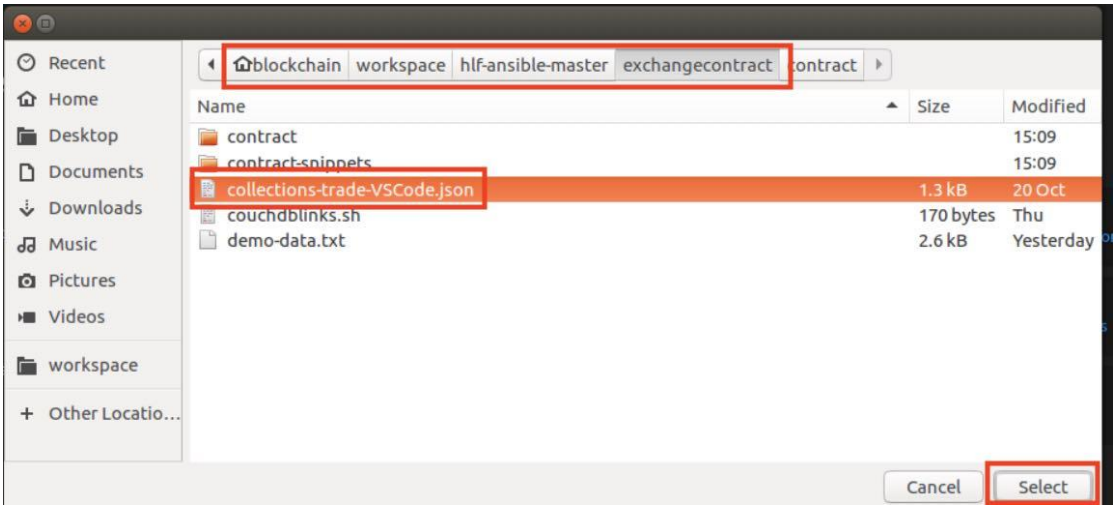
__ **217.** There are no parameters for the start function, just press **enter** leaving the empty square brackets **[].**

__ **218.** When prompted to enter a collections configuration file, select **Yes.**

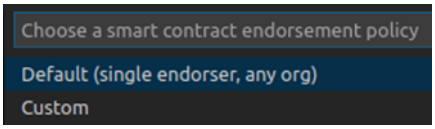


__ **219.** Click **Browse** and navigate to the folder
Home > workspace > hlf-ansible-master > exchangecontract

__ **220.** Select the file **collections-trade-VSCode.json.**



__ **221.** In the next dialogue that asks “Choose a smart contract endorsement policy” choose the default “Default (single endorser, any org)”



The upgrade is now taking place - after a short time, it will show a successful upgrade message.

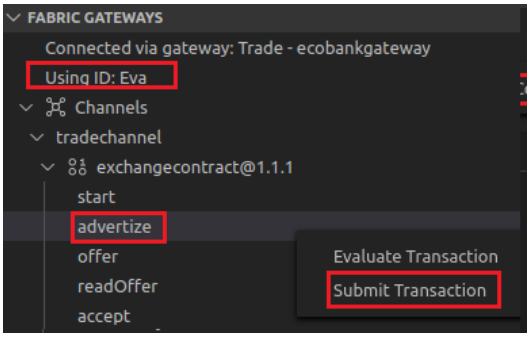
When done, disconnect from the **Trade** Fabric environment.

We’re now ready to try out the newly added custom cross-verify function, and the first step is to advertise a new commodity contract asset for sale on the marketplace.

6.3 Performing the custom Cross Verification of Private Data

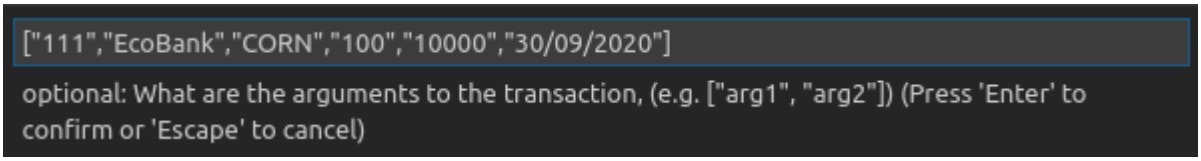
__ **222.** Connect to the EcoBank gateway – using the identity **Eva.**

__ **223.** Right click on the **advertize** transaction and click **Submit transaction**.



__ **224.** 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**

["111","EcoBank","CORN","100","10000","30/09/2020"]



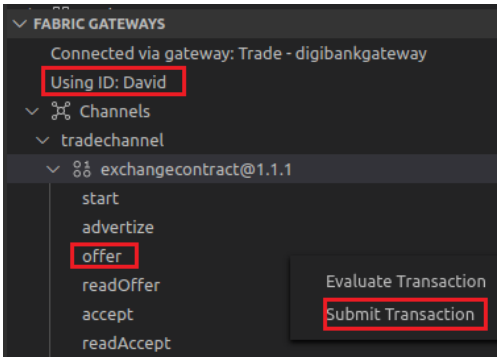
__ **225.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **226.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

After a successful submission of the *advertize* transaction, we can create an **offer** transaction – creating private data relating to Contract ID 111.

__ **227.** Disconnect from EcoBank's gateway and **connect** as **David** to DigiBank's gateway.

__ **228.** Highlight**,** then right-click on the **offer** transaction and **Submit Transaction**

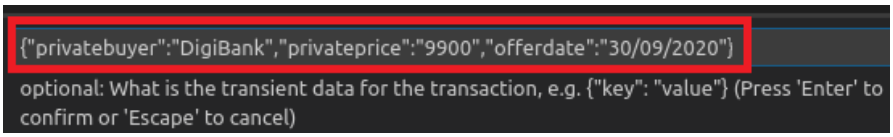


__ **229.** Provide the following parameter list (overwriting the existing '[]' text) when prompted:

["111"]

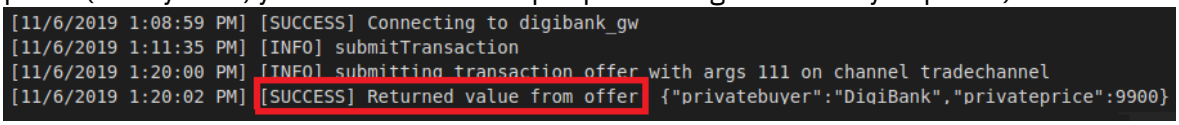
__ **230.** When prompted for transient data, replace the existing {} and paste in the following data list, overwriting the offered '{}' text – paste in the text below and press **enter**

{"privatebuyer":"DigiBank","privateprice":"9900","offerdate":"30/09/2020"}

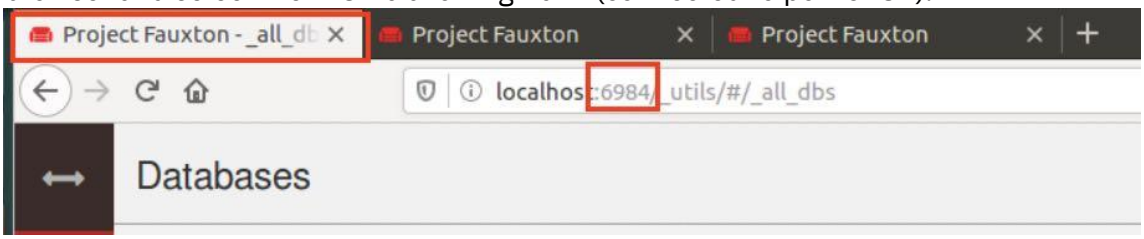


__ **231.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

__ **232.** Scroll down, to check for confirmation of a successful **Offer** transaction in the output pane. (At any time, you can clear the output pane using the icon if you prefer)

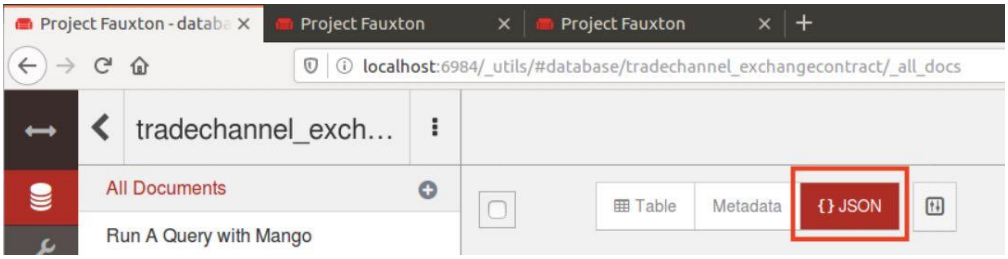


__ **233.** Once again, let's examine the CouchDB views for activity. Click on the **Firefox** browser and select the **first tab** for DigiBank (connected to port 6984).



Let's examine the Contract id 111 record.

__ **234.** Click on the database named **tradechannel_exchangecontract** the channel ledger database and click the **{} JSON** view.



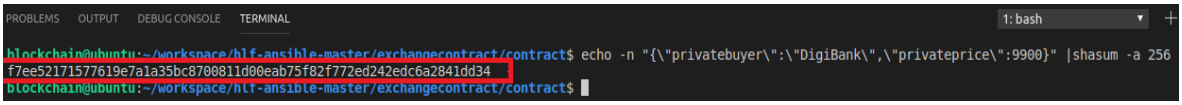
__ **235.** Examine the contents of the Trade111 record **`org.example.`**marketplace**`.Trade111`**

We can see that in the table that the record has a field called '**offerhash**' (shown above) – this was calculated written at the time of the 'offer' transaction.

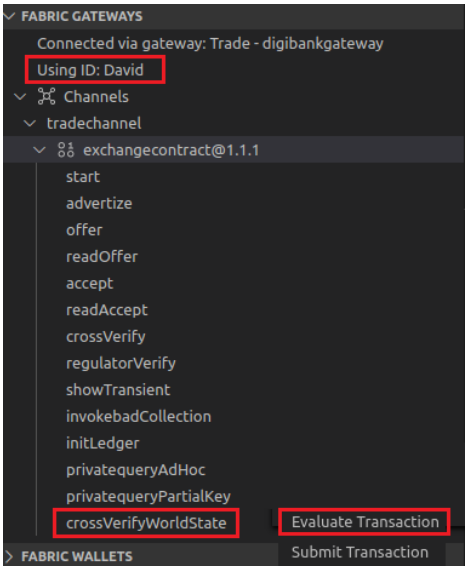
To calculate your own hash - for a given source data set – open a terminal window in VS code – click on '**Terminal'** in the bottom pane of VS Code.

__ **236.** On the command line **paste** in the following command using right click or **CTRL+SHIFT+V** to carry out the hash calculation:

echo -n "{\"privatebuyer\":\"DigiBank\",\"privateprice\":9900}" | shasum -a 256

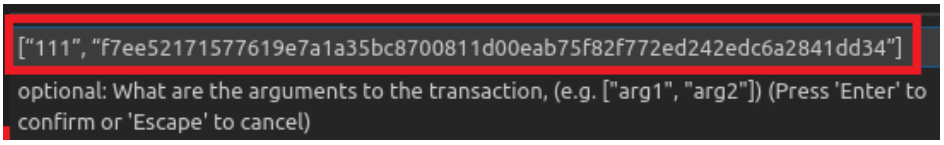


__ **237.** Now go back to the transaction list in the VS Code extension, still connected as DigiBank and **right click** on the newly added function **crossVerifyWorldState** and click **Evaluate Transaction.**



__ **238.** When prompted, provide two parameters as shown, paste in the following parameter list from the open `demo-data.txt` file, overwriting the '[]' text then press **enter.**

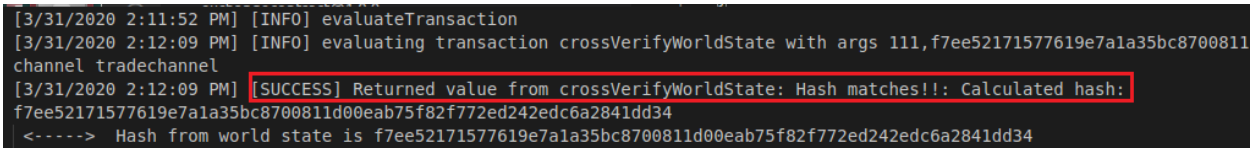
["111","f7ee52171577619e7a1a35bc8700811d00eab75f82f772ed242edc6a2841dd34"]



__ **239.** There is no transient data for this transaction, just press **enter** leaving the empty curly braces **{}**

__ **240.** When asked to "**Select a peer targeting policy for this transaction**" just press enter to accept the Default policy

__ **241.** Review the messages in the output pane for the result of this cross-verify – it should show that there is a match between the calculated hash – and the value that was written (as a channel ledger record hash) to the world state after the invocation of the 'offer' transaction.



This concludes the end of the executing a custom, cross-verify pattern lab.

Review

In this section of the Lab you have:

- Created a custom cross-verify function that validates the hash of source data, against a custom hash (that's written on the ledger World state).
- Performed the cross-verify function and checked whether the two hashes match.

Having completed this lab, return to **Step 198-204** to **clean up** the environment after this lab – this is important.

Congratulations on completing this lab!

7 We Value Your Feedback!

- Your feedback is very important to us as we use it to continually improve the lab material.
- To give us feedback after the lab has finished, please send your comments 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), channel members who are not authorized to the private data collection could try to guess the content of the private data via brute force hashing of the domain space, in hopes of finding a match with the private data hash on the chain. Private data that is predictable should therefore include a random "salt" that is concatenated with the private data key and included in the private data value, so that a matching hash cannot realistically be found via brute force. The random "salt" can be generated at the client side (e.g. by sampling a secure pseudo-random source) and then passed along with the private data in the transient field as transient data, at the time of chaincode invocation. For more information, see the Fabric documentation:

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