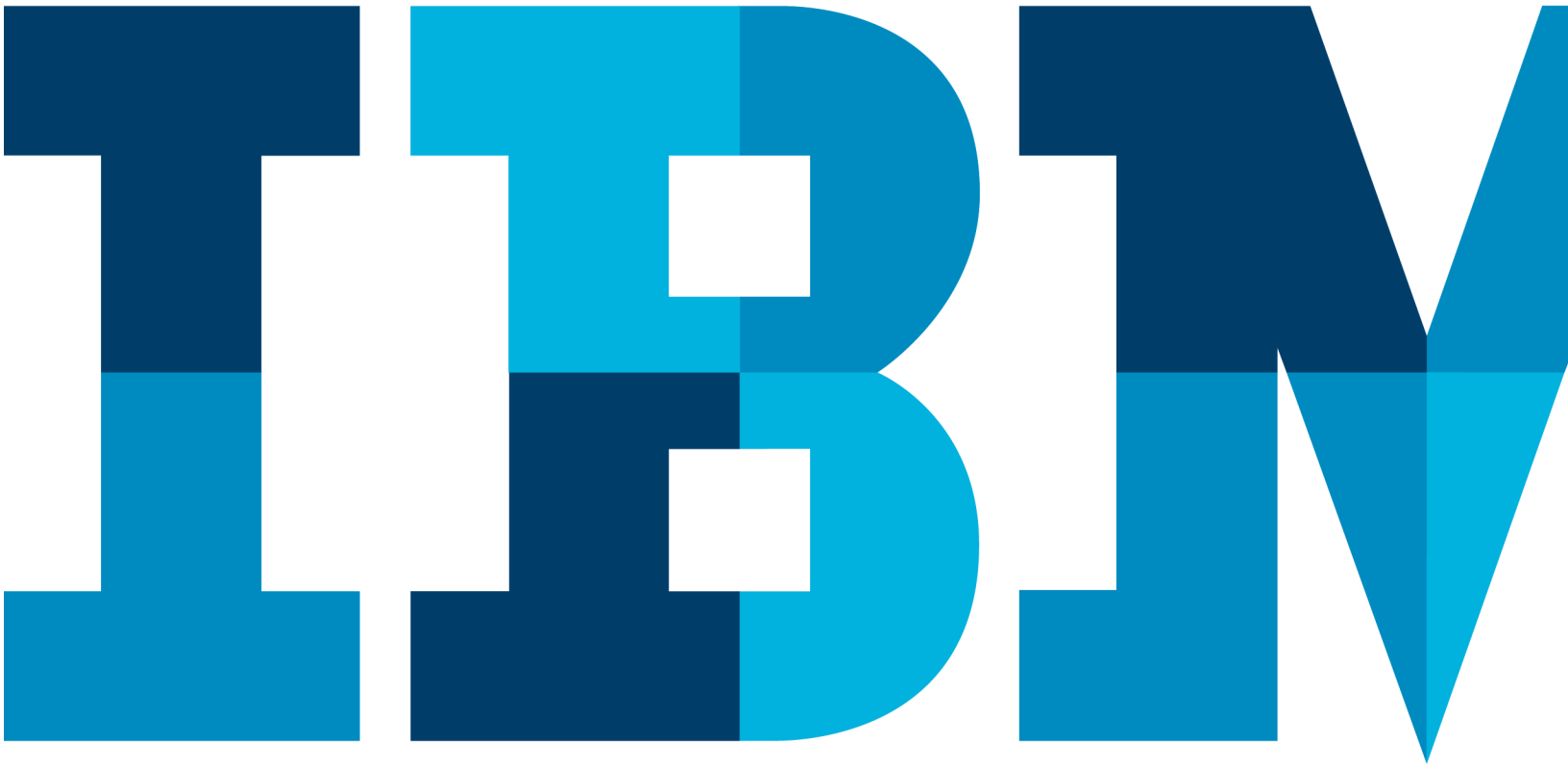
IBM Blockchain Hands-On

IBM Blockchain Platform Visual Studio Code Extension:

Added Indexes for Better Performing Queries

Lab Six



IBM

Table of Contents

Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated here.
Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either express or implied. In no event, shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.
IBM products are manufactured from new parts or new and used parts.
In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."
Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.
Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.
References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.
Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.
It is the customer's responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.
Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of

1 Overview of the lab 5 environment and scenario

This lab is a technical introduction to blockchain, specifically smart contract development using the latest developer enhancements in the Linux Foundation's Hyperledger Fabric v1.4 and shows you how IBM's Blockchain Platform's developer experience can accelerate your pace of development.

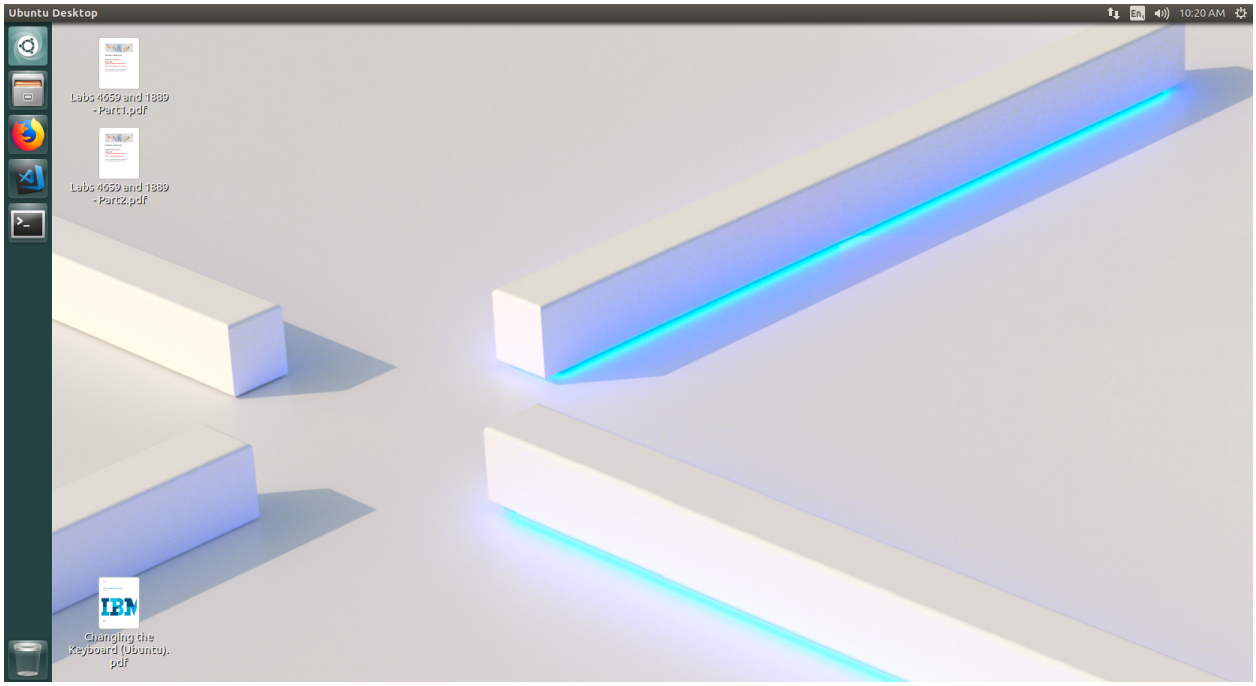
***Note**: The screenshots in this lab guide were taken using version **1.31.1** of **VSCode**, and version **0.3.0** of the **IBM Blockchain Platform** plugin. If you use different versions, you may see differences those shown in this guide.*

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. The 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.

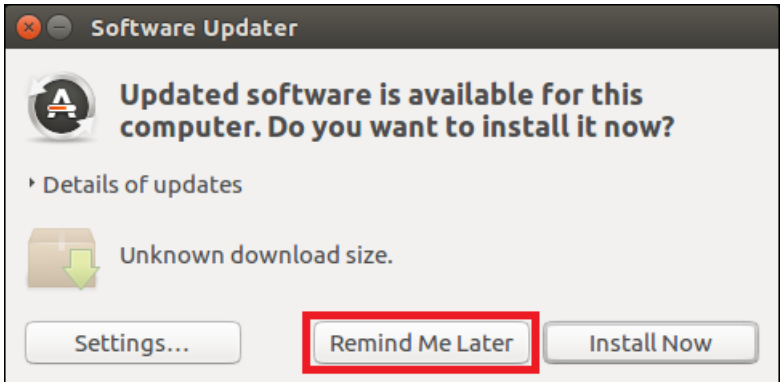
If it asks you to login, the userid and password are both "blockchain".



1.1 Lab 6 Scenario

In this lab, we will import the Commercial Paper sample into VSCode and modify the Smart Contract to create queries to work with CouchDB Indexes which can help improve performance.

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



2 Lab 6: Import Commercial Paper Sample

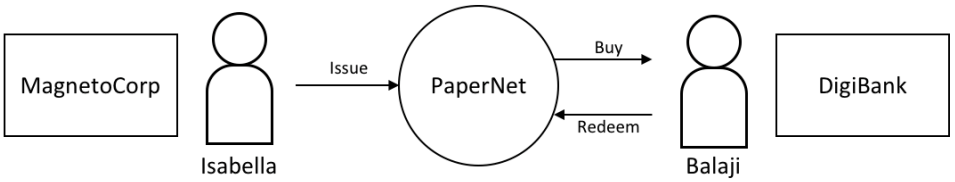
As mentioned above, this lab will be using the Hyperledger Fabric "Commercial Paper" tutorial. The full version of this tutorial is available online and we will be using a simplified version of it.

The scenario the tutorial follows is one of a commercial paper trading network called **PaperNet**. Commercial paper itself is a type of unsecured lending in the form of a "promissory note". The papers are normally issued by large corporations to raise funds to meet short-term financial obligations at a fixed rate of interest. Once issued at a fixed price, for a fixed term, another company or bank will purchase them at a discount to the face value and when the term is up, they will be redeemed for their face value.

As an example, if a paper was **issued** at a face value of 10M USD for a 6-month term at 2% interest then it could be **bought** for 9.8M USD (10M – 2%) by another company or bank who are happy to bear the risk that the issuer will not default. Once the term is up, then the paper could be **redeemed** or sold back to the issuer for their full face value of 10M USD. Between buying and redemption, the paper can be bought or sold between different parties on a commercial paper market.

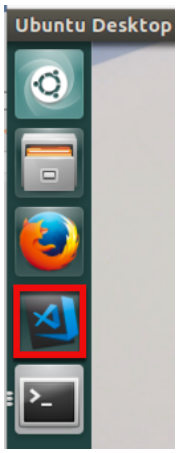
These three key steps of, **issue**, **buy** and **redeem** are the main transactions in a simplified commercial paper marketplace, which we will mirror in our lab. We will see a commercial paper **issued** by a company called MagnetoCorp and once issued on the PaperNet blockchain network, another company called DigiBank will first **buy** the paper and then **redeem** it.

In diagram form it looks like this:

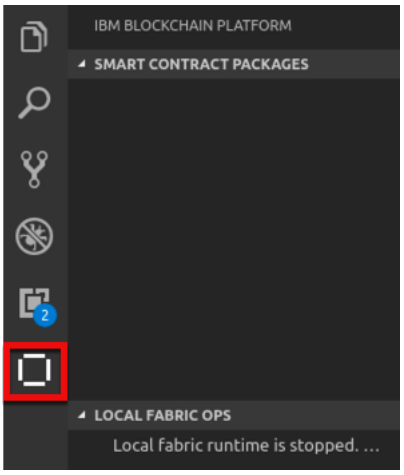


So, let's begin!

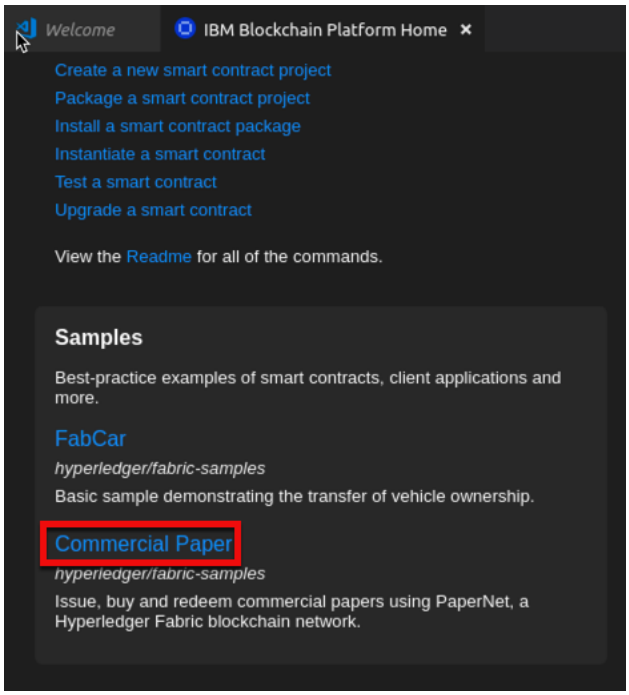
__ 3. Launch VSCode by clicking on the VSCode Icon in the toolbar.



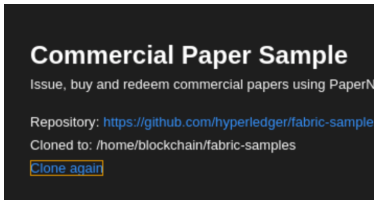
__ 4. When VSCode opens, click on the IBM Blockchain Platform (IBP) icon in the Activity Bar in VSCode as shown below.



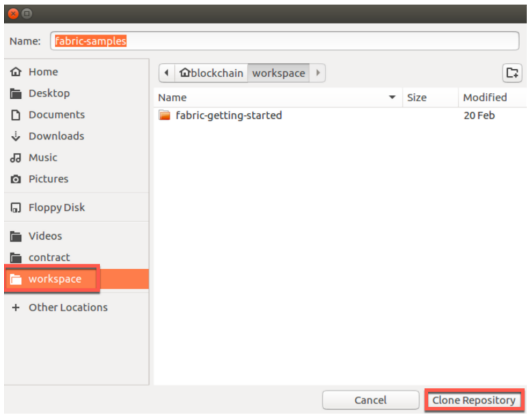
__ 5. Navigate to the IBM Blockchain Platform Home page and click the **Commercial Paper** link under Samples.



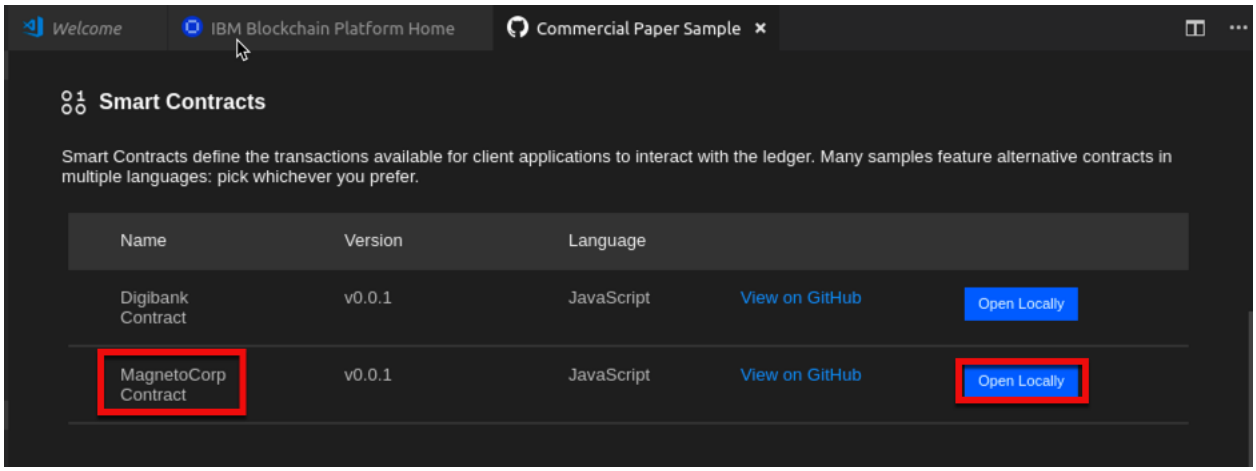
__ 6. The Commercial Paper Sample tab opens. Clone the samples repository by clicking on the **Clone** or **Clone again link** button as shown below.



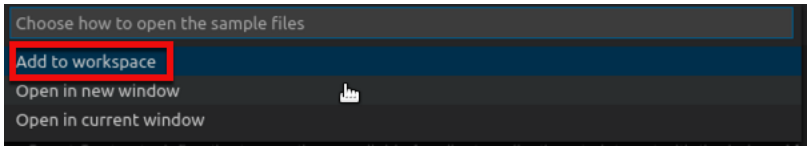
__ 7. At the next panel, select the workspace folder and click the **clone repository** button.



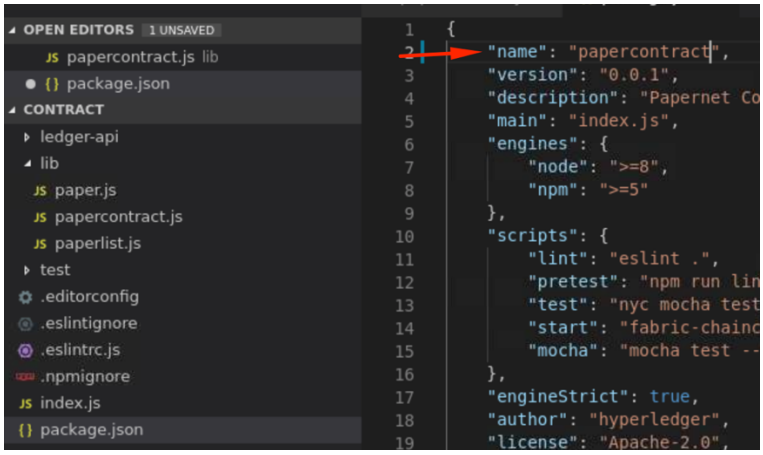
__ 8. Scroll down in the Commercial Paper Sample page and under Smart Contracts, click **Open Locally** next to MagnetoCorp Contract.



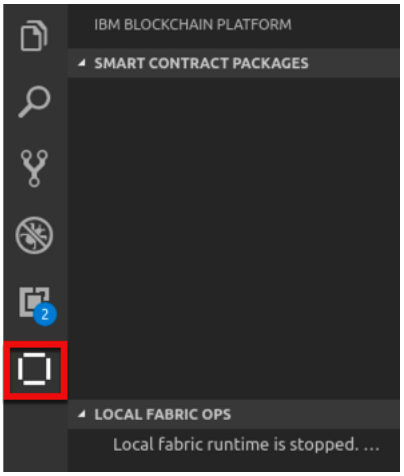
__ 9. Select **Add to workspace** at the Choose how to open the sample files prompt.



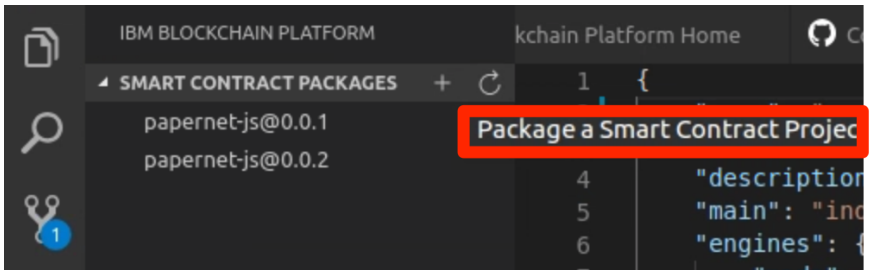
__ 10. Select the Explorer icon in the Activity Bar in VSCode as shown below, open the **package.json** file. Modify the name to papercontract and save the file.



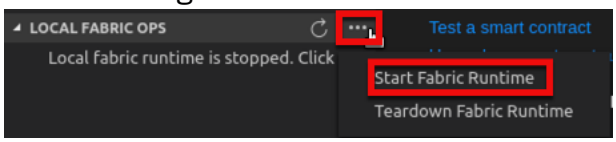
__ 11. Click on the IBM Blockchain Platform (IBP) icon in the Activity Bar in VSCode as shown below.



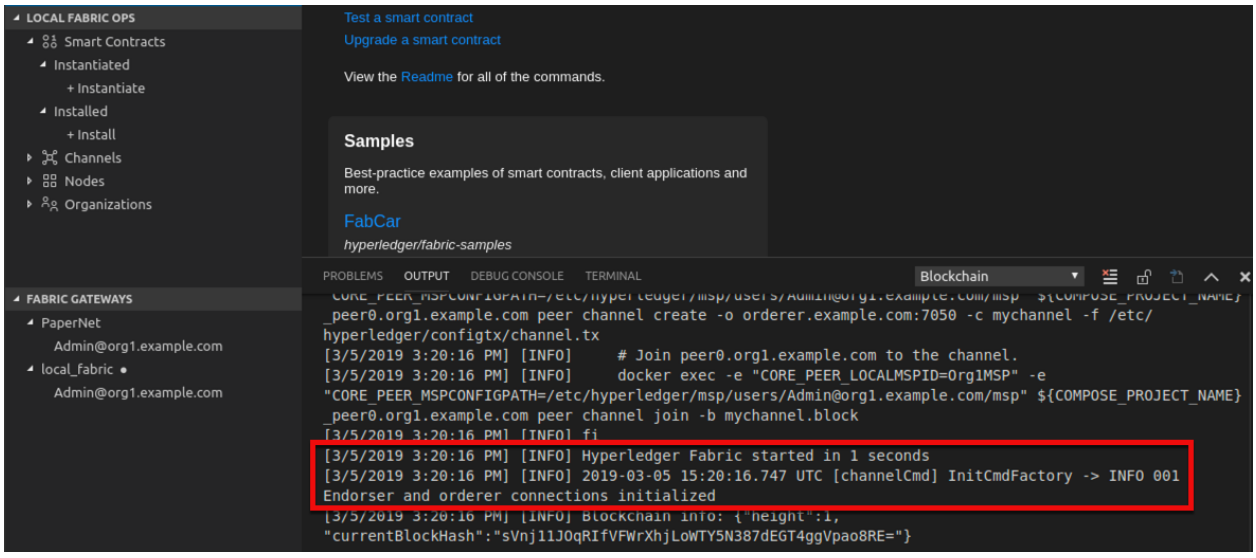
__ 12. From the IBM Blockchain Platform Extension view, Click on the + symbol next to SMART CONTRACT PACKAGES(Add a new package), to package up the commercial paper smart contract package for installing onto a peer. It will be called something like papercontract@0.0.1. You may have previous packages from an earlier lab.



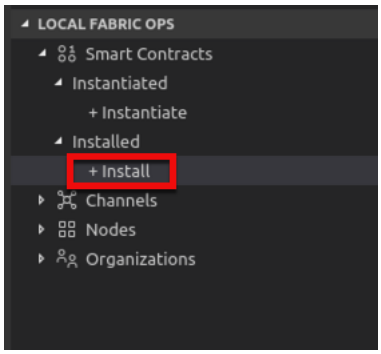
__ 13. Start the local_fabric by hovering over the ... next to LOCAL FABRIC OPS and selecting Start Fabric Runtime.



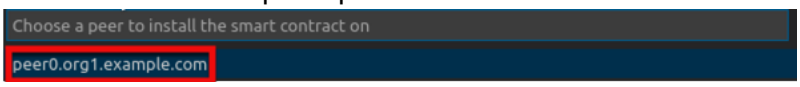
__ 14. The local_fabric runtime is successfully started when you see the following messages in the console pane. You are automatically connected to the local_fabric runtime.



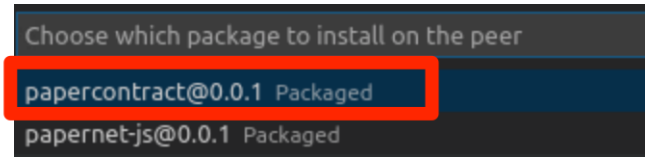
__ 15. Now we will install the Smart Contract. Under LOCAL FABRIC OPS, click **+ Install**.



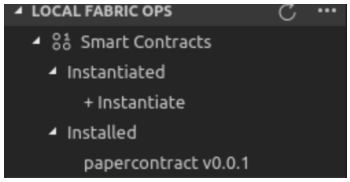
__ 16. Select **peer0.org1.example.com** at the Choose a peer to install the smart contract on prompt.



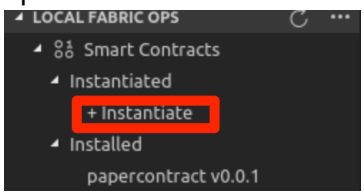
__ 17. Select the newly created **papercontract@0.0.1** at the Choose which package to install on the peer prompt.



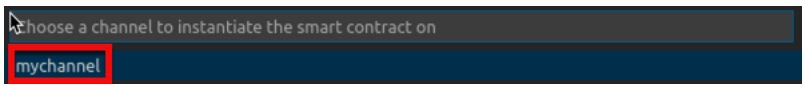
__ 18. You should now see the installed contract



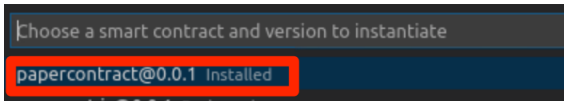
__ 19. Instantiate the newly installed contract by clicking the **+Instantiate** option.



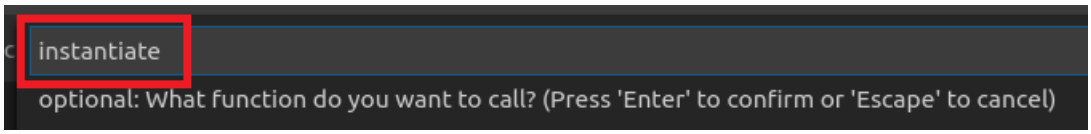
__ 20. Select **myChannel** at the Choose a channel to instantiate the smart contract on prompt.



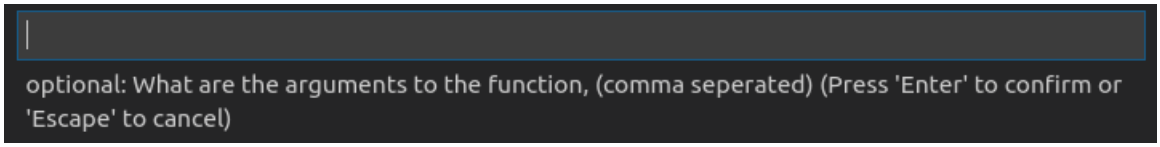
__ 21. Select the newly installed papercontract contract.



__ 22. In the pop-up dialogue box at the top of the screen asking "**optional: What function do you want to call?** ..." make sure you enter the word **instantiate** into the entry field as shown below. Before you press enter, check your spelling and make sure it is correct and is all lowercase without any quotes or spaces around it. This name has to exactly match the name of the transaction in your contract that will be called at instantiate time and in our default contract as we saw above this is called **instantiate**.

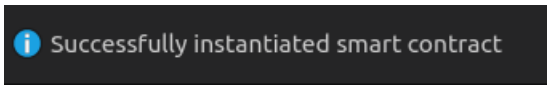


__ 23. In the next dialogue that asks for parameters to the function, just press "**Enter**" as our **instantiate** function does not require any apart from the context "**ctx**" which is automatically provided by the framework.



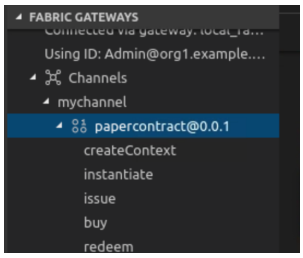
Instantiating a contract can take several minutes as a new docker container is built to contain the contract. Whilst it is happening you should see this information message

When it is complete you will see this information message



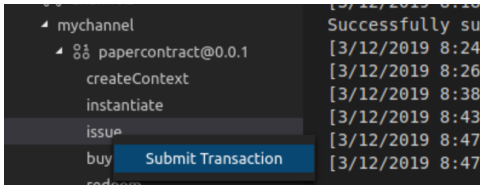
Once complete, the "**LOCAL FABRIC OPS**" view under **Instantiated** will change to show the Smart Contract papercontract@0.0.1 to be instantiated.

__ 24. The instantiated contract should now show under our channel under the Gateways section.

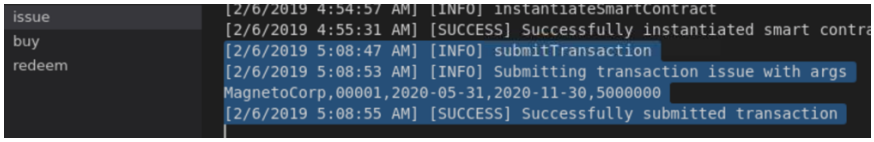


__ 25. Run an **issue** transaction. Right click on **issue** method and select **Submit Transaction**. Use the following data for the method arguments and do not use any spaces between arguments:

```
MagnetoCorp,00001,2020-05-31,2020-11-30,5000000
```

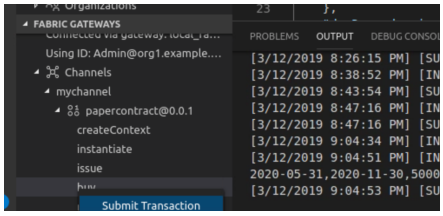


The screenshot below shows a successful completion of issue transaction. Issued transaction

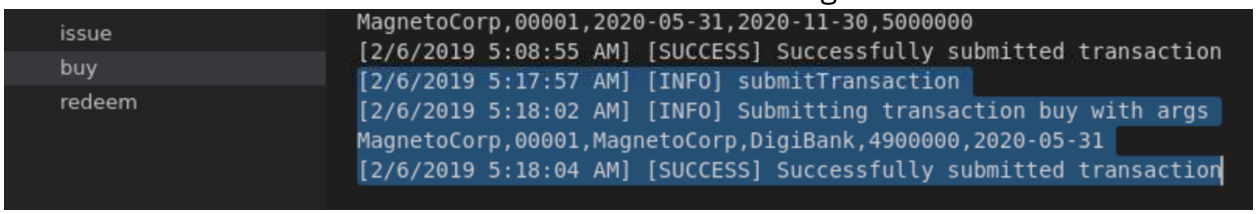


__ 26. Run the **buy** transaction. Right click on the **buy** method and select **Submit Transaction**. Use the following data for the method arguments:

```
MagnetoCorp,00001,MagnetoCorp,DigiBank,4900000,2020-05-31
```



A successful transaction should look similar to the image below.



__ 27. In the next section we'll start working on queries...

Article I. Querying the World State

In this section we will take a look at how you can query the world state of a channel within Hyperledger Fabric. Before we jump into showing how to query the world state, we first need to understand what database indexes are and how they can help us with queries. Also note that Hyperledger Fabric currently supports two different world state database implementations. One implementation uses leveldb, the other implementation uses couchdb which provides richer query capabilities. Factors regarding which implementation to use often are based on performance versus those richer query options.

Article II. What are database indexes?

In order to understand indexes, let's take a look at what happens when you query the world state. Say, for example, you want to find all assets owned by the user, "Bob". The database will search through each json document in the database one by one and return all documents that match user = "bob. This might not seem like a big deal but consider if you have millions of documents in your database. These queries might take a while to return the results as the database needs to go through each and every document. With indexes you create a reference that contains all the values of a specific field and which document contains that value. What this means is that instead of searching through every document, the database can just search the index for occurences of the user "bob" and return the documents that are referenced.

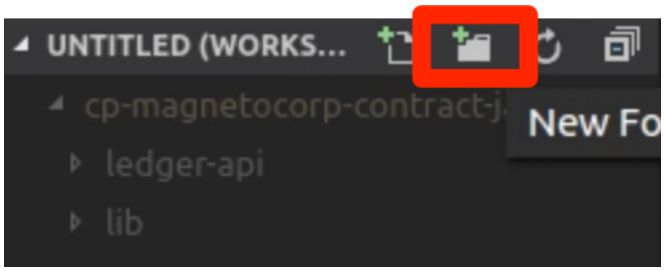
It's important to note that every time a document is added to the database the index needs to be updated. Normally in CouchDB this is done when a query is received but in Hyperledger Fabric the indexes are updated every time a new block is committed which allows for faster querying. This is a process known as **index warming**.

1. Think of different queries that are needed for the application to function The first step in building an index is understanding what queries are commonly run. Once we understand what queries are important to the application, then we can decide on what fields to include in the index.

In the commercial paper use case we will be querying by issuer, by owner, and by the current state of each asset.

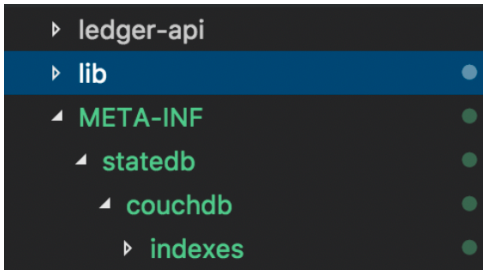
Article III. Create indexes for those commonly used queries

1. From the VSCode Explorer view, create a directory and name the new directory **META-INF**.



2. Then, in the new directory, create another new directory named **statedb**
3. After that, create a new directory inside of **statedb** called **couchdb**
4. Next, you guessed it, create a new directory inside of **couchdb** and name it **indexes**

The directory structure should look like the image below.



1. Now we can start creating our index definitions. Create a new file in the **indexes** directory and name it **issuerIndex.json**
2. Then, copy the following code into that file:

```
{
    "index": {
        "fields": [ "issuer"]
    },
    "ddoc": "issuerIndexDoc",
    "name": "issuerIndex",
    "type": "json"
}
```

This file states that the index will:

- keep track of the *issuer* field of each document

- store this index in a design document (ddoc) named *issuerIndexDoc*
- is named issuerIndex
- will be in json format

Now let's create two more.

3. Create a new file in the **indexes** directory and name it **ownerIndex.json**
4. Then, copy the following code into that file:

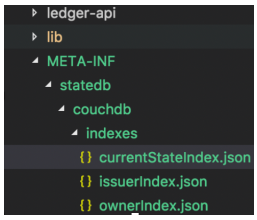
```
{
    "index": {
        "fields": ["owner"]
    },
    "ddoc": "ownerIndexDoc",
    "name": "ownerIndex",
    "type": "json"
}
```

This index is very similar to the previous one for the issuer field but instead we are indexing the *owner* field.

5. Finally, create one last file in the **indexes** directory and name it **currentStateIndex.json**
6. Then, copy the following code into that file:

```
{
    "index": {
        "fields": [ "currentState"]
    },
    "ddoc": "currentStateIndexDoc",
    "name": "currentStateIndex",
    "type": "json"
}
```

Your directory structure should now look like this:



And that's all it takes to build indexes. These indexes will be deployed next time the smart contract is installed and instantiated.

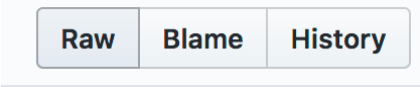
(i) Implement query transactions in the smart contract

Now we need to implement the query logic in the transactions of the smart contract. These transactions will be invoked by the Node SDK to execute our queries.

1. If you haven't already, open this repo in your browser within the VM. Do this by going to http://www.github.com/rojanjose/commpaper or the http://www.github.com/jeffet/commpaper repo.

2. In Github, click on the **papercontract.js** file to view it.

3. Then, click on the **Raw** button



4. Once in the raw view, copy everything in this file. It's easy if you use the *control + A* shortcut to copy all.

5. Then, switch back to the VS Code editor and open **lib/papercontract.js** file.

6. Delete everything in this file and paste in the version you copied from Github.

This updated contract already has the query logic added. Let's take a look at the transactions that were added. You can find the transactions starting at approximately line 155 of the code.

- queryByIssuer, queryByOwner, and queryByCurrentState - These transactions are all similar in that they take one parameter and query the respective fields in the database. If you look at the *queryString* for each transaction, you will notice that they are pointing to the design documents that hold the indexes that were created earlier. This query string is then passed to *queryWithQueryString* to be executed.

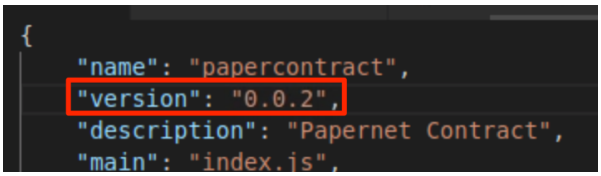
- queryAll - This transaction does what it says. It gets all asset states from the world state database. This query string is then passed to *queryWithQueryString* to be executed.

- queryWithQueryString - This function receives a query string as a parameter and is called by other transactions in the contract to do the actual querying. You can also do ad hoc queries with this transaction by passing in your own query strings.

(ii) Upgrading the deployed contract

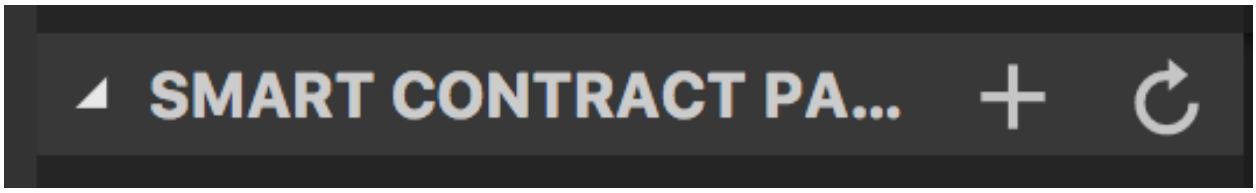
Since we made changes to the smart contract we now need to re-deploy it to the peer.

1. Open up **package.json** in VSCode
2. Change the *version* property to **0.0.2** and save the file.

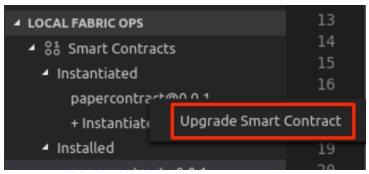


3. Click on the IBM Blockchain extension icon on the left side of VSCode.
4. Package the contract again by clicking on the plus icon at the top of the *Smart Contract Packages* section on the upper left side.

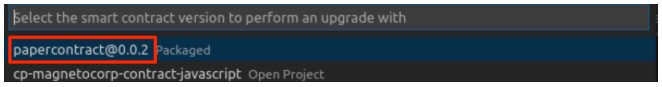
If necessary, specify to create the package from the *contract* workspace.



5. If you haven't already, click on *PaperNet* under the *Blockchain Connections* section in the lower left in the IBM Blockchain platform extension.
6. Then click on *mychannel* to expand the contents of the channel.
7. Under LOCAL FABRIC OPS under Smart Contracts under Instantiated right click papercontract v0.0.1 and select *upgrade smart contract*



In the dialog that appears, select the newly packaged papercontract 0.0.2.

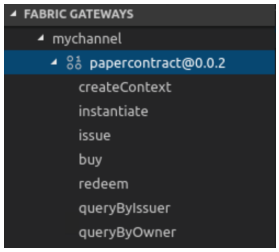


8. Install onto peer0.org1.example.com

When asked about what function you'd like to call, enter **instantiate**

Then when it asks for arguments to pass, just press enter without typing anything.

10. The Upgrade may take a few minutes. If successful, you should now see **papercontract@0.0.2** in the bottom left pane under FABRIC GATEWAYS -> Channels -> *mychannel*. Notice the new query functions.



Article IV. Querying the world state

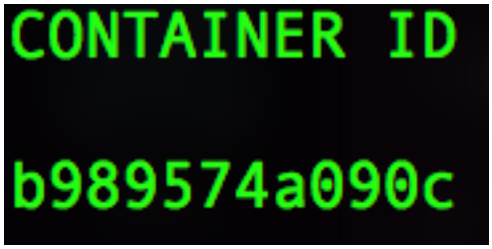
Before we start querying, there's a few quick things we need to do.

1. Open a new terminal window using the left hand toolbar or use the terminal view in VSCode.

2. From the new terminal, run the following command to list all running Docker containers

```
docker ps
```

3. Then, copy the container ID for the very first entry.



4. Next, run the following command while replacing "container ID" with the actual container ID that you copied.

```
docker logs -f "container ID"
```

For example, using the container ID from the screenshot above I would enter this:

```
docker logs -f b989574a090c
```

5. This will allow us to see the outgoing logs from our chaincode container. This will allow you to debug the transactions or, in our case, view the results of the queries.

6. Leave the terminal window open in the background and go back to VS Code.

7. From the IBM Blockchain extension, under FABRIC GATEWAYS ->Channels -> mychannel -> click on **papercontract@0.0.2** if it isn't already expanded to show all the transaction methods contained in the chaincode contract.

8. You should see our new query methods.

9. Let's issue another commercial paper. Right click on **issue** and click **submit transaction**

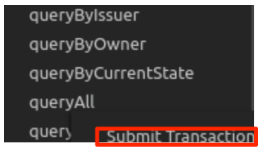
10. For the arguments, enter the following (again with no spaces):

```
MagnetoCorp,0003,2019-02-13,2020-02-13,5000000
```

11. Press enter

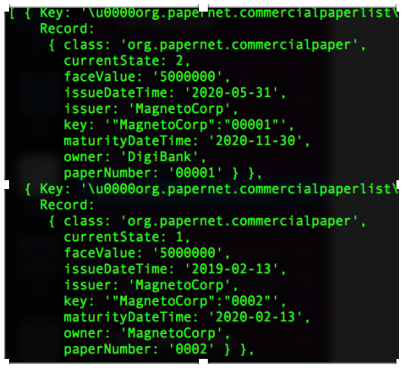
Now we can begin querying

12. Now right click on **queryAll** from the bottom left of the IBM Blockchain extension and click on **submit transaction**.



13. This transaction doesn't take any arguments so you can just press enter again.

When submitted, check out your terminal window that we left open. It should now show the results of our query.

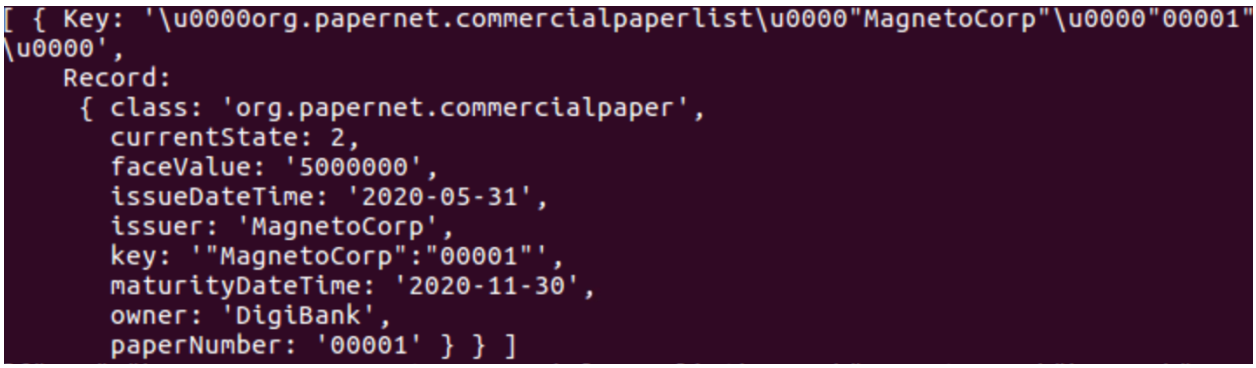


Since this query simply returns everything in the world state, let's try a different query.

14. Right click on **queryByOwner** and enter the following for the argument:

```
DigiBank
```

Take a look at the logs and see what the output is. Notice that only one document is returned this time. This is because only one asset is owned by DigiBank. This is the asset that we ran the **buy** transaction on earlier which represented DigiBank buying the commercial paper from MagnetoCorp.



15. Next let's right click on **queryByCurrentState** and enter the number **1** as the only argument.

Take a look at the logs again. Only one document is returned again but this time it's for the asset that we just issued. This is because the currentState value of **1** indicates those assets that have been issued but not yet bought.

```
[ { Key: '\u0000org.papernet.commercialpaperlist\u0000"MagnetoCorp"\u0000"0003"\
u0000',
    Record:
     { class: 'org.papernet.commercialpaper',
       currentState: 1,
       faceValue: '5000000',
       issueDateTime: '2019-02-13',
       issuer: 'MagnetoCorp',
       key: '"MagnetoCorp":"0003"',
       maturityDateTime: '2020-02-13',
       owner: 'MagnetoCorp',
       paperNumber: '0003' } } ]
```

Article V. Recap of querying

In this section we took a look at how querying works in a Hyperledger Fabric network with CouchDB as the state database. First, we created indexes for commonly used queries. Then, we added the query logic to the smart contract. Finally, we ran some queries and took a look at what the world state contained. Thanks for checking it out!