

IBM Blockchain Hands-On

IBM Blockchain Platform Visual Studio Code Extension:

Developing your First Contract

Lab Two

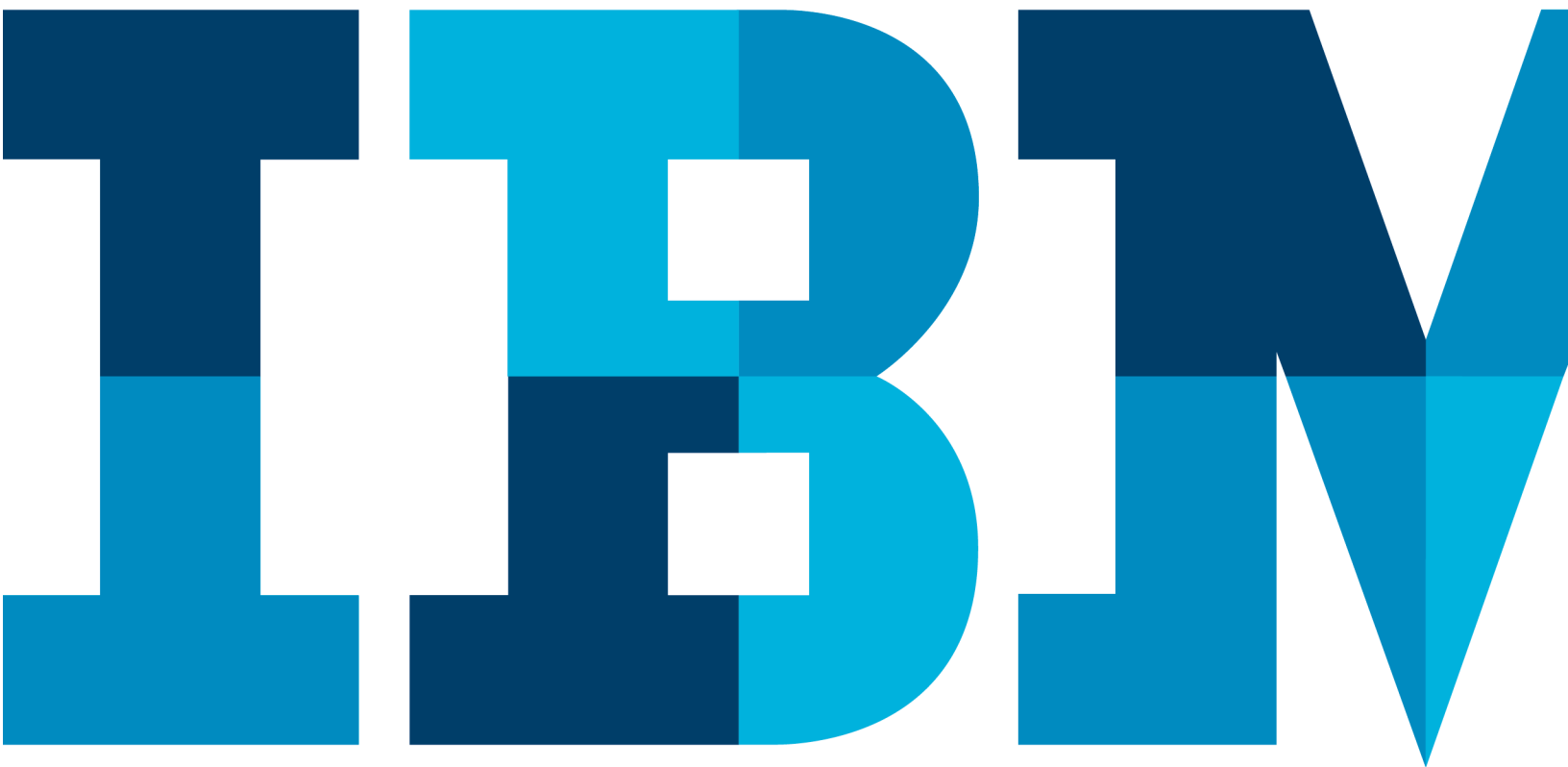


Table of Contents

Disclaimer	3
1 Overview of the lab 2 environment and scenario	5
1.1 Lab 2 Scenario.....	6
2 Lab 2: An overview of the VSCode development experience.....	7

Disclaimer

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

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

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

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

IBM products are manufactured from new parts or new and used parts. In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply."

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

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

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

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and

discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**

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

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

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

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

1 Overview of the lab 2 environment and scenario

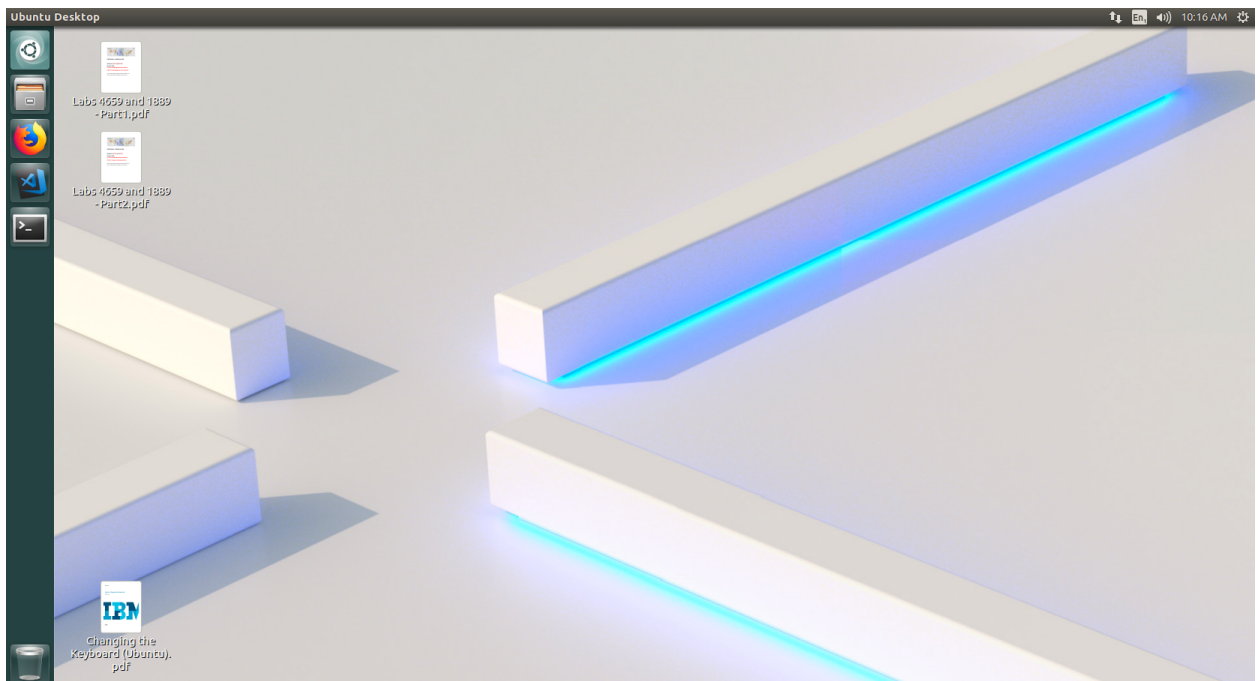
This lab is a technical introduction to blockchain, specifically smart contract development using the latest developer enhancements in the Linux Foundation’s Hyperledger Fabric v1.4 and shows you how IBM’s Blockchain Platform’s developer experience can accelerate your pace of development.

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 2 Scenario

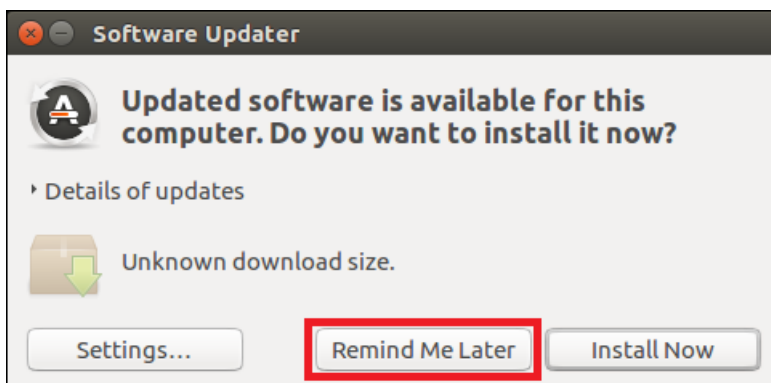
This lab will take you through using the smart contract development environment in Visual Studio Code (VSCode). Although smart contracts can be developed in any editor, IBM Blockchain Platform provides a plugin for VSCode that greatly simplifies the steps required. In addition, it also provides a “sandbox” development environment for easy development and test purposes using a real Hyperledger Fabric runtime.

In lab3, we will take you through using a sample smart contract that comes with Hyperledger Fabric through the VSCode lens, where you will learn how to import contracts and interact with the development environment in more detail.

In lab 4, you will use VSCode to connect to another Hyperledger Fabric sample network, this time running outside of VSCode and learn how to interact with an external network in order to test smart contracts belonging to an existing network.

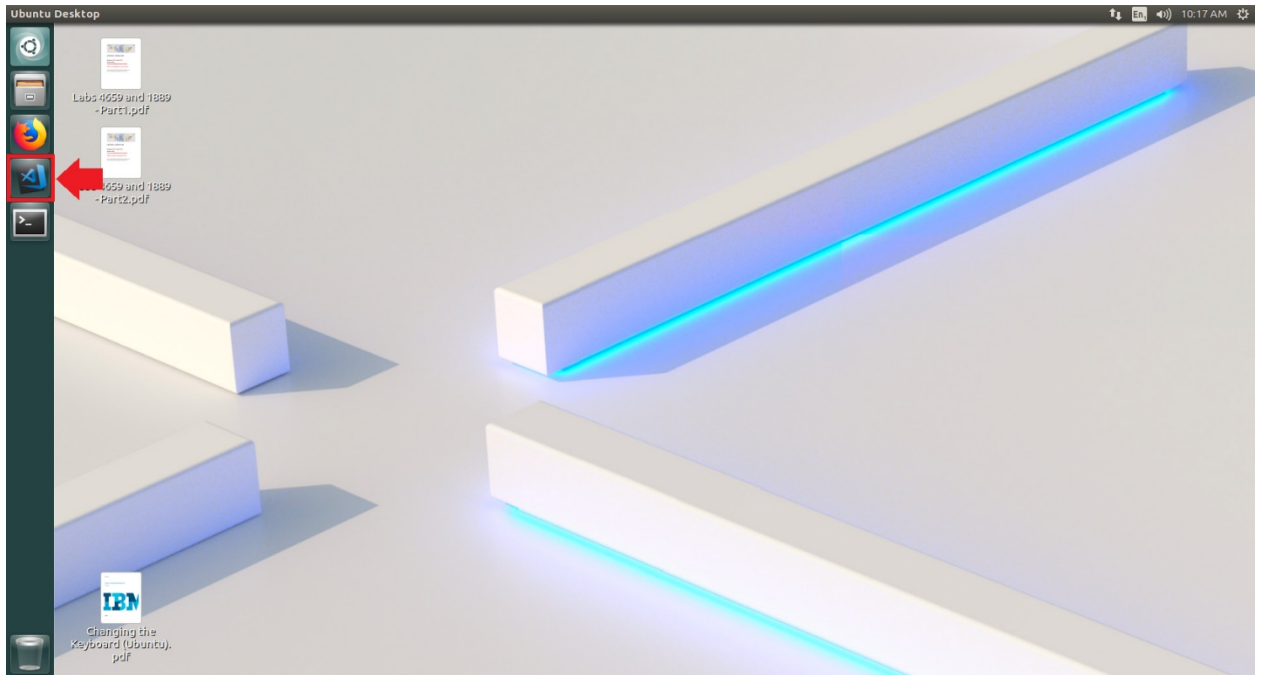
In lab 5, you will use VSCode to import the same Commercial Paper sample provided in lab 4, but will use all the features in the IBM Blockchain Platform extension and local_fabric embedded instance to deploy, upgrade and extend smart contracts and generate functional tests.

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



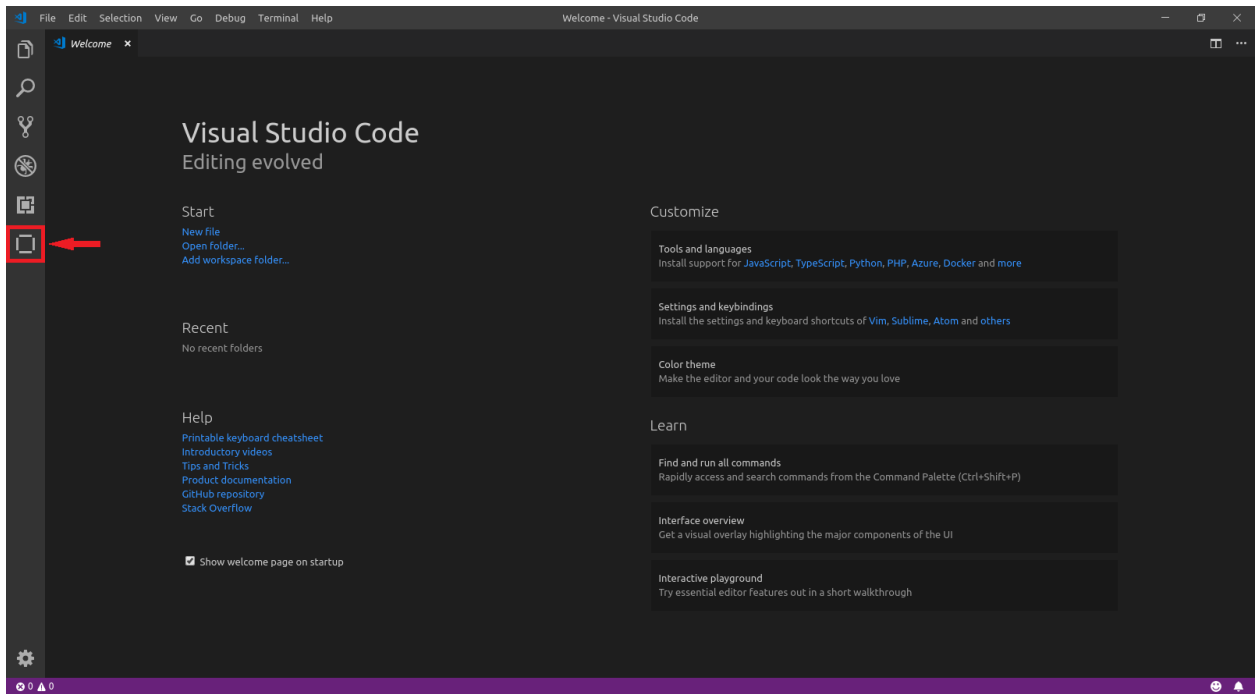
2 Lab 2: An overview of the VSCode development experience

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



IBM Blockchain

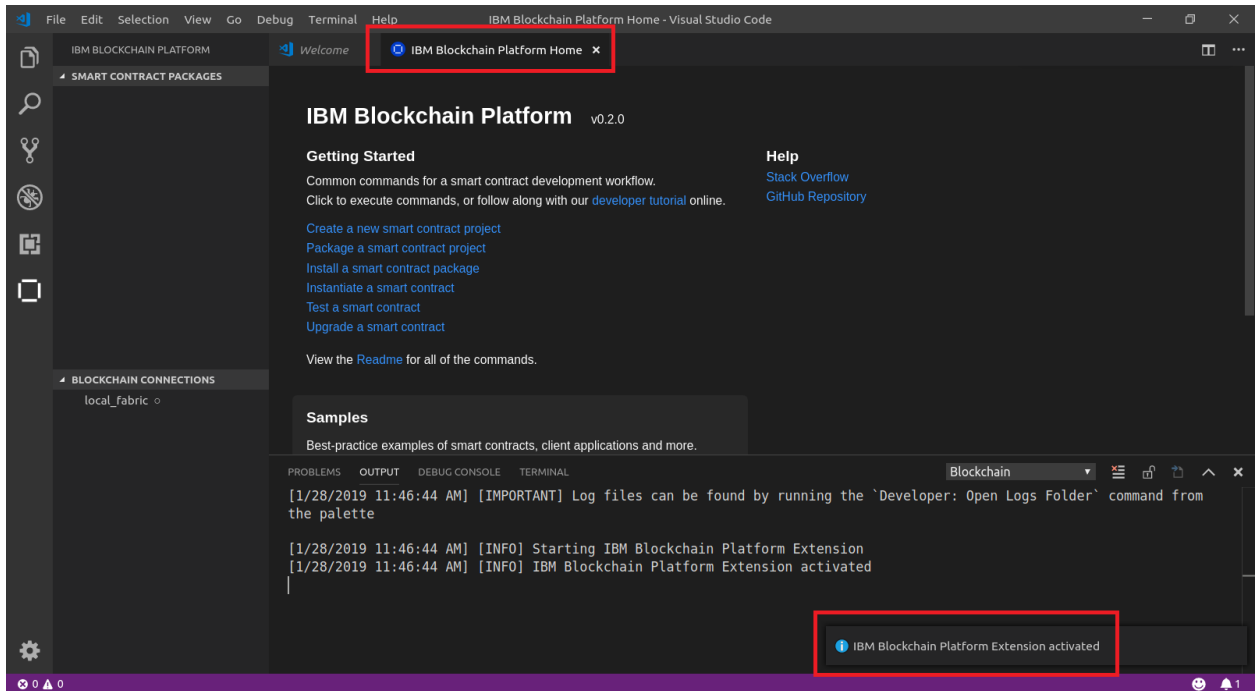
- ___ 4. When VSCode opens, click on the IBM Blockchain Platform (IBP) icon in the Activity Bar in VSCode as shown below.



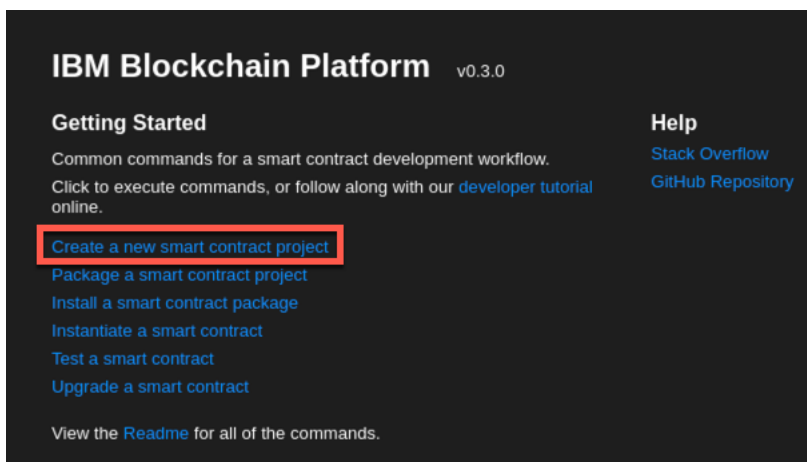
This will open the Homepage for IBM Blockchain Platform. As highlighted below, there will be a message in the bottom right telling you the extension has activated and other information telling you some of the capabilities of the plugin. These informational messages are used a lot in VSCode and will appear throughout this lab at various points.

If you close the Homepage by mistake and need to get back to it, you can open it again by pressing **“ctrl + shift + p”** to open the VSCode command palette and typing the word **home** in the dialogue box to find the **“IBM Blockchain Platform: View Homepage”** command.

IBM Blockchain

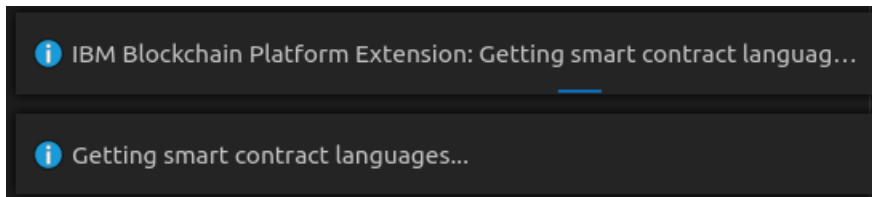


- ___ 5. Once you have had a look at the Homepage, click on the “**Create a new smart contract project**” link toward the top of the Homepage.

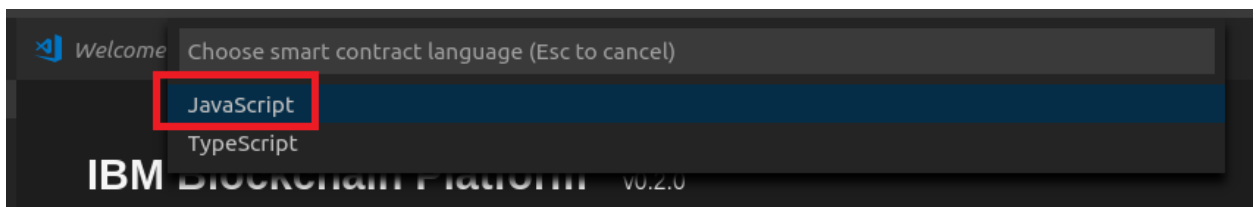


IBM Blockchain

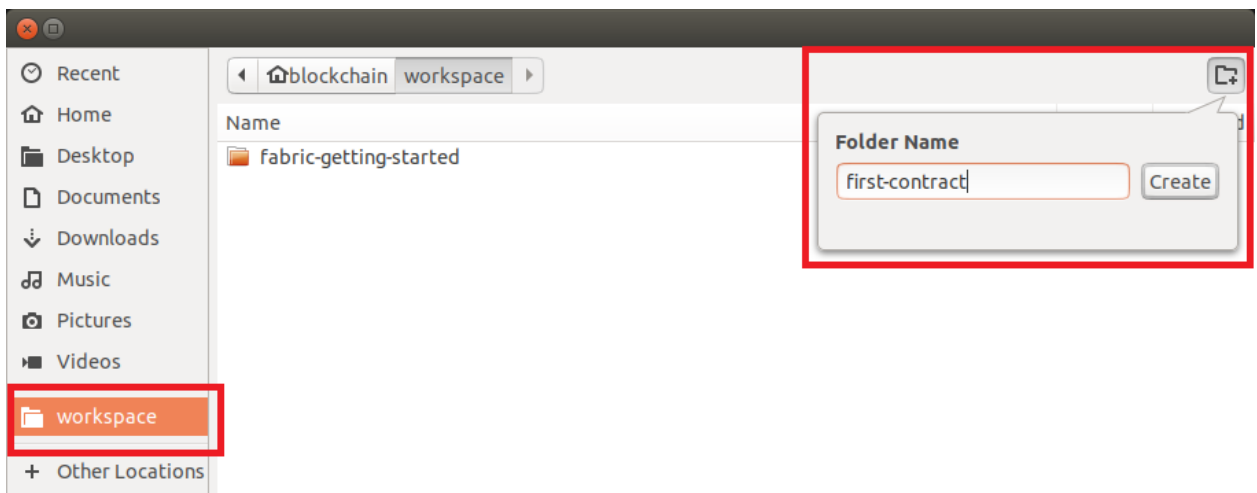
- ___ 6. Notice a couple of informational messages will appear in the bottom right of the VSCode window. You can dismiss these once you have read them:



- ___ 7. After a little delay, at the top of the screen, you will be presented with a choice of languages. For this lab we are going to use JavaScript, so click to choose the JavaScript option:

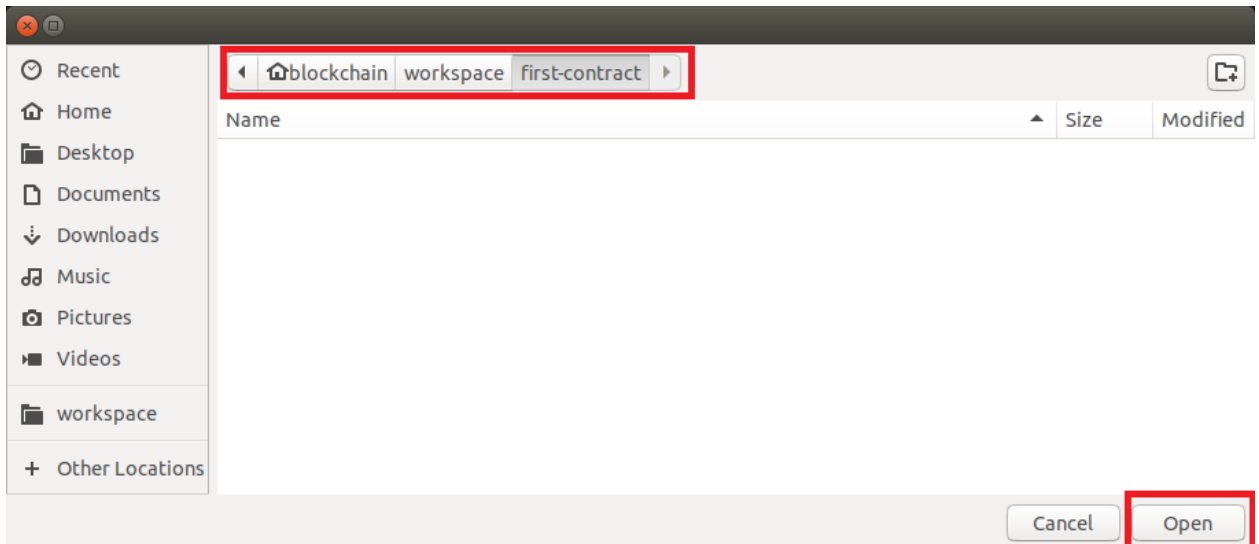


- ___ 8. In the dialogue that appears, first choose **workspace** in the left-hand bar, to change into the **workspace** folder, then click the create folder icon in the top right (📁), enter the name **first-contract** as the "folder name" and click **Create**.

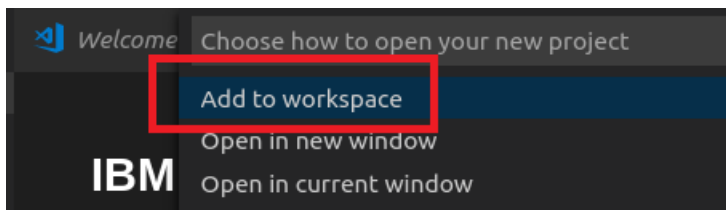


IBM Blockchain

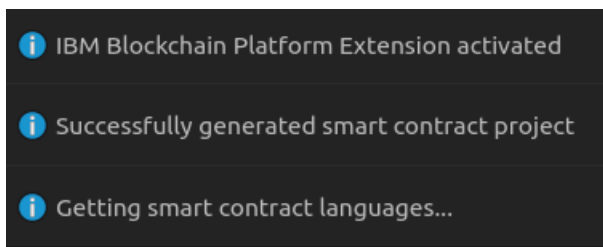
- ___ 9. After making sure you are in the /blockchain/workspace/first-contract folder, click **Open** in the bottom right.



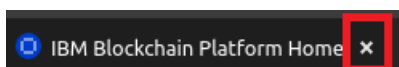
- ___ 10. Next, select the **“Add to workspace”** option that appears at the top of the VSCode window.




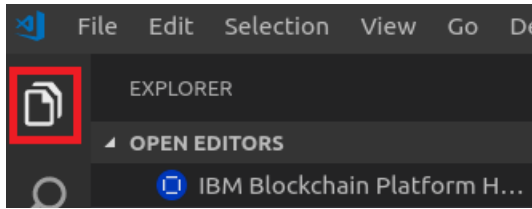
- ___ 11. Wait for the information messages to appear whilst the project is generated as this can take up to a minute or more to complete.



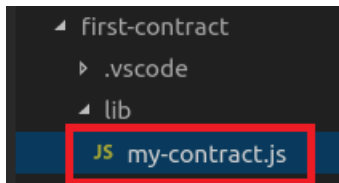
- ___ 12. You may notice that another copy of the IBP Homepage opens at this point. This can safely be closed by clicking on the **“x”** on the window.



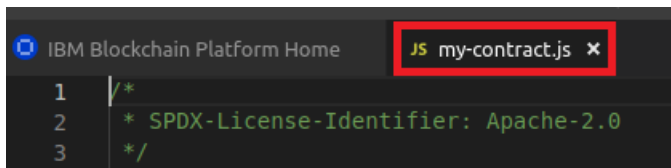
- ___ 13. We can now switch to the Explorer view briefly to see the files generated by the plugin for the new smart contract. To do this you can either press “**ctrl + shift + e**” on your keyboard or choose the top left folder shaped icon in the activity bar ().



- ___ 14. Click to expand the **lib** folder and open the **my-contract.js** file by double-clicking on it. This is the main file for the contract.



When the **my-contract.js** file is open, you should see it in the main window to the right.



- ___ 15. Take a look at the contents of this file. As you can see, the main body of the code is on lines 7 – 23. These define a very simple smart contract that does not make any updates to the ledger, it only echoes a string when a transaction function is called.

A more detailed overview of the code is as follows:

Line 7 imports a **Contract** definition from the **fabric-contract-api** node module. This makes the Fabric API available to the smart contract to use.

Line 9 starts the definition of a class called **MyContract** that extends the **Contract** we imported above. This provides our class with several capabilities such as defining our class to be a contract that can be called by the framework and giving us access to transaction handlers and a transaction based **context** called **ctx**. The context allows the framework to pass extra information into the transaction function when it is called. For example it can pass information about the identity of the caller of the contract as well as methods to query the world-state when the transaction is called.

Line 11 shows the definition of an **instantiate** method which can be used to initialise the contract when it is first deployed or after an upgrade. By default, this method takes a single context argument called **ctx** which is a context as described above. If required, it could take extra parameters as well.

Line 15 defines a transaction function called “transaction1”. All transaction functions must take a context as their first parameter, usually called “ctx”. This is normally followed by one or more arguments that are passed to the transaction from the client.

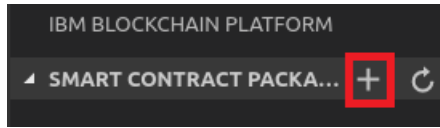
Line 19 defines a second transaction function called “transaction2” which this time takes two parameters after the context.

```
7  const { Contract } = require('fabric-contract-api');
8
9  class MyContract extends Contract {
10
11     async instantiate(ctx) {
12         console.info('instantiate');
13     }
14
15     async transaction1(ctx, arg1) {
16         console.info('transaction1', arg1);
17     }
18
19     async transaction2(ctx, arg1, arg2) {
20         console.info('transaction2', arg1, arg2);
21     }
22
23 }
24
25 module.exports = MyContract;
```

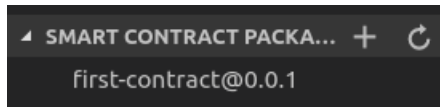
- __ 16. Next, we will package the smart contract. Click on the IBP icon in the sidebar to switch back to the IBM Blockchain Platform view.



- __ 17. From the **Smart Contract Packages** view click the “+” icon to package the smart contract into a deployment package. If you do not see the “+”, first click in the **Smart Contract Packages** view.

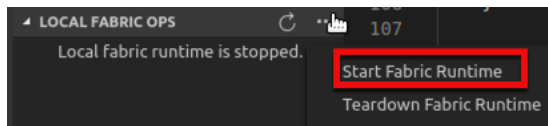


You will first see an informational message about packaging the contract, then you will see a package appear after it is created, called **first-contract@0.0.1**

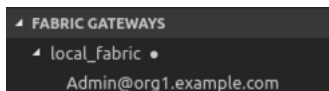


This package is now ready to be installed onto a blockchain peer.

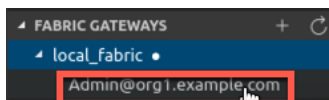
- __ 18. In the **LOCAL FABRIC OPS** view, click on the ... and select **Start Fabric Runtime**.

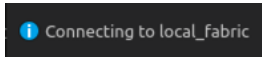


The circle icon next to **local_fabric** under **FABRIC GATEWAYS** may appear to spin and text will appear in the **Output** window to show progress. Note that this may take a little time to complete.

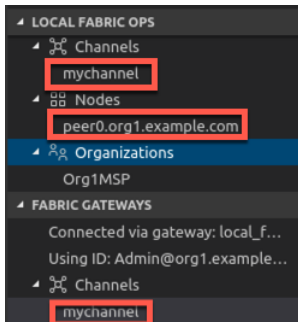


- __ 19. Once the text “[channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel” appears in the **Output** window and the **local_fabric** circle icon is solid, click on admin@org1.example.com under **local_fabric** in the **FABRIC GATEWAYS** view to connect. At this point there will be an information message confirming that the connection has been made:





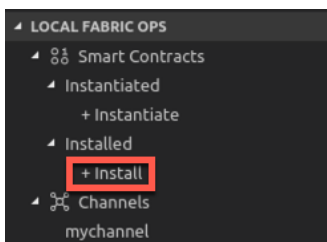
- ___ 20. When the connection is made, you should see the channel called **mychannel** appear, both under the **LOCAL FABRIC OPS** view and the **FABRIC GATEWAYS** view. Underneath Nodes in the LOCAL FABRIC OPS view, you will see the peer called **peer0.org1.example.com**.



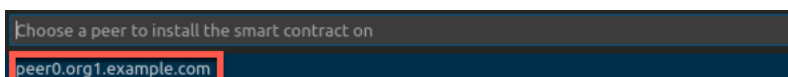
This is the same single-channel, single-peer network that the plugin creates for test and development purposes.

- ___ 21. Hyperledger Fabric requires that contracts are **installed** on a peer and then **instantiated** on a channel before use, so that's what we will do next over the next few steps.

- ___ 22. Click on +Install in the **LOCAL FABRIC OPS** view. You may need to scroll on the view to see this option.

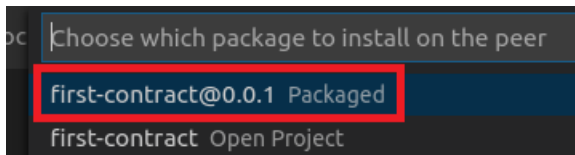


- ___ 23. From the “**Choose a peer to install the smart contract on**” pop up at the top of the screen, choose “**peer0.org1.example.com**” from the options.

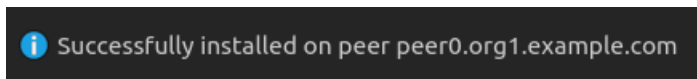


IBM Blockchain

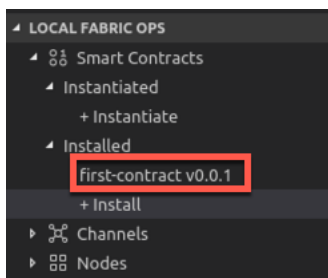
- ___ 24. From the “**Choose which package to install on the peer**” pop up at the top of the screen, choose “**first-contract@0.0.1 Packaged**” from the options.



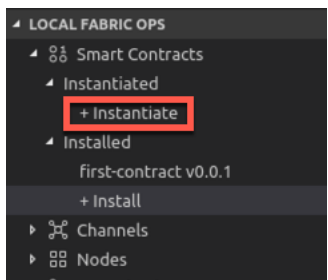
When the package is installed, an information message will be shown confirming the install:



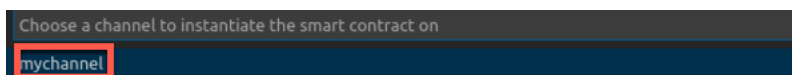
Now under **Installed** in the **LOCAL FABRIC OPS** view you can see the installed contract:



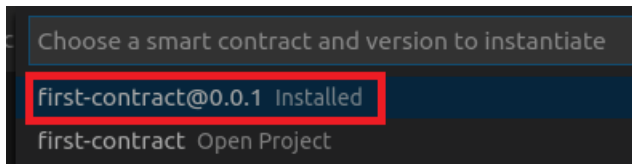
- ___ 25. Next, we have to instantiate the contract. Click on **+Instantiate** in the **LOCAL FABRIC OPS** view.



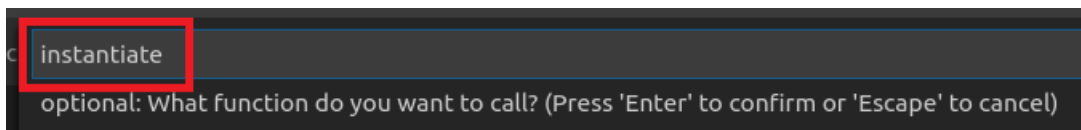
- ___ 26. From the “**Choose a channel to instantiate the smart contract on**” pop up at the top of the screen, choose “**mychannel**” from the options:



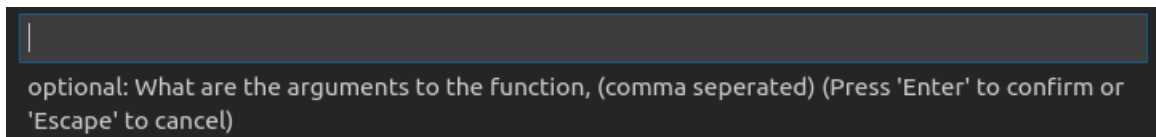
- __ 27. From the “**Choose a smart contract and version to instantiate**” pop up at the top of the screen, choose “**first-contract@0.0.1 Installed**” from the options:



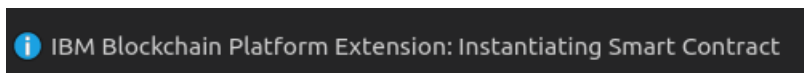
- __ 28. In the pop-up dialogue box at the top of the screen asking “**optional: What function do you want to call? ...**” make sure you enter the word **instantiate** into the entry field as shown below. Before you press enter, check your spelling and make sure it is correct and is all lowercase without 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**.



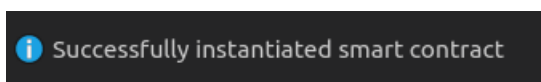
In the next dialogue that asks for parameters to the function, just press “**Enter**” as our **instantiate** function does not require any apart from the context “**ctx**” which is automatically provided by the framework.



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

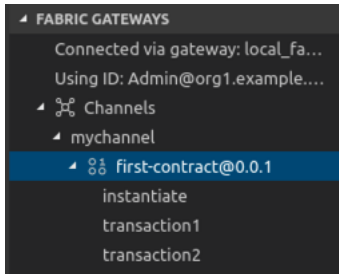


When it is complete you will see this information message



Once complete, in the “**FABRIC GATEWAYS**” view, **mychannel** can be expanded change show **fabcar@1.0.0**.

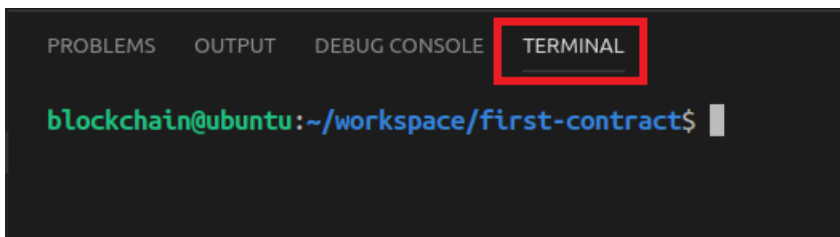
- ___ 29. Expand the instantiated contract **first-contract@0.0.1**, and you will see the three transactions that were defined in the contract are now available.



At this point the contract is now ready to be called. The instantiate transaction has already been run when the contract was instantiated by the framework. This transaction simply prints out the word “instantiate” to the console. Because the contract is running inside a docker container, we need to look at the docker logs to see the output. To see the docker logs we need to get the name of the docker container that is running the contract. To do this we could list all the running containers with “**docker ps**” and look for the right one.

However, as the names of the containers used by Fabric and the IBP plugin are deterministic, we can simple issue the command in the terminal window inside VSCode.

- ___ 30. Switch to the terminal window at the bottom of the VSCode screen

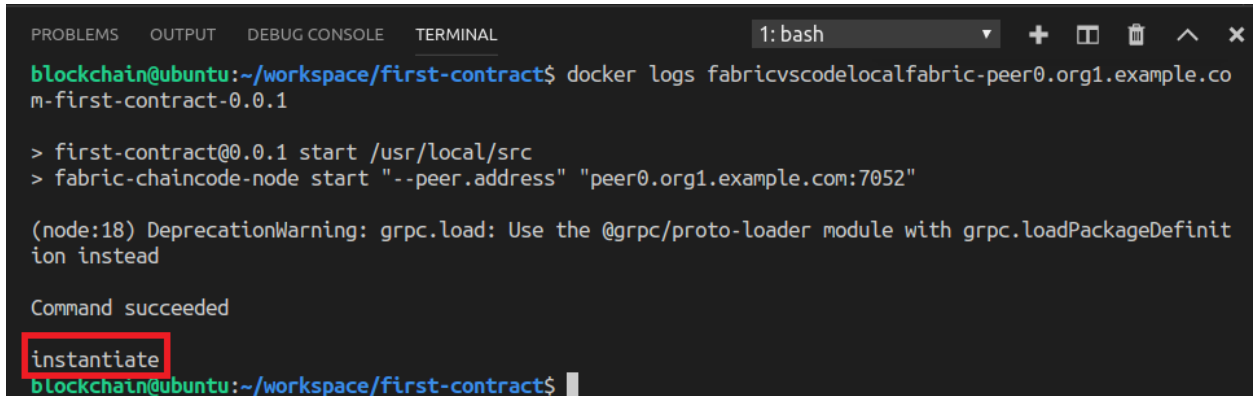


Note that if your window size is small, you might not be able to see the **Terminal** window and you must first click on the ellipsis (...) to allow you to view it.

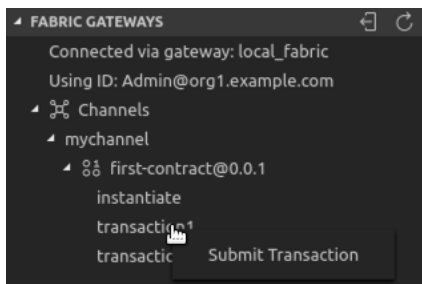
__ 31. At the prompt enter this command (you can copy and paste it if you wish):

```
docker logs fabricvscodelocalfabric-peer0.org1.example.com-first-contract-0.0.1
```

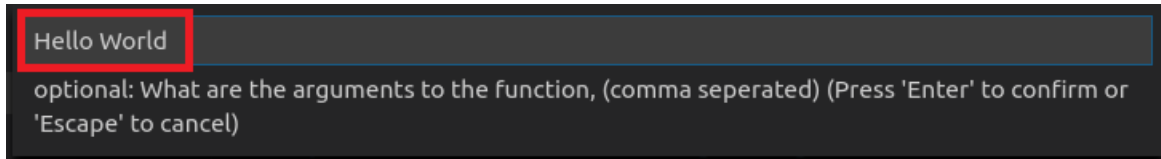
This will produce several lines of output, most of which you can ignore and at the bottom of which will be the word “**instantiate**”.



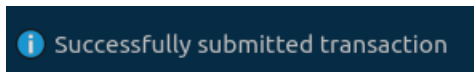
__ 32. Now we will execute another transaction. From the “**FABRIC GATEWAYS**” view, expand out the instantiated contract as you did before until you can see the transactions, right click on “**transaction1**” and choose “**submit transaction**”:



- __ 33. In the dialogue at the top of the screen enter the text “Hello World” or some similar text as shown below and press “Enter”. Note that you should not enter the quotes around the words as otherwise they will be taken as part of the string itself.



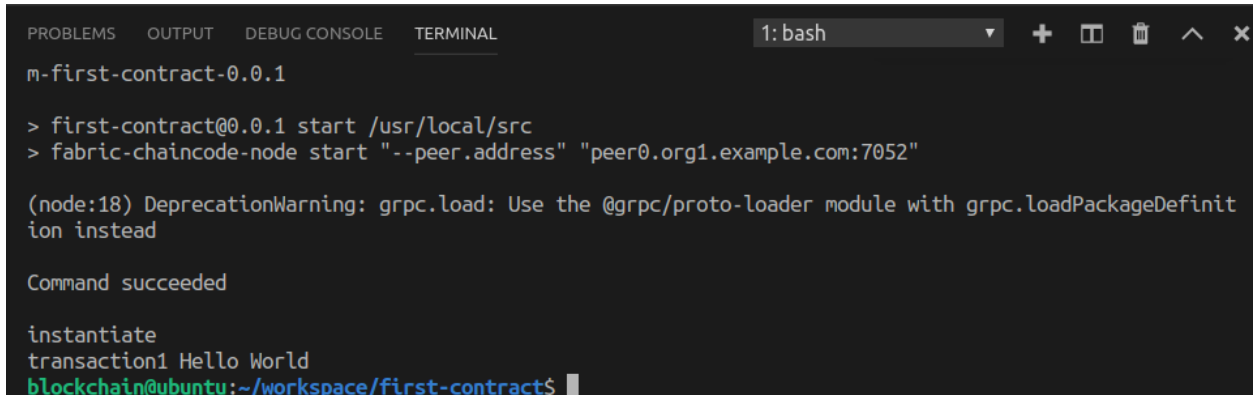
An information message will inform you when the transaction is complete:



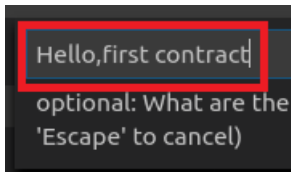
- __ 34. To take a look at the output, switch back to the terminal view and press the “up arrow” key to choose the docker logs command again. If you have trouble you can just re-enter it:

```
docker logs fabricvscodelocalfabric-peer0.org1.example.com-first-contract-0.0.1
```

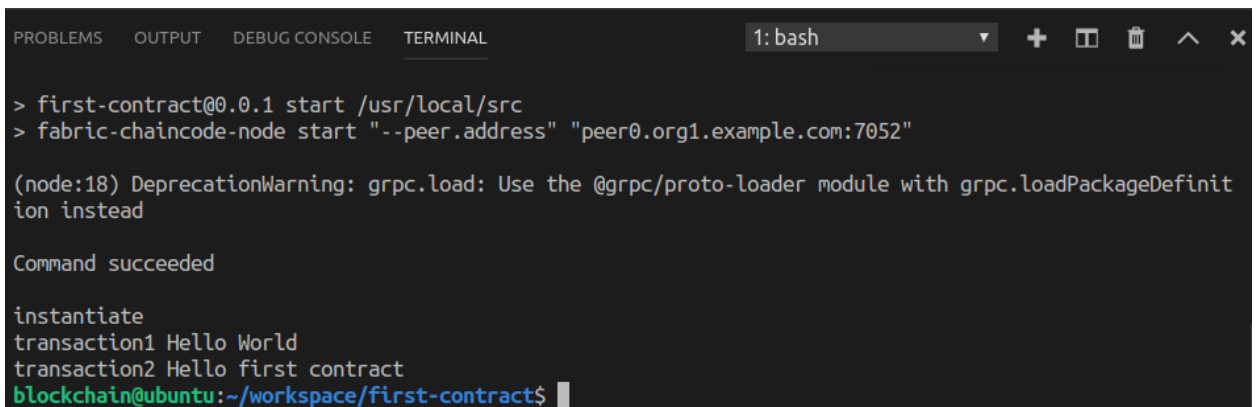
You should now see the output contains “transaction1” followed by the text you entered above.



- ___ 35. Finally let's call the second transaction for completeness. Right click on **transaction2** as you did above for the first transaction. This time the transaction takes two parameters, so we need to enter each one separated by a comma, such as **Hello,first-contract** . Note that there should be no quotes or spaces around the parameters as shown below as otherwise they are taken as part of the parameters itself - they are not stripped off before the transaction is called. Press **“Enter”** when you are done.

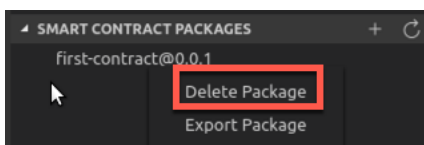


- ___ 36. Again, switch back to the terminal view as you did before and re-execute the docker logs command again to see the output from the second transaction call. You should see something like this below:



We have now almost completed this lab – an overview of the VSCode development experience. All that remains is to clear up the environment ready for the next lab.

- ___ 37. From the IBP **Smart Contract Packages** view, right-click on the **first-contract@0.0.1** package and choose the **Delete Package** option as shown below to remove it:

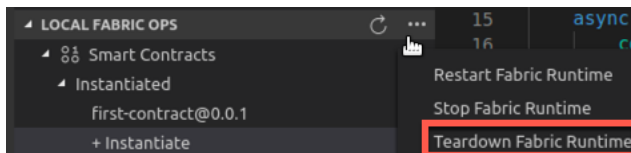


IBM Blockchain

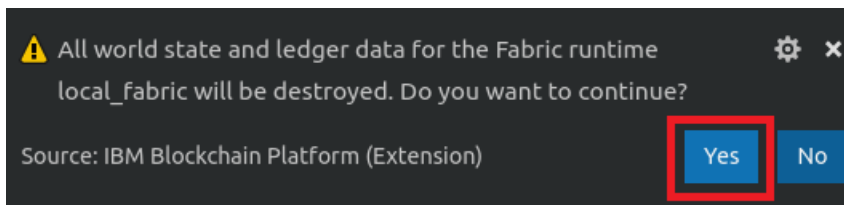
- ___ 38. From the **FABRIC GATEWAYS** view, select the **Disconnect from Gateway** icon as shown below:



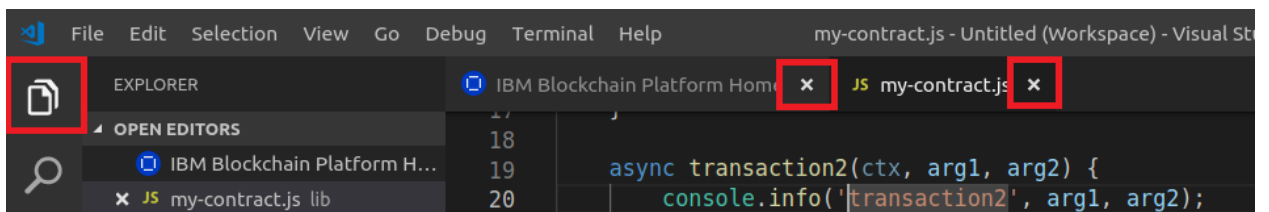
- ___ 39. From the **LOCAL FABRIC OPS** view, click on the ... and select the “**Teardown Fabric Runtime**” option from the context menu:



- ___ 40. From the dialogue that appears in the bottom right, choose the “Yes” button:

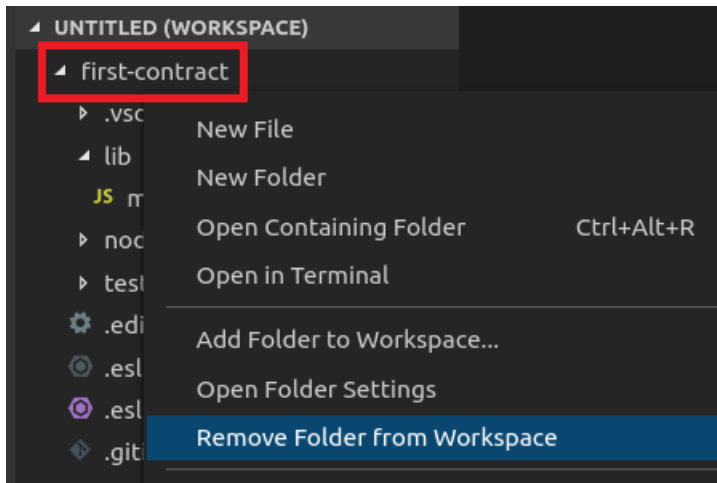


- ___ 41. Switch back to the explorer view and close all open editors, including the “IBM Blockchain Platform Home” by clicking on the “x” button on each tab



IBM Blockchain

- ___ 42. Right click on the “first-contract” folder and chose the “Remove Folder from Workspace” context menu option.



Note: if you cannot see the **Explorer** view, click on the **Explorer** icon again to make it re-appear.

- ___ 43. This will re-open the “IBM Blockchain Platform Home” page and leave your workspace ready for the next lab as shown below:

