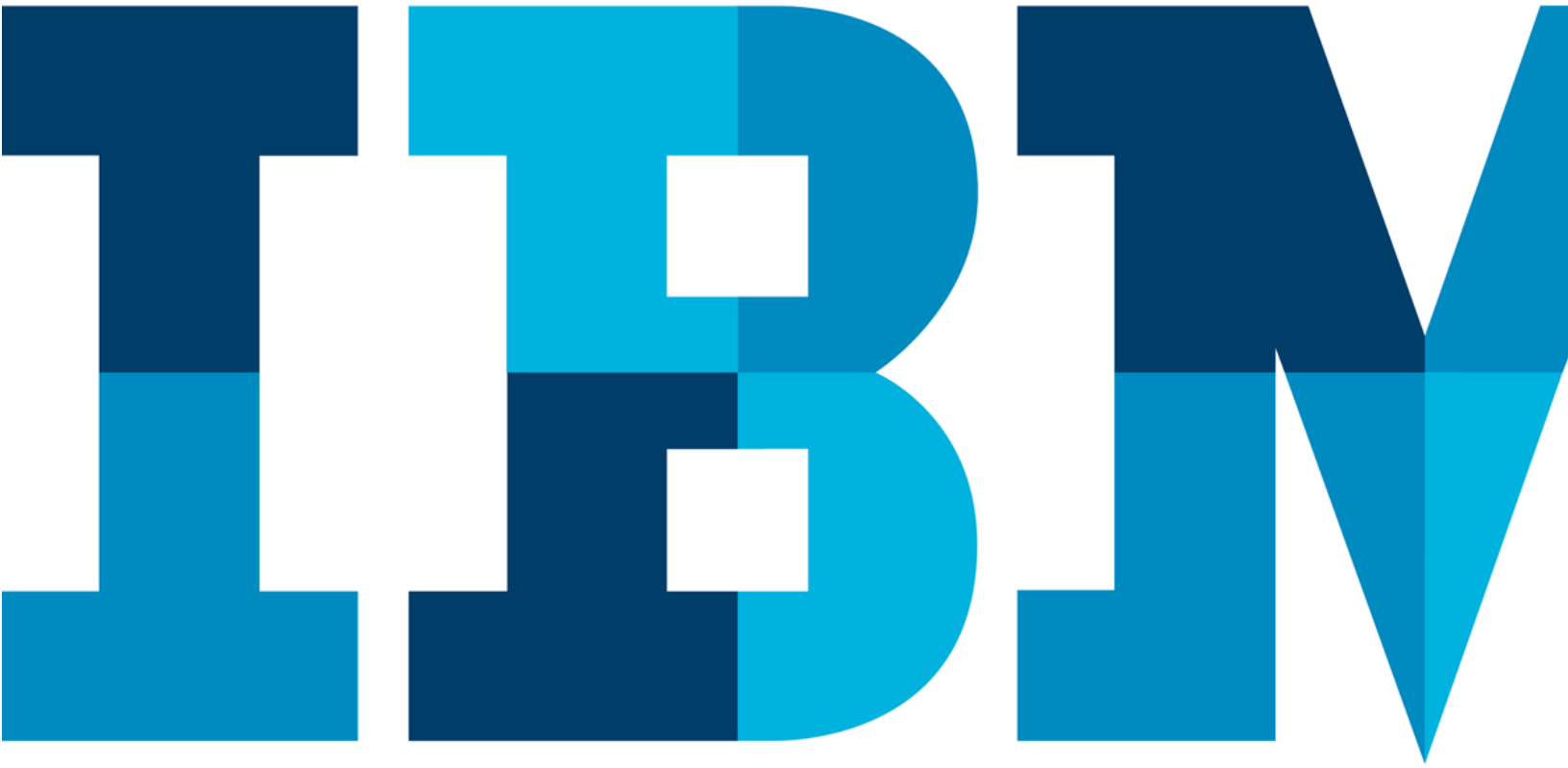
IBM Blockchain Platform Hands-On

IBM MQ Bridge for Blockchain Lab



IBM

Table of Contents

Disclaimer

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Overview of the lab environment and scenario

***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. Please note that any commands you are asked to execute in a terminal, can be copied and pasted from the lab guide. Also, from section 4 onwards, you will repeat some commands to launch a Dealer or Regulator applications; you can simply hit the **UP** or **DOWN** arrow, to scroll back/forward to a previous command.*

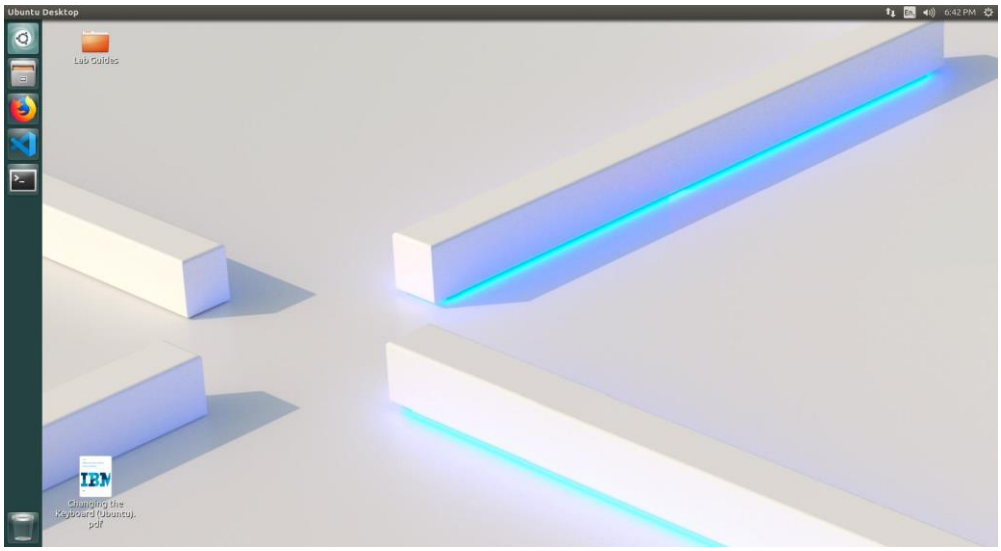
Steps:

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 user id and password are both "**blockchain**".

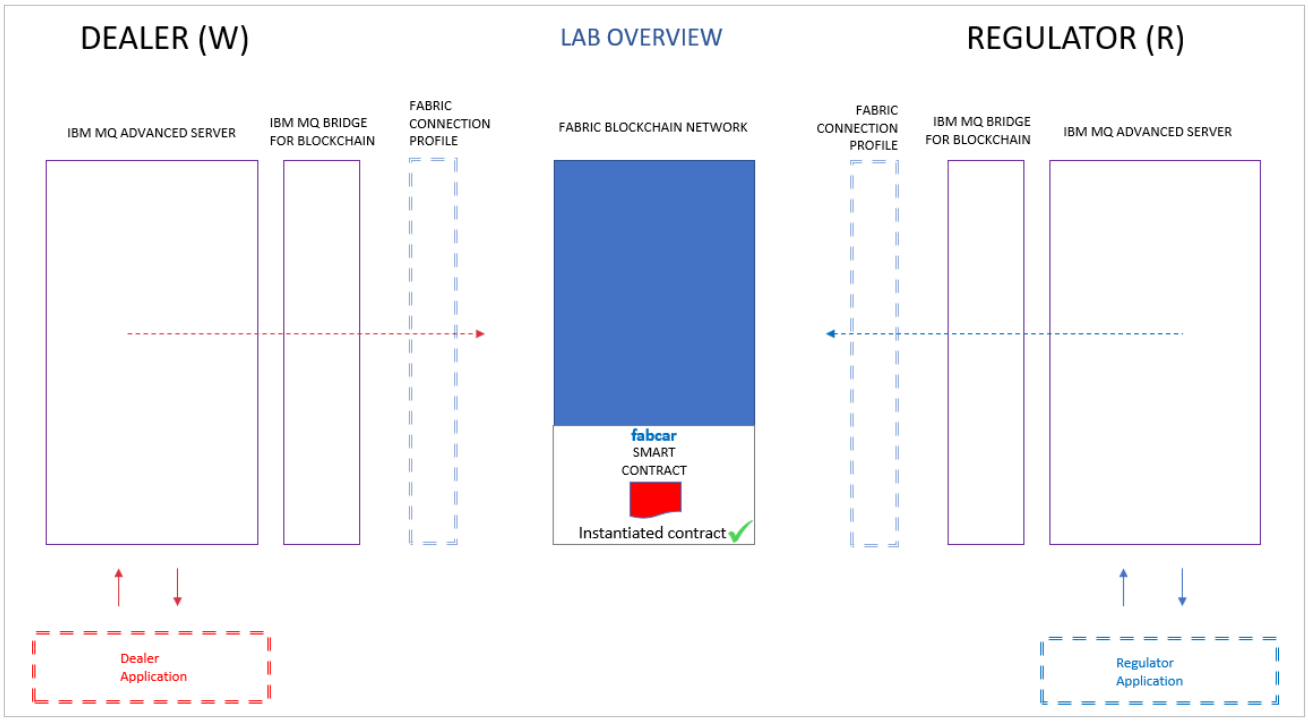


1.1 Introduction

This lab shows how to integrate IBM MQ, a messaging solution for applications, with a Hyperledger Fabric blockchain network. Two organisations participate in a vehicle lifecycle business network; each uses an application to record or query changes to vehicle records on the shared ledger. Applications post MQ messages on application queues – as such, they do not need to understand 'where' the blockchain is. The runtime for integrating IBM MQ queues with the Fabric network is provided by the IBM MQ Bridge for Blockchain. An instance of the bridge runs in each organisation; it processes application messages on input/request queues in IBM MQ, issues smart contract transactions to the blockchain and manages the responses/results returned, passing them back to the applications to consume.

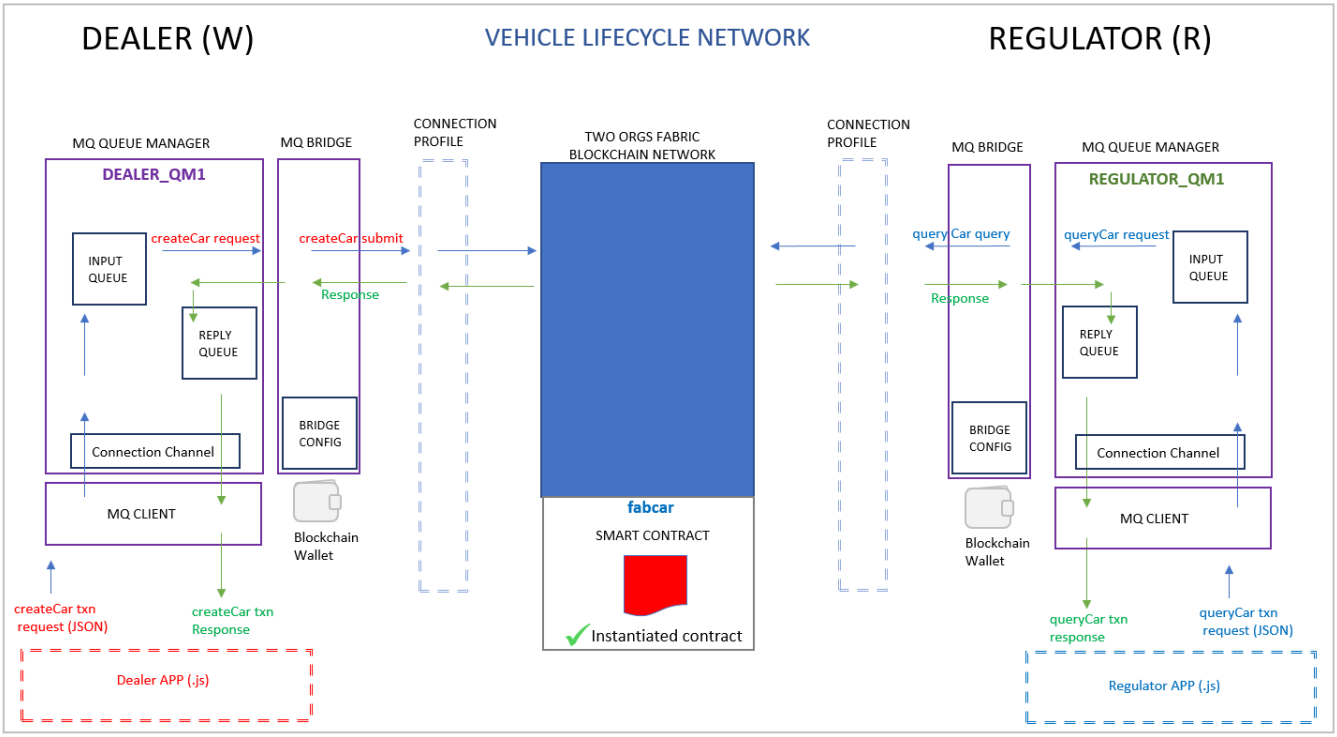
1.2 Lab Overview

The lab outlines how to integrate applications, that are MQ enabled to be able to interact with a blockchain ledger. In the lab, you complete some basic configuration steps in IBM MQ and for the IBM MQ Bridge for Blockchain, to integrate with the running Fabric network. You then run applications that test the end-to-end integration. The lab emphasises the application perspective; i.e. showing how the IBM MQ Bridge for Blockchain component abstracts away technical complexity. For applications, it's 'business as usual' – see diagram for an overview.



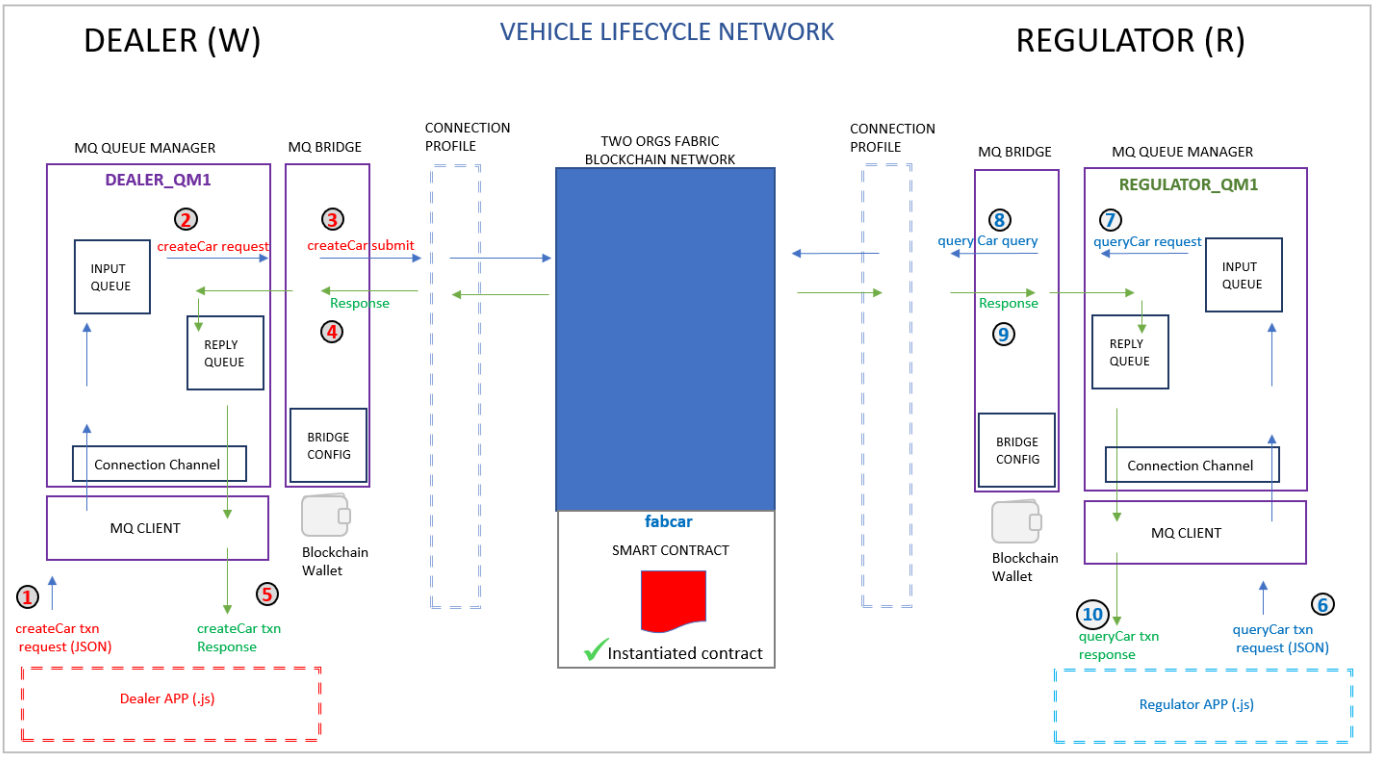
1.3 **Scenario**

The scenario follows two application users **Dino** and **Ron**, from a Dealer and a Regulator organisation in a business network. Each uses their respective applications to create or query car records – e.g. the dealer creates a car record, the regulator queries car records. Whilst creating or querying records on the blockchain, you also will examine the structure of MQ messages that go to/from the blockchain, such as a create car request or a reply message from a smart contract transaction. You will also configure a bridge, to connect MQ to the blockchain.



1.4 Lab Flow

The diagram shows the Dealer and the Regulator interacting with the blockchain network. The message flow on the left (numbered 1 through 5) shows the **Dealer message flow** for the `createCar` sequence. On the right , steps 6 through 10, show the **Regulator message flow**, for the `queryCar` sequence. The bridge component picks up requests from a request queue, and posts responses on a reply queue, which their respective applications consume.



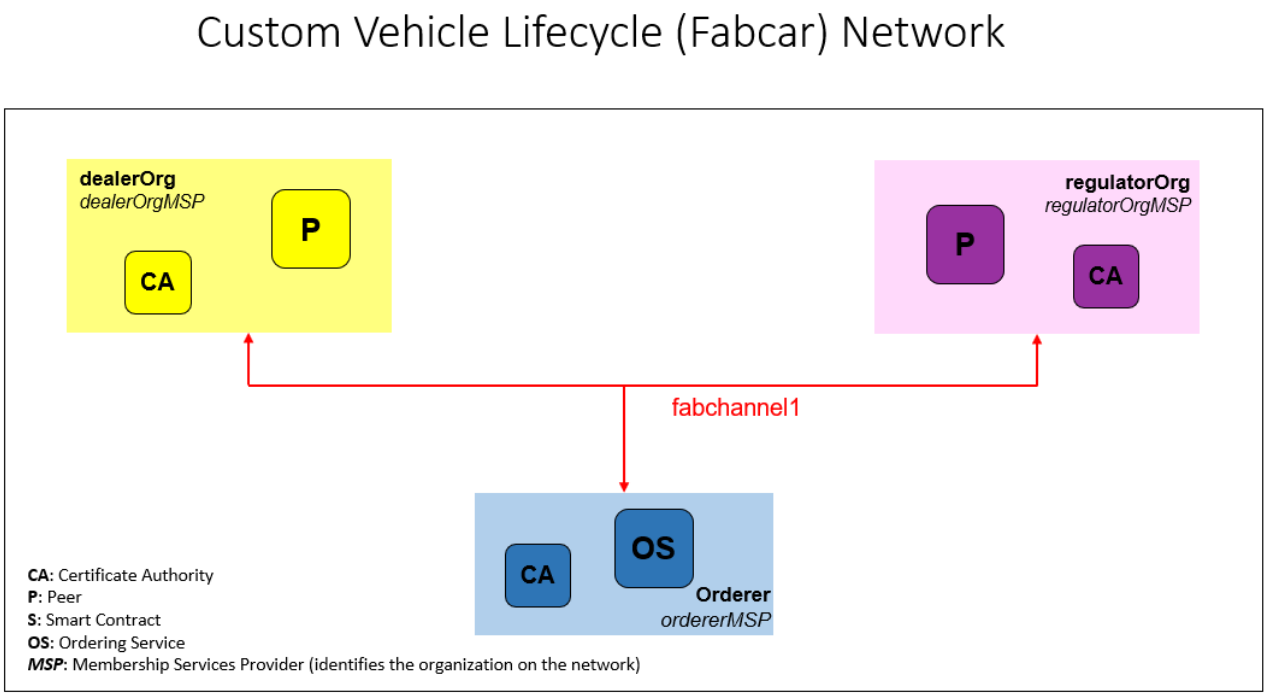
1.5 Vehicle Lifecycle blockchain network – how it was built

The lab uses a custom Hyperledger Fabric network 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 VM comes with a running, two organization Fabric network – the ansible script that built this is called `site.yml` under the `mqbridge/hlf-ansible` folder. No knowledge of Ansible is required for this lab.

The network uses one Fabric channel, **fabchannel1.** Each Peer uses CouchDB as the State Database. The IBM Blockchain Platform for VS Code extension has a built-in feature to import the Fabric environment (i.e. import all Fabric nodes, gateways and wallets/identities generated by Ansible).

The ansible script also instantiates a Fabcar smart contract (fabcar@1.0.1) on channel fabchannel1.

Footnote: for this lab, custom organisation names (Dealer, Regulator) and identities are used, to make lab scenarios easier to read and execute - hence why the custom network was built. You could adapt this lab for your own purposes, to use the standard, built-in 'Two Org' network, under the 'Fabric Environments' view in the IBM Blockchain Platform VS Code extension – but would use standard names like Org1 and Org2 and use administrative identities like 'admin' for performing the equivalent tasks in this lab. See diagram for overview.



1.6 Lab Structure

This lab steps are structured into an overview section and 7 distinct lab parts:

The overview describes the lab aims and Lab environment, including the scenario and how the lab flows. It also describes how the custom blockchain network was built.

Part 1 of the lab takes you through **Importing the Vehicle Lifecycle Fabric network** An ansible script was executed prior to this lab to bring up the two-organisation Fabric network; you will complete the setup by importing the Fabric environment, using a new feature in the IBM Blockchain Platform VS Code extension.

Part 2 of this lab will **configure Dealer and Regulator MQ environments**, to create Queue Managers (one for each Org) and Server Connection channels, so that application clients can connect to MQ queues to send requests to the blockchain. You will also create sample Queue

definitions in each Queue Manager. All MQ configuration steps are completed using a combination of bash and MQ batch scripts.

Part 3 of this lab will focus on **configuring the Dealer and Regulator IBM MQ Bridge for Blockchain** components, namely the requisite information required in the bridge, for IBM MQ Advanced server to be able to interact with the blockchain network. You will carry out the manual configuration steps for the Dealer bridge component and Dealer MQ environment.

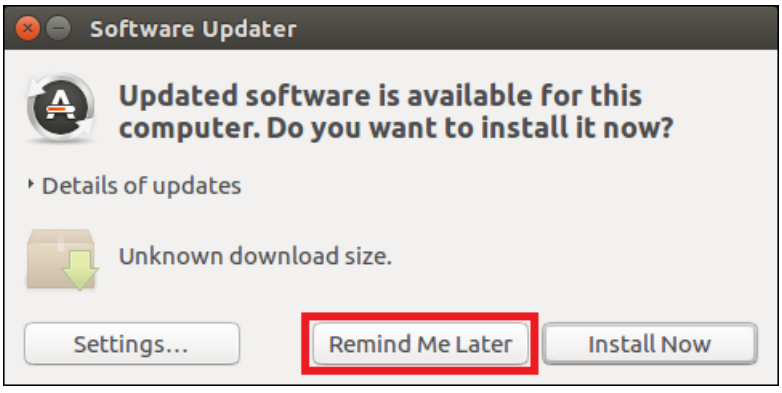
Part 4 of this lab provides insight into **reviewing and running the Dealer App environment**; you will review key information about the Dealer Node.JS application. You will perform a test message from the App, so that they can be examined in MQ Explorer – then start the bridge for the Dealer organisation to process it. Next, you will complete an end-to-end transaction (createCar) showing the Dealer App consuming the response for that transaction, from the blockchain.

Part 5 of this lab provides an insight into **reviewing and running the Regulator App environment;** similar to Part 4 – this time, you start the bridge component for the Regulator organisation, then launch the Regulator App, and query the details of car(s) created by the Dealer.

Part 6 of this lab shows an end-to-end **Change Car Ownership** transaction where the Dealer changes the owner of a car. You will then carry out a verify pattern using the Regulator App; user Ron (Regulator) queries the car details, to show the current ownership on the ledger.

Part 7 of this lab deals with **Audit history of Previous Ownership**, as the Regulator. The Dealer user performs another series of changes to create a history of owners. The Regulator queries the history of previous ownership for the car in question, to reveal that history from the ledger.

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



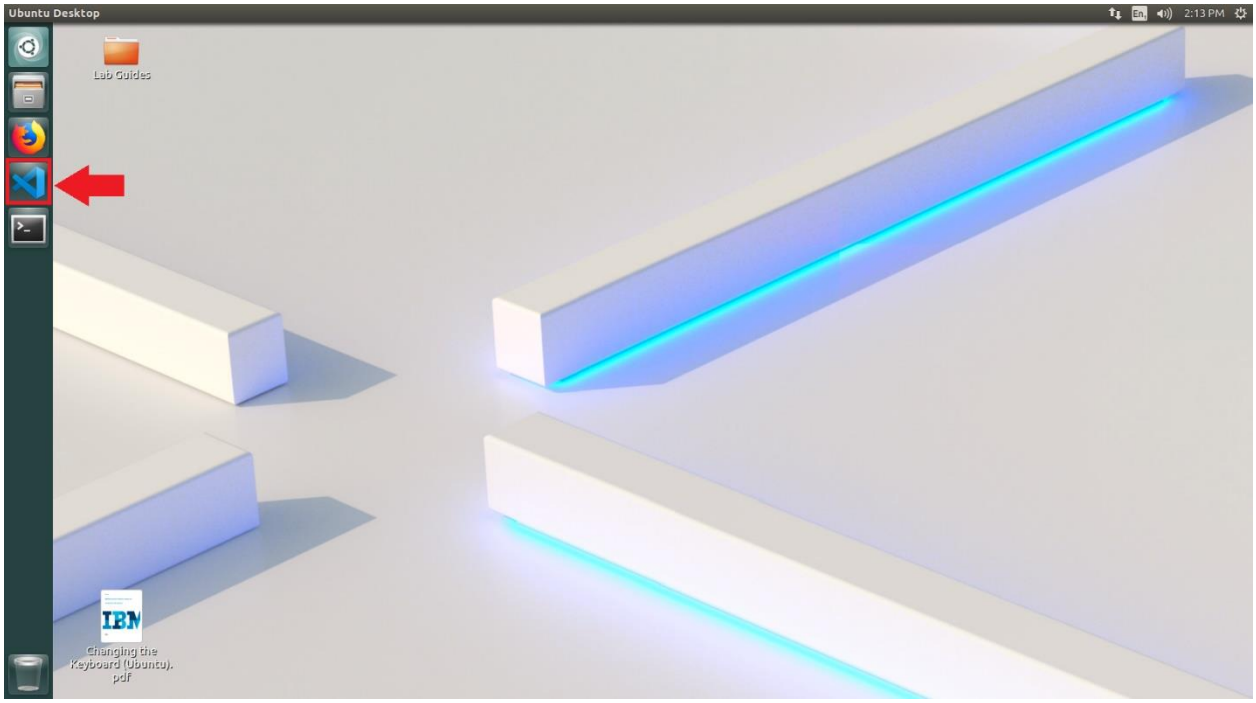
1 Import ‘Vehicle Lifecycle Network’ Fabric Environment

1.1 Introduction

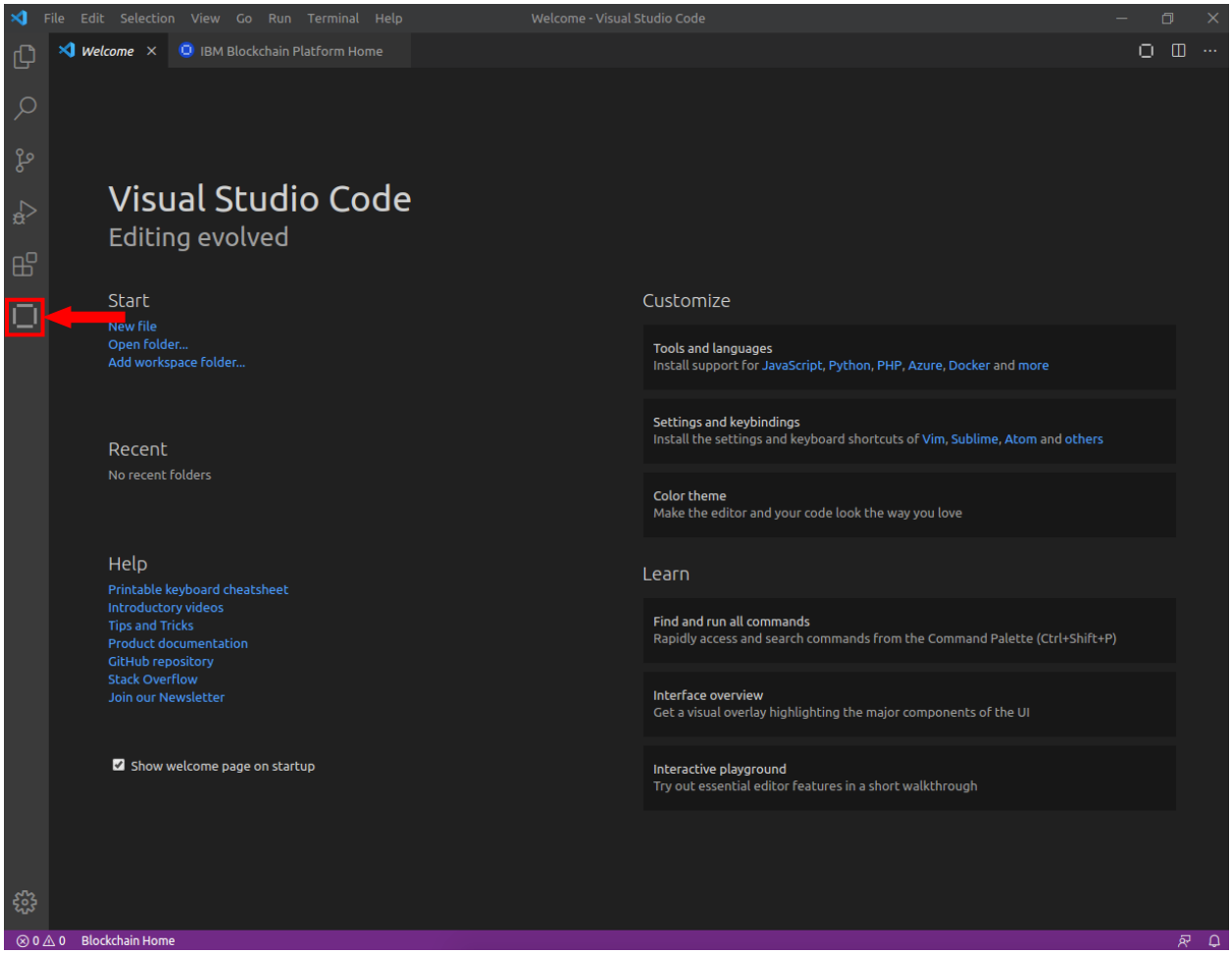
This section of the lab covers the import of the custom Vehicle Lifecycle network. The Virtual Machine (VM) you are using, has already executed an Ansible playbook to create the Fabric network, comprising Dealer and Regulator network members. You now need to import this Fabric environment in the IBM Blockchain VS Code extension, as a single step. After this step, you can interact with the imported Nodes, Wallets and Gateways that are part of the blockchain network.

Steps:

__ **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.



Let's browse the Ansible "playbook" file (site.yml) that deployed the Fabric network; you do this from a terminal window in the `mqbridge/hlf-ansible` subdirectory.

__ **5.** Open a terminal window from the Ubuntu task bar:

__ **6.** Copy and paste the following commands in the terminal window:

```
cd ~/workspace/mqbridge/hlf-ansible
more site.yml
```

```
blockchain@ubuntu:~$ cd workspace/mqbridge/hlf-ansible
blockchain@ubuntu:~/workspace/mqbridge/hlf-ansible$ more site.yml
---
- name: Deploy MQ demo blockchain infrastructure and smart contracts
  hosts: localhost
  vars:
    #
    # For information on these configuration options, read the documentation:
    # https://github.com/IBM-Blockchain/ansible-role-blockchain-platform-manager#example-playbook
```

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

__ **7.** Although the Fabric network is built, the containers need to be started – execute the **start.sh** bash script as follows (with leading “.” below) to start up the containers.

```
./start.sh
```

```
blockchain@ubuntu:~/workspace/mqbridge/hlf-ansible$
blockchain@ubuntu:~/workspace/mqbridge/hlf-ansible$ ./start.sh
Starting the containers
ffb2f5f86a68
80bd81b67b0b
cc456b64d5ac
41f3cb6a9bea
47e88a0127de
3c1e7fe79584
3e5e3977ed9b
00ba5e1bf8ab
b177396ec696
done....
blockchain@ubuntu:~/workspace/mqbridge/hlf-ansible$
```

__ **8.** Next, verify that the **8** expected containers are running for the Vehicle Lifecycle Fabric network, using the following docker command:

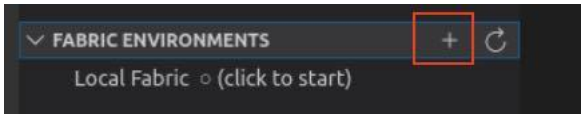
```
docker ps --format 'table {{.Names}}:\t {{.Ports}}'
```

```
blockchain@ubuntu:~/workspace/mqbridge/hlf-ansible$ docker ps --format 'table {{.Names}}:\t {{.Ports}}'
NAMES:                          PORTS
orderer.example.com:            0.0.0.0:17050->17050/tcp, 7050/tcp, 0.0.0.0:17055->17055/tcp
ca.orderer.example.com:         7054/tcp, 0.0.0.0:16054->16054/tcp
peer0.regulator.example.com:    0.0.0.0:18051-18053->18051-18053/tcp
couchdb0.regulator.example.com: 4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp
ca.regulator.example.com:       7054/tcp, 0.0.0.0:18054->18054/tcp
peer0.dealer.example.com:       0.0.0.0:17051-17053->17051-17053/tcp
couchdb0.dealer.example.com:    4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp
ca.dealer.example.com:          7054/tcp, 0.0.0.0:17054->17054/tcp
```

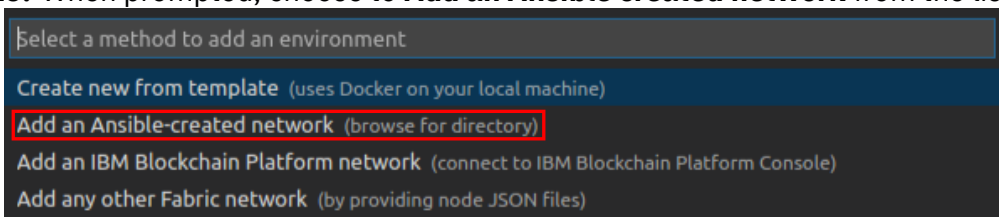
The output (example above) should confirm the network is running.

Now import this Fabric environment (i.e. the Fabric nodes/wallets/gateways) using the import Fabric environment feature in the IBM Blockchain Platform VS Code extension.

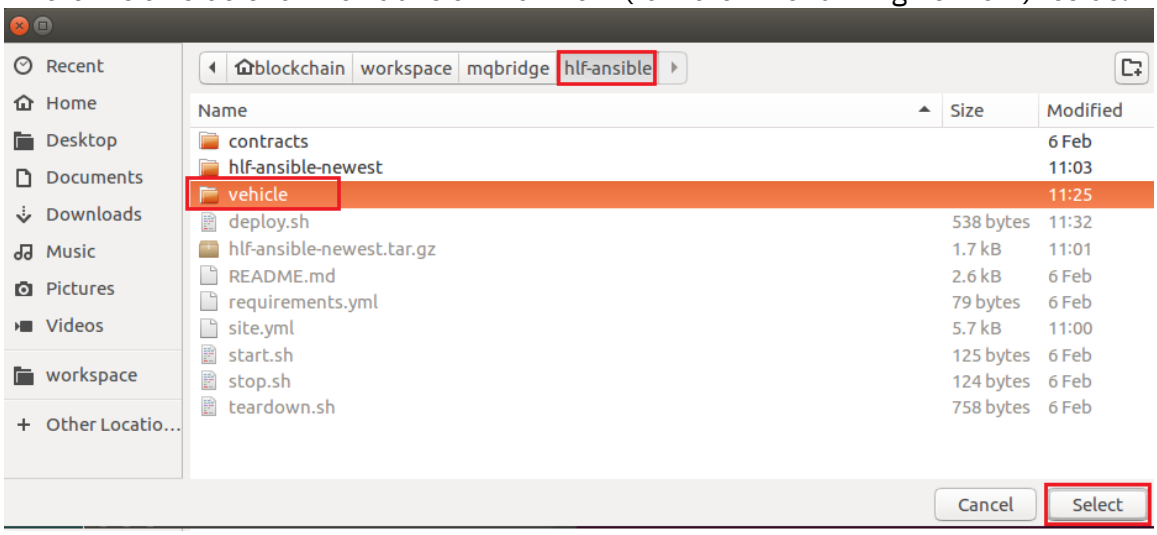
__ **9.** Back in the VS Code extension, click the '**+**' sign in Fabric Environments to add a new environment



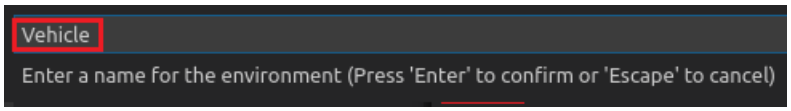
__ **10.** When prompted, choose to **Add an Ansible created network** from the list



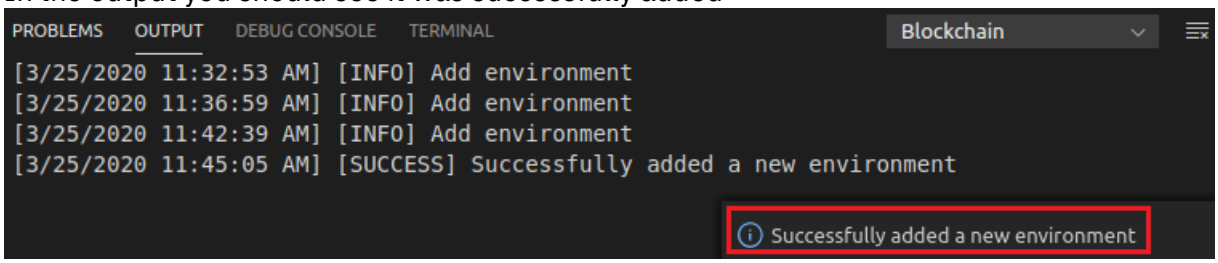
__ **11.** Browse to the directory “Home” > blockchain > workspace > mqbridge > hlf-ansible and then highlight the **vehicle** folder and click **Select** on the bottom right. This folder is where the artefacts for the Fabric environment (to match the running network) reside.



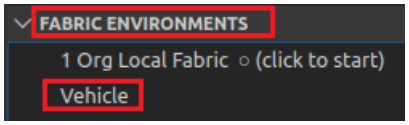
__ **12.** Specify the name **Vehicle** (upper-case ‘V’) for your two organization Vehicle Lifecycle network and hit **enter**.



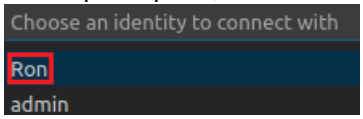
In the output you should see it was successfully added



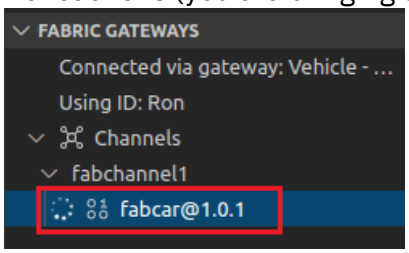
__ **13.** Under **Fabric Environments**, click on **Vehicle** to verify you can successfully connect to the new environment – confirmation of this is shown in the Output panel.



__ **14.** Next, under **Fabric Gateways**, click on the **Vehicle - regulatorOrg_gw** gateway and when prompted, connect as the identity **Ron** from the list of identities offered.

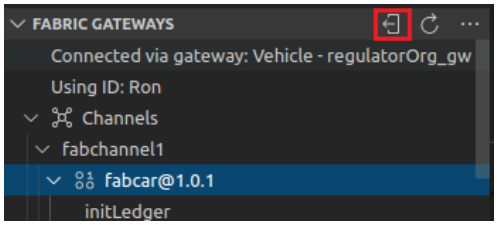


__ **15.** Expand the channel **fabchannel1....** then expand the contract twisty for **fabcar@1.0.1** – it will take a number of seconds (spinning icon) to reveal the list of transactions (you are bringing up the Regulator organization's chaincode container).

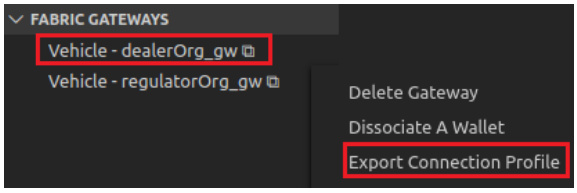


The two-organization network, including all identities, gateways and nodes under the 'vehicle' subdirectory in the `hlf-ansible` folder, is ready for use in VS Code. You will now need to export Connection Profiles, for use later by the IBM MQ Bridge for Blockchain component (one exported profile for each org).

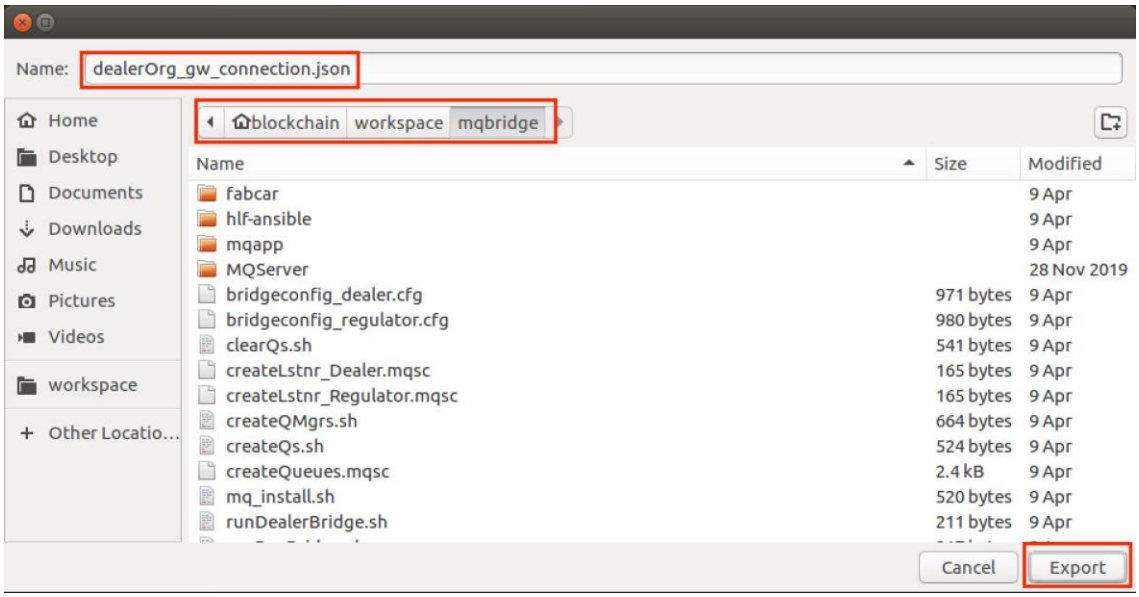
__ **16.** On the **Fabric Gateways** panel click **Disconnect** from the Regulator Gateway using the disconnect icon



__ **17. Hover** over on the gateway **Vehicle - dealerOrg_gw** and right-click ... **Export Connection Profile**



__ **18. Export** the file with the name **dealerOrg_gw_connection.json** (remove the leading '*Vehicle* – ' prefix, incl. the '–') and export it to the **workspace/mqbridge** folder:



__ **19.** Using steps *17 – 18* as a guide, hover over the **Vehicle – regulatorOrg_gw** gateway and right-click … **Export Connection Profile** to the same folder as above – **export** it as the filename **regulatorOrg_gw_connection.json** .

In the next section, you will focus on configuring IBM MQ for both the Dealer and Regulator organizations.

Review

In this section you have:

- Examined an Ansible playbook (*site.yml*) that built the vehicle lifecycle blockchain network used in this lab.
- Executed a single step to import the running Fabric environment, including Nodes, Wallets and Gateways then connected to the environment.
- Exported connection profiles for both the Dealer and Regulator environments, for inclusion in the IBM MQ Bridger for Blockchain configuration steps later.

2 Configure Dealer and Regulator MQ Environments

2.1 Introduction

This section executes the IBM MQ ("MQ") configuration needed for this lab. MQ application queues (defined in Queue Managers) and MQ channels are required for both the client applications and the MQ Bridge instances – each will post or process messages.

The steps to create these MQ artefacts are **automated** using bash scripts and 'MQSC' batch commands (more info in Appendix 4) – these are run for the Dealer and Regulator MQ environments. Once done, the Node.JS applications can interact with MQ, as well as the IBM MQ Bridge for Blockchain – you will configure this later.

An INPUT Queue (to receive application messages) and a REPLY queue (blockchain results) are created for each organisation. You will review queues using MQ Explorer (MQ Explorer comes with IBM MQ – it is a UI, that is very useful for administering MQ, performing tasks/operations and checking the status of MQ objects).

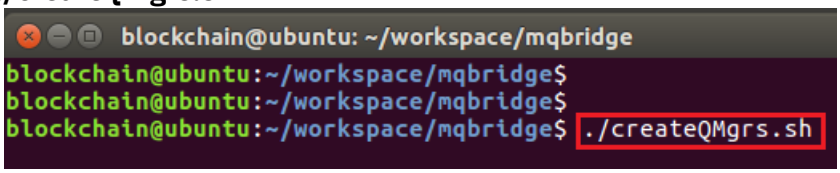
Steps:

__ **20.** Using the **Ubuntu Explorer,** enter the letters **MQ** to find the MQ Explorer application:

__ **21.** **Drag the MQ icon** onto the task bar on the left - press **Escape to** leave the Ubuntu Explorer

__ **22.** Return to the **terminal window** and change directory to HOME > blockchain > workspace > mqbridge subdirectory, run the following command to **create the Queue Managers and Channels** for both the Dealer and the Regulator.

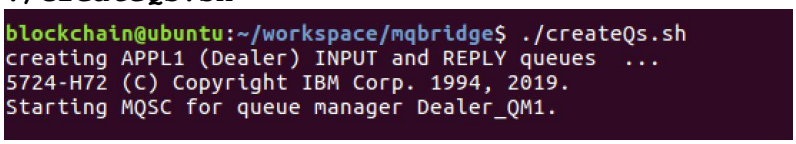
cd ..
./createQMgrs.sh



You should get confirmation that the sequence of steps was successful:

__ **23.** From the same subdirectory **`/home/blockchain/workspace/mqbridge`**, run the `createQs.sh` script to create the Application Queues:

`./createQs.sh`



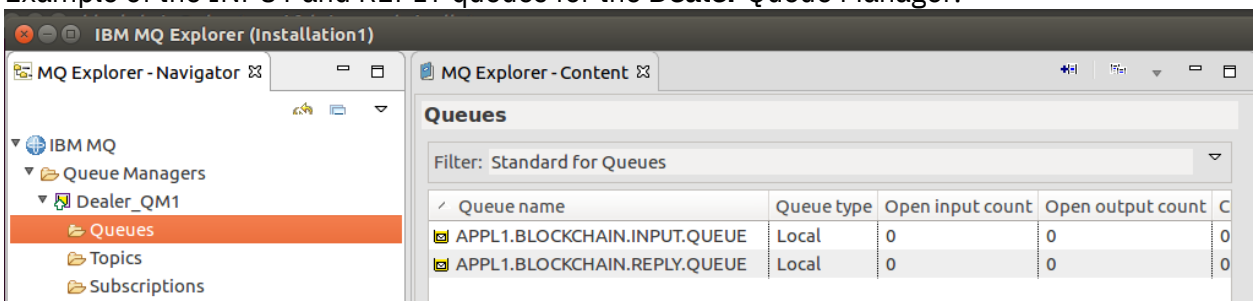
See output next page

From the output, you should see messages that two Queues were created (as well as certain security settings being disabled for this lab).

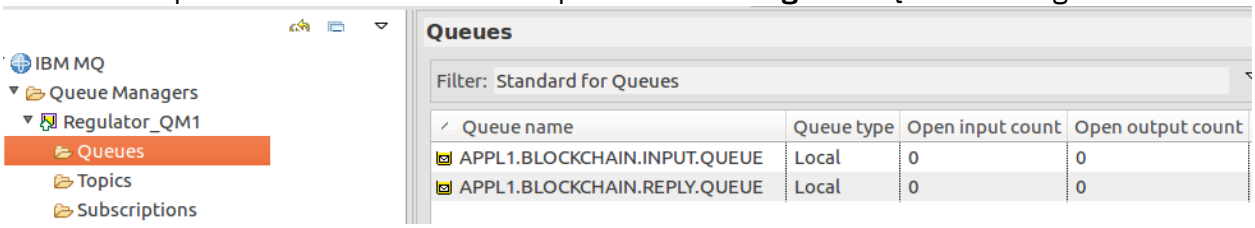
```
AMQ8006I: IBM MQ queue created.
        :
     5 : DEFINE QLOCAL(APPL1.BLOCKCHAIN.INPUT.QUEUE) +
       :   LIKE(SYSTEM.BLOCKCHAIN.INPUT.QUEUE) +
       :   DESCR('Sample Blockchain application input queue')
AMQ8006I: IBM MQ queue created.
        :
     6 : DEFINE QLOCAL(APPL1.BLOCKCHAIN.REPLY.QUEUE) +
       :   DESCR('Sample Blockchain application reply queue') +
       :   DEFPSIST(YES) +
       :   DEFSOPT(SHARED) +
       :   SHARE       +
       :   REPLACE
AMQ8006I: IBM MQ queue created.
        :
6 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Starting MQSC for queue manager Regulator_QM1.


     1 : DEFINE LISTENER(TCP.LISTENER) TRPTYPE(tcp) CONTROL(qmgr) PORT(1415)
AMQ8626I: IBM MQ listener created.
     2 : START LISTENER(TCP.LISTENER)
AMQ8021I: Request to start IBM MQ listener accepted.
     3 : DEFINE CHANNEL('APPL.CLIENT.SVRCONN') CHLTYPE(SVRCONN) TRPTYPE(TCP)
AMQ8014I: IBM MQ channel created.
3 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed
blockchain@ubuntu:~/workspace/mqbridge$
```

__ **24.** Start **MQ Explorer** by clicking on the icon you dragged to the left task bar, and verify that you can see two queues (INPUT and REPLY) for each of **Dealer_QM1** and **Regulator_QM1** queue managers by navigating to the **Queue Managers**....**Queues** folder.

Example of the INPUT and REPLY queues for the **Dealer** Queue Manager:



And an example of the INPUT and REPLY queues for the **Regulator** Queue Manager



OK. You have now completed the MQ Configuration setup steps in the lab.

Review

In this part of the Lab you have:

- Created application Queue Managers for Dealer and Regulator in IBM MQ.
- Created application input and reply queues in each Queue Manager and applied channel and security settings on the Queue Manager via MQ command files.

3 Configure Dealer/Regulator MQ Bridge for Blockchain

3.1 Introduction

In this section, you configure the IBM MQ Bridge for Blockchain – an instance of the bridge runs for each of the organisations. Once configured, they will be launched and the bridge monitors designated Queues; INPUT application queues for blockchain transactions, and REPLY queues for the bridge to post responses / blockchain results - the application consumes the results. Applications do not need to know anything about the blockchain network setup – IBM MQ Bridge handles this. The applications are aware of IBM MQ only.

When you configure the MQ Bridge for Blockchain component for the first time, using a CLI based configuration program called **runmqbcb** - it simply asks a series of questions and creates a configuration file based on your answers. More information in the IBM MQ Bridge for Blockchain configuration can be found here https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q130890_.htm and in Appendix 2 of this guide.

You will go through the steps to create the Bridge; it will need to know information about the blockchain network it is connecting with, the identity it will use, and MQ configuration details (like queue manager, queue names).

Steps:

__ **25.** In a terminal window ensure you are in the directory **/home/blockchain/workspace/mqbridge/**

```
cd /home/blockchain/workspace/mqbridge/
```

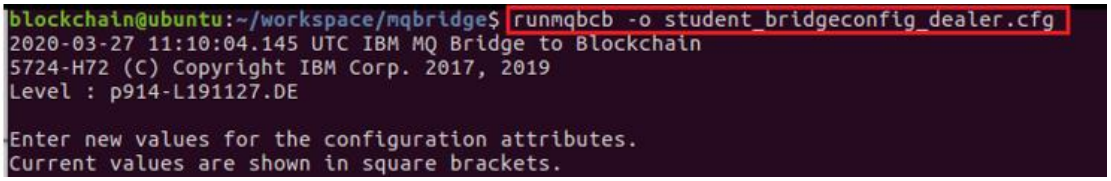
__ **26.** Set the MQ environment variable (note the leading '.' below sets in the current shell)

```
. /opt/mqm/bin/setmqenv -s -k
```

There is no output from this command – it simply sets some environment variables.

__ **27.** Launch the IBM MQ Bridge for Blockchain configuration tool as follows:

```
runmqbcb -o student_bridgeconfig_dealer.cfg
```



You are asked to provide answers to questions interactively – use the answers in **BOLD** in **column 2** below. In some cases, hit **enter** to accept the [default], or leave it blank: (For some of the longer Answers, you might like to cut and paste the values from the table.)

Parameter	Answer
--------	**------ Connection to Queue Manager questions ------**
Queue Manager	**Dealer_QM1**
Input Queue	**APPL1.BLOCKCHAIN.INPUT.QUEUE**
MQ Channel	**APPL.CLIENT.SVRCONN**
MQ Conname	**127.0.1.1(1414)**
MQ CCDT URL	<leave blank>
JNDI implementation class	Hit **enter** (accept the default [com.sun.jndi.fscontext.RefFSContextFactory])
JNDI provider URL	Hit **enter** <leave blank>
MQ UserId	Hit **enter** <leave blank>
MQ Password	Hit **enter** <leave blank>
--------	**------ Fabric Configuration ------**
Network Configuration file	**/home/blockchain/workspace/mqbridge/dealerOrg_gw_connection.json**
Wallet	**/home/blockchain/workspace/mqbridge/hlf-ansible/vehicle/wallets/dealerOrg**
User Name	**Dino**
Certificate	Hit **enter** <leave blank>
Private Key	Hit **enter** <leave blank>
Organisation	**dealerOrgMSP**
Commit Timeout	Hit **enter** (accept default of [15])
Network Discovery	Hit **enter** (accept default of [N])
Updates wait for all Peers?	Hit **enter** (accept default of [Y])
Updates sent to all organisations?	Hit **enter** (accept default of [N])
--------	**------ Certificate Stores for MQ TLS connections ------ (no TLS used)**
Personal keystore	Hit **enter** <leave blank>
Keystore Password	Hit **enter** <leave blank>
Trusted store for signer certs	Hit **enter** <leave blank>
Trusted store password	Hit **enter** <leave blank>
--------	**------ Behaviour of bridge program ------**
Runtime logfile for stdout/stderr	**/home/blockchain/workspace/mqbridge/dealer-bridge.log**

Number of logfiles	Hit **enter** (accept default of [3])
Maximum size of each logfile	Hit **enter** (accept default of [2097152])

The completed output of your configuration will look something like the following:

```
Enter new values for the configuration attributes.
Current values are shown in square brackets.

Press ENTER to accept current values; use SPACE+ENTER
to clear values; use <new value>ENTER to set a new value.

If lists of values are required these may be separated by commas
or entered on multiple lines. A blank line terminates the list.

NOTE: You cannot edit existing values - you can only keep, replace or clear them.

Connection to Queue Manager
---------------------------
Queue Manager                       : []Dealer_QM1
Bridge Input Queue                  : [SYSTEM.BLOCKCHAIN.INPUT.QUEUE]APPL1.BLOCKCHAIN.INPUT.QUEUE
MQ Channel                          : []APPL.CLIENT.SVRCONN
MQ Conname                          : []127.0.1.1(1414)
MQ CCDT URL                         : []
JNDI implementation class           : [com.sun.jndi.fscontext.RefFSContextFactory]
JNDI provider URL                   : []
MQ Userid                           : []
MQ Password                         : []

Fabric Server
-------------
Network configuration file          : []/home/blockchain/workspace/mqbridge/dealerOrg_gw_connection.json
Wallet                              : []/home/blockchain/workspace/mqbridge/hlf-ansible/vehicle/wallets/dealerOrg
User Name                           : []Dino
Certificate                         : []
Private Key                         : []
Organisation                        : []dealerOrgMSP
Commit Timeout(seconds)             : [15]
Network Discovery                   : [N]
Updates wait for all peers to respond? (Y/N) : [Y]
Updates sent to all organisations in network? (Y/N)  : [N]

Certificate stores for MQ TLS connections
-----------------------------------------
Personal keystore                   : []
Keystore password                   : []
Trusted store for signer certs      : []
Trusted store password              : []

Behaviour of bridge program
---------------------------
Runtime logfile for copy of stdout/stderr : []/home/blockchain/workspace/mqbridge/dealer-bridge.log
Number of logfiles                  : [3]
Maximum size of each logfile (bytes) : [2097152]
```

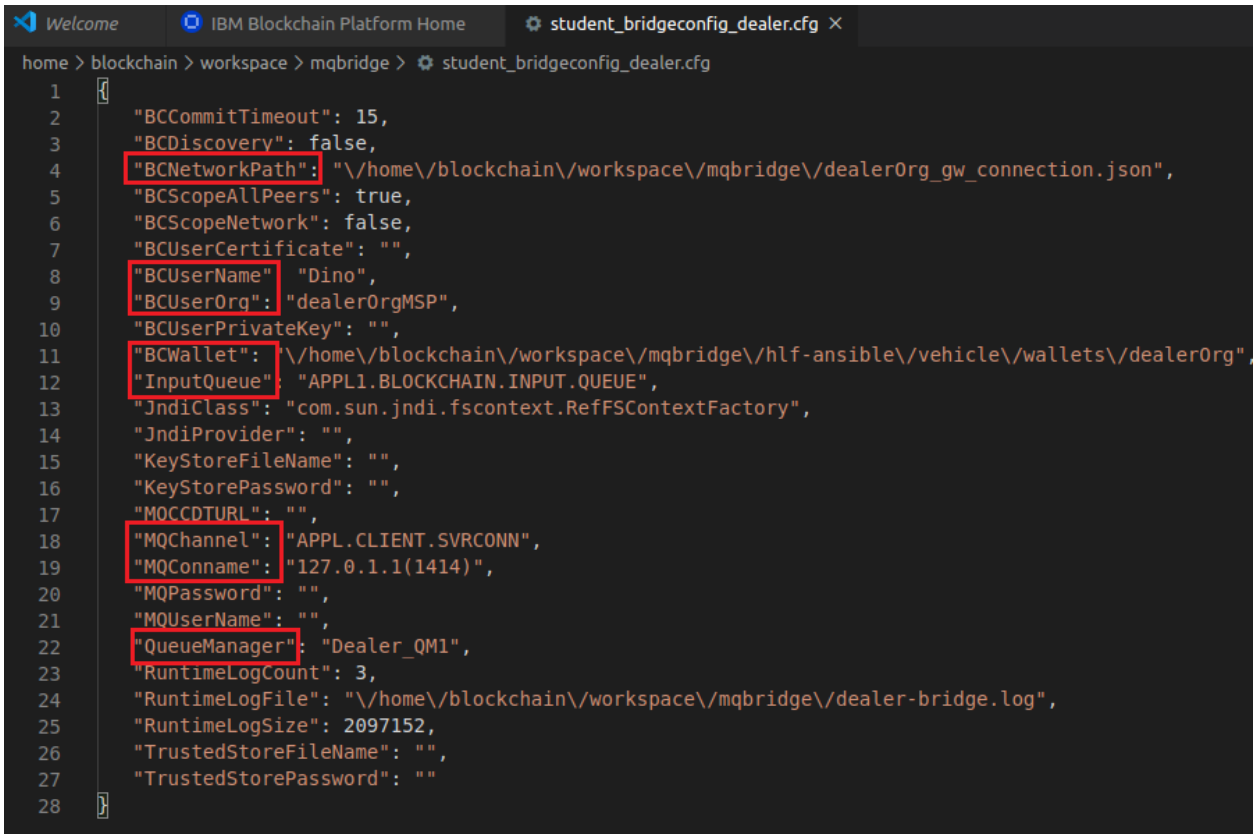
Upon completion, a “Done” message confirms the configuration is written to file.

```
Behaviour of bridge program
---------------------------
Runtime logfile for copy of stdout/stderr : []/home/blockchain/workspace/mqbridge/dealer-bridge.log
Number of logfiles                  : [3]
Maximum size of each logfile (bytes) : [2097152]
2020-03-27 11:55:09.374 UTC Done.
```

If any of the values are incorrect, simply delete **student_bridgeconfig_dealer.cfg** and rerun the **runmqbcb** command.

__ **28.** From the **/home/blockchain/workspace/mqbridge/** directory run the following command in the terminal to peruse the created **Dealer MQ bridge configuration file**:

```
code student_bridgeconfig_dealer.cfg
```

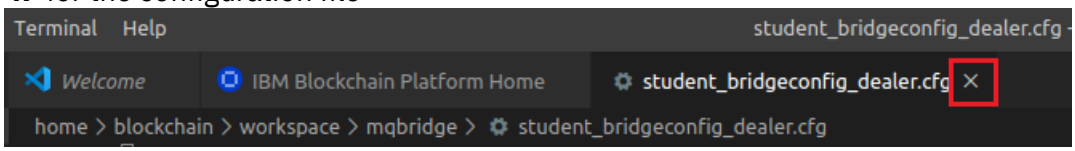


Line 4: the Network Path to the gateway connection profile to be used by a running MQ Bridge for Blockchain, exported via the IBM Blockchain Platform VS Code extension to the subdirectory **"HOME"/workspace/mqbridge/**. There are two files, one for each organisation.
Line: 8-9: the identity to use for sending requests and the organisation MSP of the identity
Line 11-12: Wallet location for identities being used and the MQ queue name that applications can post to, so that the Bridge component can process using a proscribed format, then submit the request via the SDK to that organisations peer.
Line 18-19: Details about the MQ channel the client application can connect to, so it can post messages to a designated queue – and the TCP and listener port address for the Queue Manager.
Line 22: provide details of the Queue Manager name that the application queue resides in, with which to connect to from the IBM MQ Bridge for Blockchain component.

Similarly, a configuration file called **bridgeconfig_regulator.cfg**, (created as preparation for this lab) configures the Regulator bridge component.

Each running IBM MQ Bridge instance connects through separate Fabric gateway connection files; these were earlier exported using the IBM Blockchain Platform for VS Code extension. This file path is reflected in the variable **BCNetworkPath** above.

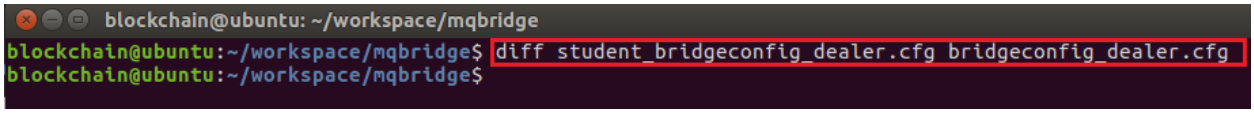
__ **29. Close** the VS Code session when you're finished browsing the file by clicking on the **'x'** for the configuration file



__ **30.** The IBM MQ Bridge components for Dealer and Regulator, use bash scripts to launch their components in the background. The bash scripts (one for Dealer bridge, one for Regulator bridge) look for filenames **bridgeconfig_dealer.cfg** and **bridgeconfig_regulator.cfg** upon launch and use 'known configurations'. You can compare your Dealer bridge file (File: student_bridgeconfig_dealer.cfg) against the master file (bridgeconfig_dealer.cfg) using the Linux **diff** command, to check if your configuration steps completed correctly.

From the terminal window, still in the **/home/blockchain/workspace/mqbridge** subdirectory, paste in the following command:

```
diff  student_bridgeconfig_dealer.cfg  bridgeconfig_dealer.cfg
```



If there is **no output** from the command**,** then you have provided all the right answers, well done – **if there is actual output** from the command, then a wrong value may have been provided in the question and answer configuration steps (but you can still continue anyway).

This concludes this part of the lab. With the MQ and MQ Bridge Components now configured for the Dealer and the Regulator, you will turn your attention to:

- Testing out car transactions using the dealer and regulator applications;
- Showing the messages deposited on the MQ queues (ie with no bridge running yet) in MQ Explorer
- Starting the Dealer bridge you configured and see it connect to the Queue Manager, then forward the stored request to the blockchain, then you will check the blockchain response in MQ Explorer.
- Test the complete end-to-end application to blockchain integration; creating the car (Dealer) and displaying status results – then performing queries from the application and seeing results from the ledger (as Regulator).

Review

In this part of the Lab you have:

- Performed the steps to configure the Dealer's IBM MQ Bridge for Blockchain
- Explored and reviewed the key settings for configuring the Bridge, both for the Dealer and the Regulator bridge configurations.

4 Review and execute the Dealer Car Application

4.1 Introduction

In this section will look at how the bridge connects to IBM MQ and the blockchain and review stored MQ messages when the bridge is not yet running. You will review the Node.JS based dealer application, understand (at a high level) how it sends/receives messages. Finally, you will carry out end-to-end transactions using the Dealer application, creating cars on the blockchain, getting results back to display in the application. The same applies to blockchain queries via the Regulator application. This final section will see both bridge components running, and processing MQ messages.

The files being used for the Dealer application consist of:

dealer.js – This is the main application file. It launches as a CLI application, with a menu to create cars with specific car IDs. Upon selection of an ID, a message containing a JSON request is sent to MQ and deposited on an INPUT queue. This is processed by MQ Bridge, and a transaction request sent to the blockchain network using a blockchain identity. The bridge puts the response from the blockchain into an MQ REPLY queue. The application tracks the reply to the original create car request, and the result is displayed in the dealer application.

inquirechoices.js – this file is the interactive component of the Dealer application that requests inputs – invoked by **dealer.js**, it enables a user to select a car from a list – and uses the npm module *inquirer*

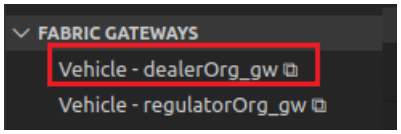
clientcfg.json is a metadata configuration file, which contains a list of JSON transaction request structures, such as Queue Manager names, and transaction selectors that the dealer App uses, when it asks the user to select a car identifier. After a car is selected in the dealer App, it sends a transaction to the blockchain via MQ Bridge. The clientcfg.json file contains entries that consist of 'create' request operations (and some car query/history requests, for the Regulator app later).

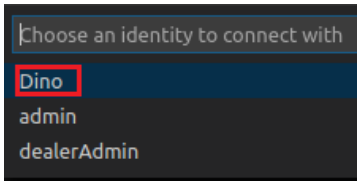
runDealerApp.sh sets up the Queue Manager (QM) connection info. Uses the *MQSERVER* variable to locate the dealer QM and the communication method to be used to connect. Once set, the bash script calls the **dealer.js** Node.js file so the MQ Client APIs can connect to the queue manager.

Steps:

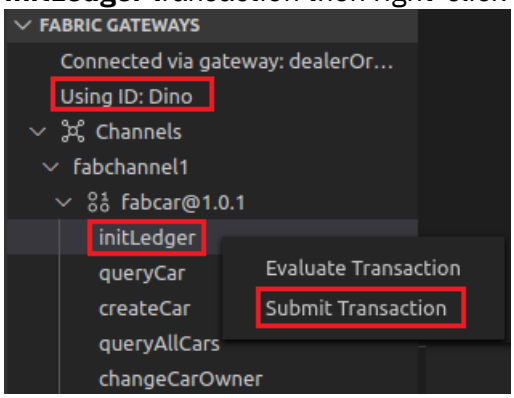
First, initialise the ledger with some sample car data as identity Dino. You do this by executing the Fabcar initLedger smart contract transaction. After, you will run the Dealer Application and see the end-to-end MQ to blockchain integration in action.

__ **31.** Back in the IBM Blockchain Platform VS Code extension, connect to the Dealer gateway **dealerOrg_gw** as **Dino**

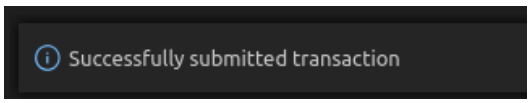




__ **32.** Expand fabchannel1 followed by the Fabcar smart contract and select the **initLedger** transaction then right-click and **Submit Transaction**



When prompted, press **enter three times** to accept the defaults**:** that is, no parameters, no transient data, and accept the default peer-targeting policy respectively – on the bottom right, a message indicates that the transaction was submitted.



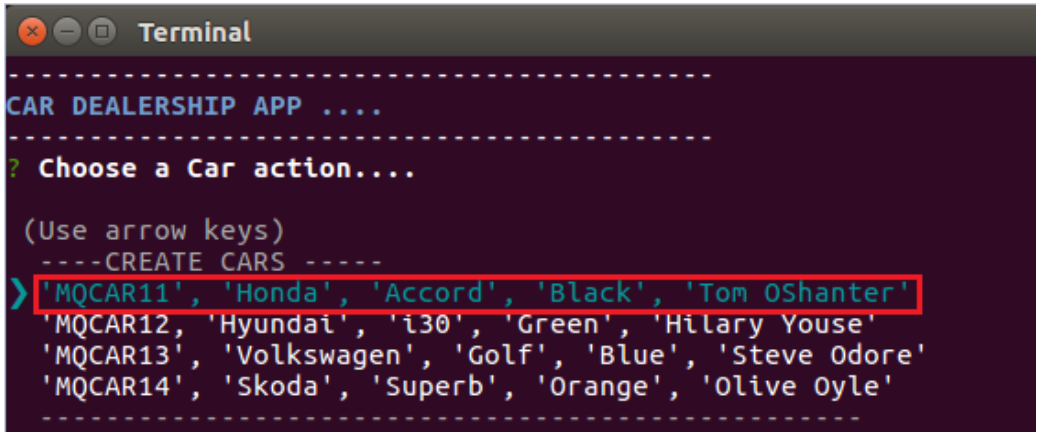
Next, you will start the **Car Dealer application**. Note that you won't yet start the IBM MQ Bridge component for the Dealer organisation; this is so that you can first examine the MQ input request message (by the Dealer app) from inside MQ Explorer, before going on to then start the Bridge. Once up and running, it will process any queued request messages, and handle the interaction with the Dealer's blockchain network.

__ **33.** Return to the main terminal window and start up the Dealer client application (in its own window) from `/home/blockchain/workspace/mqbridge/mqapp`

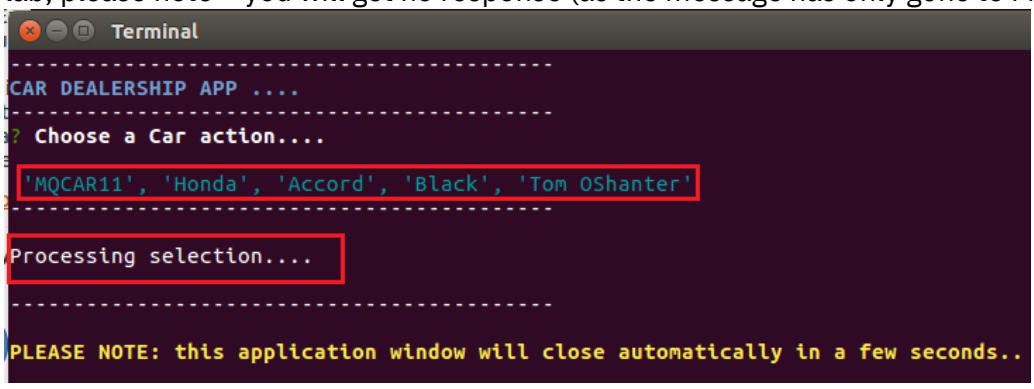
```
cd mqapp
gnome-terminal --tab -e ./runDealerApp.sh --geometry=160x40
```

```
blockchain@ubuntu:~/workspace/mqbridge/mqapp$ gnome-terminal --tab -e ./runDealerApp.sh -geometry=160x40
blockchain@ubuntu:~/workspace/mqbridge/mqapp$
```

__ **34.** Select the first car id **MQCAR11** from the list and press **enter**. (Each menu entry describes some attribute values that the car will be created with)

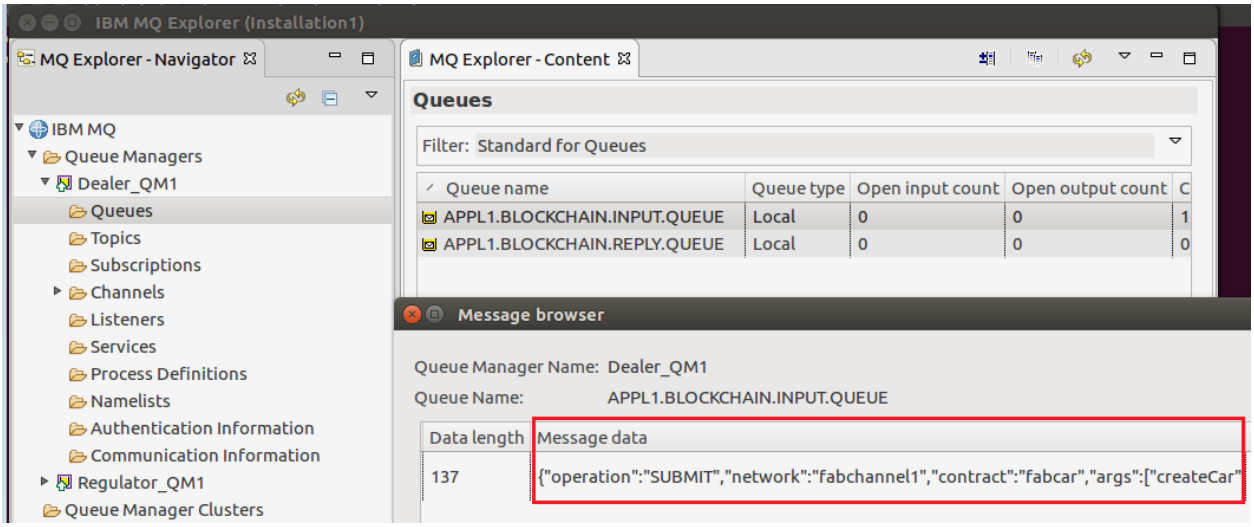


The create car action is confirmed on screen as shown below and submitted as a request to the application queue. The application window will close itself, as part of the lab, please note – you will get no response (as the message has only gone to MQ so far)



At this point, the queued application message should be visible in MQ Explorer.

__ **35.** Switch back to MQ Explorer. Select queue manager 'Dealer_QM1' followed by the input queue APPL1.BLOCKCHAIN.INPUT.QUEUE and right click….**Browse Messages** then **scroll** to the right to see the column **Message Data,** it contains the smart contract JSON request.

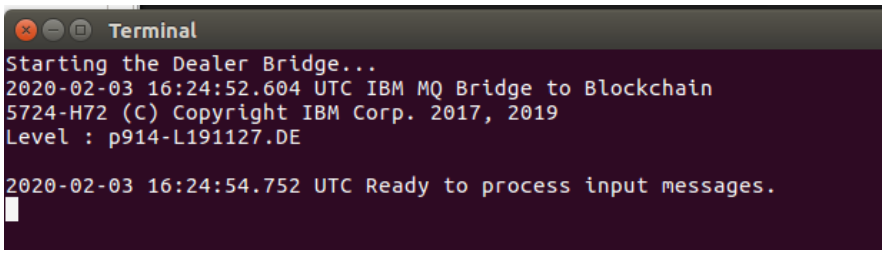


Click **close** to close the Message browser.

__ **36.** Back in the terminal window, launch the Dealer Bridge component in the background – copy/paste this command (the bash script is one directory level up):

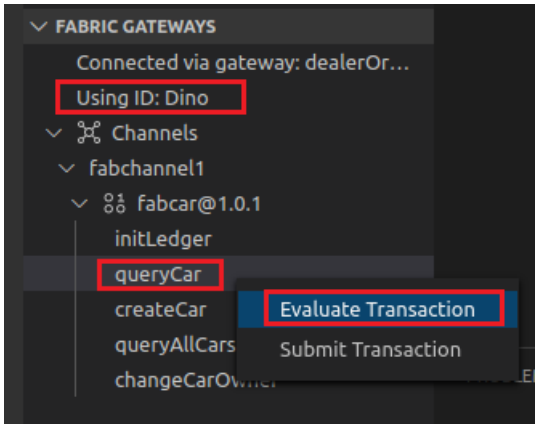
```
gnome-terminal --tab -e ../runDealerBridge.sh --geometry=100x10
```

```
gnome-terminal --tab -e  ../runDealerBridge.sh --geometry=100x10
```



The Dealer bridge is now running in a background window, bottom left, and ready to process queue messages for the Dealer's network. The queued message from earlier is automatically processed - and as a result, should create a new car record with MQCAR11 on the blockchain. To check the car was created, return to the IBM Blockchain Platform VS Code extension in VS Code.

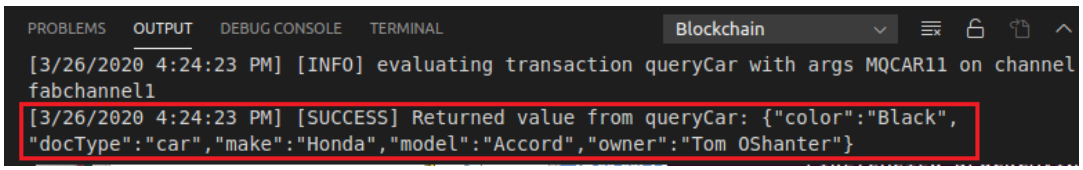
__ **37.** In the **Fabric Gateways** view, right click on the transaction **queryCar** and right-click ... **Evaluate Transaction**



__ **38.** Ensure the parameters you enter are as follows (in [] square brackets):

```
["MQCAR11"]
```

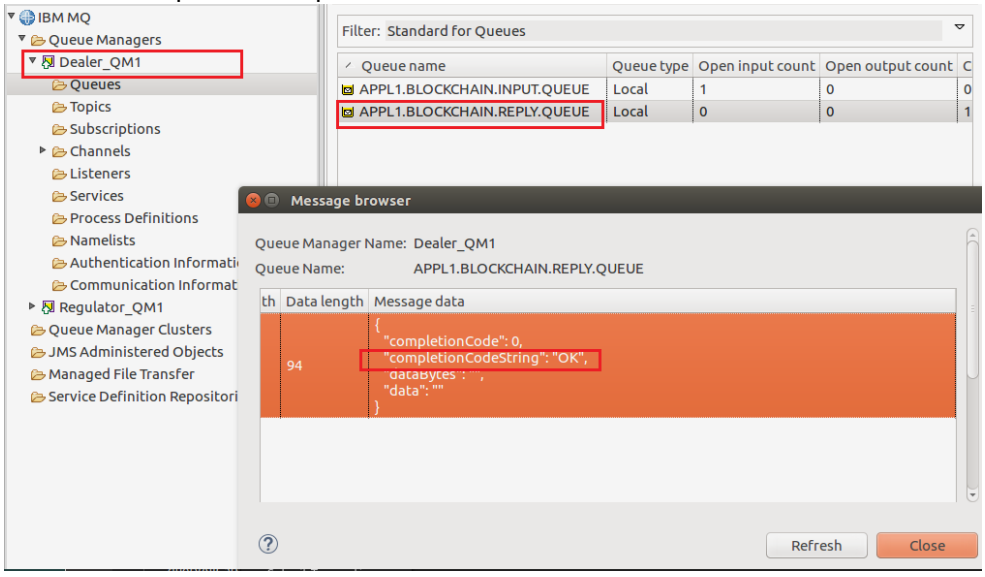
Press **enter twice**, to accept the defaults for the transient data and peer targeting prompts. Review the output in the Output pane – you should see the query result shows that the car was created successfully, from the MQ request message.



Note that we will also have a blockchain response now – because the transaction was submitted successfully. The Dealer Bridge has placed this response on the REPLY queue (i.e. in APPL1.BLOCKCHAIN.REPLY.QUEUE), for the Dealer App to consume. Let's look at the response in MQ Explorer.

__ **39.** In **MQ Explorer**, return to the Queue list in the Queue Manager 'Dealer_QM1', select the APPL1.BLOCKCHAIN.REPLY.QUEUE queue and right click...**Browse Messages** –

__ **40. Scroll** to the right to see the message data – and it shows the response that was returned from the fabcar smart contract, ready for processing by an application. It shows a completion response of 'OK'.

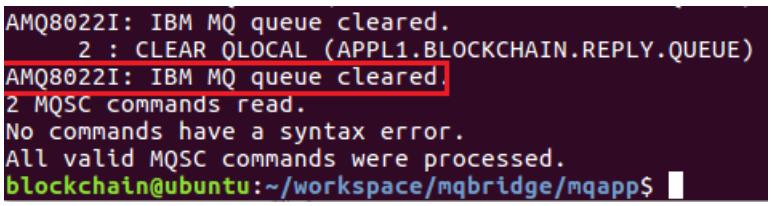


Click on **Close** to close the message browser window.

At this point - you've seen how the messages appear in MQ Explorer. Next, you will do an end-to-end transaction within the application by creating another new car record. As an application could have many messages on the same reply queue, the application needs to process transaction responses against the original car creation request.

__ **41.** Switch back to the terminal, and from the **`mqapp`** subdirectory, enter the following command to clear the REPLY queue. We want to discard this response for now – the following bash script executes MQ queue operations to clear queues. Note the leading two '..' is important here.

```
../clearQs.sh
```

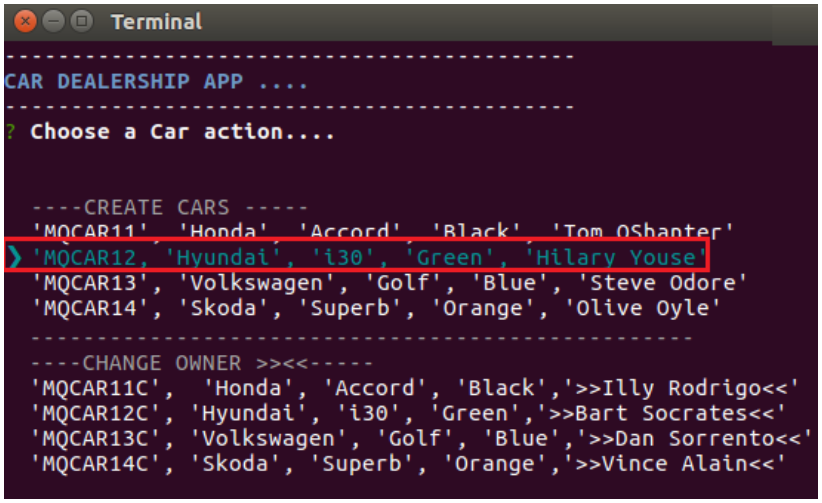


__ **42.** From the main terminal, launch the Car Dealer application from the `mqapp` directory using the following command sequence:

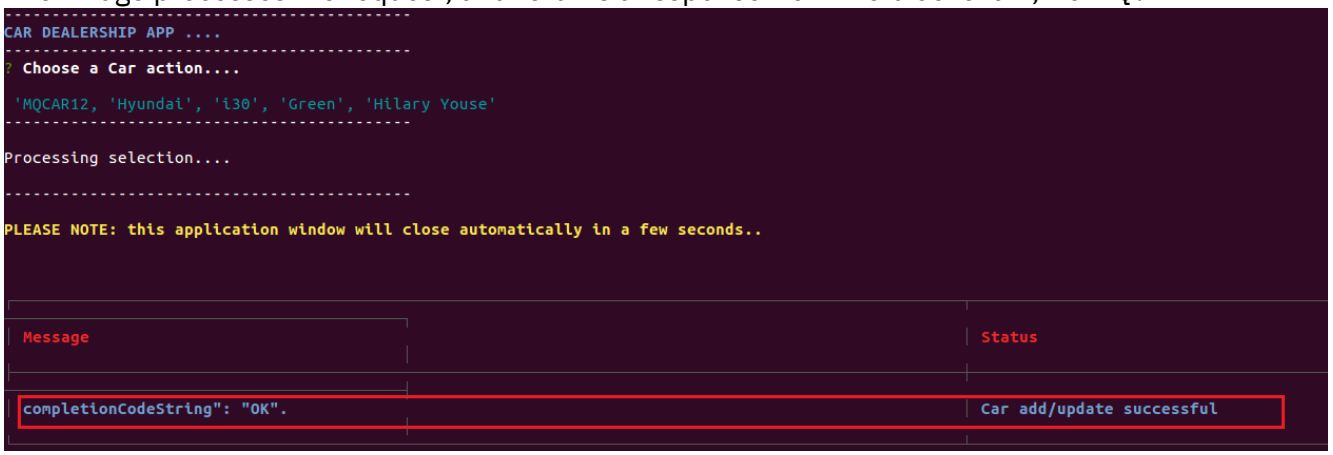
```
gnome-terminal --tab -e ./runDealerApp.sh --geometry=160x40
```

```
blockchain@ubuntu:~/workspace/mq/mqapp$ gnome-terminal --tab -e  ./runDealerApp.sh --geometry=160x40
blockchain@ubuntu:~/workspace/mq/mqapp$
```

This time select car **MQCAR12** from the list and press **enter** – again, it describes some attribute values that this car will be created with.

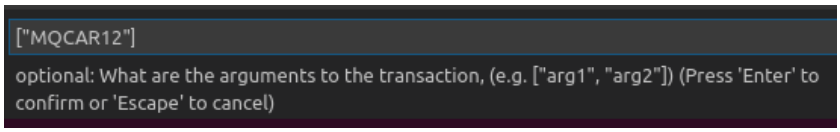


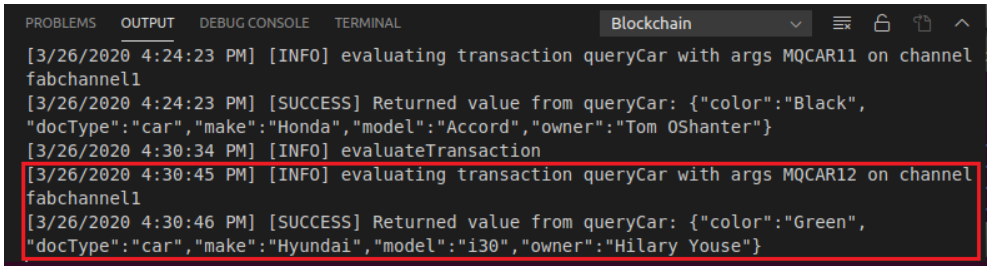
The Bridge processes the request, and returns a response from the blockchain, via MQ:



__ **43.** Next, return to the **IBM Blockchain Platform VS Code** extension icon – you want to verify the creation of car **MQCAR12** using the smart contract query. Execute the transaction **queryCar** as the Dealer Org by performing right-click…**Evaluate** in the smart contract view, providing a parameter exactly as shown below:

```
["MQCAR12"]
```





You have now successfully demonstrated end-to-end integration, all the way back to the application itself. You have also verified the creation of car assets; the first processed after the Bridge component was started; the second, as an end-to-end transaction. You then performed a direct query on the ledger to verify this.

This concludes this section of the lab.

Review

In this part of the Lab you have:

- Successfully examined MQ messages and requests (as well as responses) processing by the Dealer organisation's IBM MQ Bridge for Blockchain.
- Successfully carried out end-to-end transactions as the Dealer user Dino and seen the end-to-end integration at play.

5 Review and execute the Regulator Reporting Application

5.1 Introduction

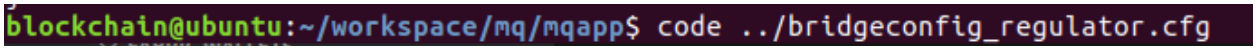
In this section, you will now look at the Regulator perspective and its application environment. The regulator's role on the network is to perform compliance checks, eg cross-verify car ownership records on the shared ledger. Like the Dealer, the Regulator application clients use MQ APIs to talk to IBM MQ. Before you use the Regulator App, you first need to start the Regulator organisation's IBM MQ Bridge for blockchain.

The Regulator Bridge configuration contains different parameters to that of the Dealer bridge configuration details.

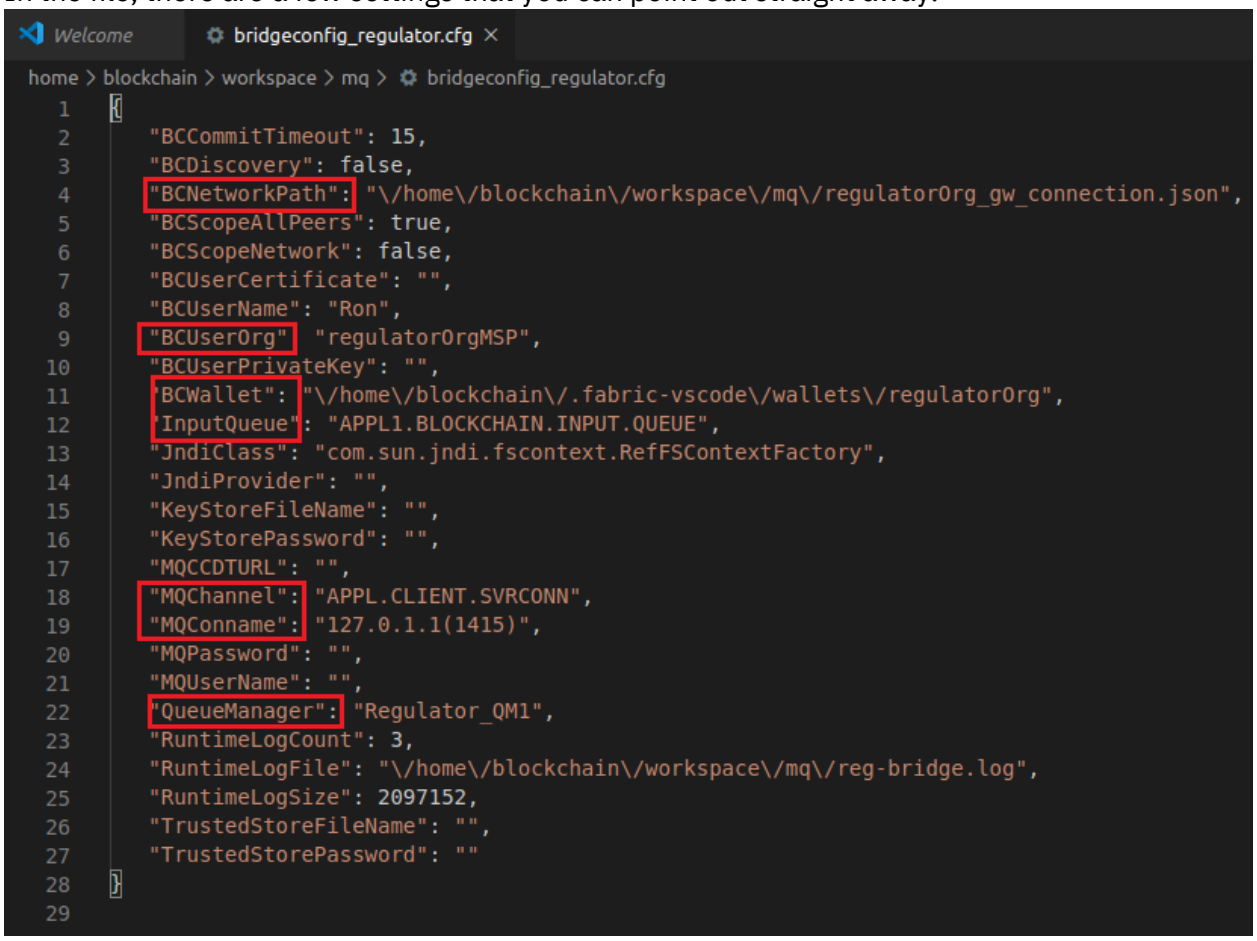
Steps:

__ **44.** From the terminal window in subdirectory `/home/blockchain/workspace/mqbridge/mqapp`, open the Regulator bridge configuration file in VS Code:

```
code ../bridgeconfig_regulator.cfg
```



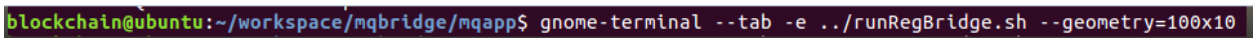
In the file, there are a few settings that you can point out straight away:



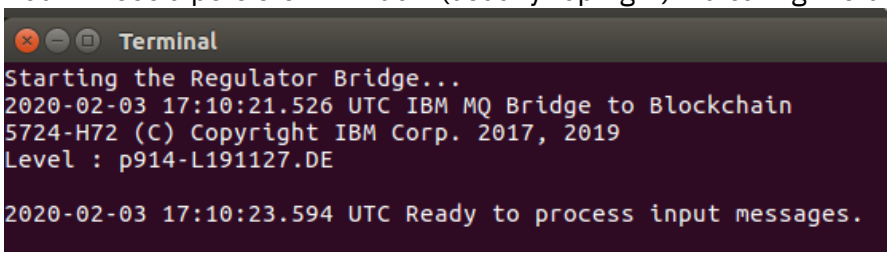
Line 4: The location of the Regulator's Fabric gateway connection profile; this is required for the MQ Bridge to know of the Regulator's member blockchain network nodes
Line 9: The organisational MSP for the regulator
Line 11-12: The wallet location (containing identities) and the INPUT queue that the bridge will examine for application requests in the Regulator's application queue manager.
Line 18-19: The Channel connection details for the application clients to make a connection to this queue manager – this is on a different port (1415) to that previous shown for the Dealer, not least because you are running on the same machine 😊
Line 22: Regulator's Queue Manager name, that you created earlier in the lab.

__ **45.** Close the configuration file and return to the main terminal window, and still in the **mqapp** subdirectory, launch the regulator IBM MQ Bridge component in the background as follows:

```
gnome-terminal --tab -e ../runRegBridge.sh --geometry=100x10
```



You will see a persistent window (usually top right) indicating the bridge is ready.



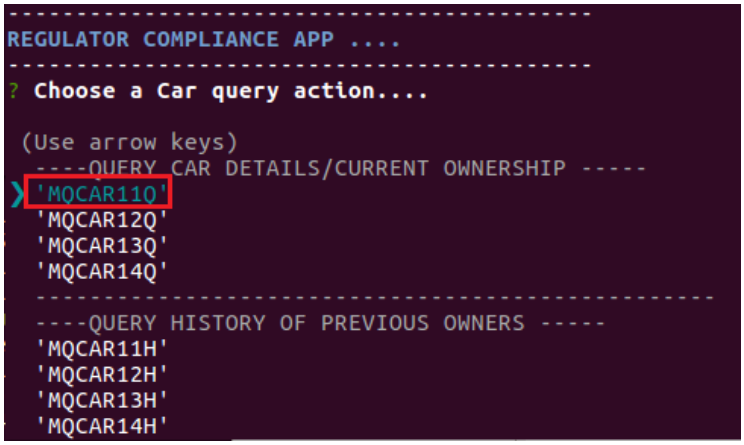
(Tip: If the window closes in a few seconds, it's likely that you didn't export the Regulator's gateway connection profile in the VS Code extension, per step 16 earlier)

__ **46.** Back in the main terminal, launch the Regulator App (in a new window) as follows:

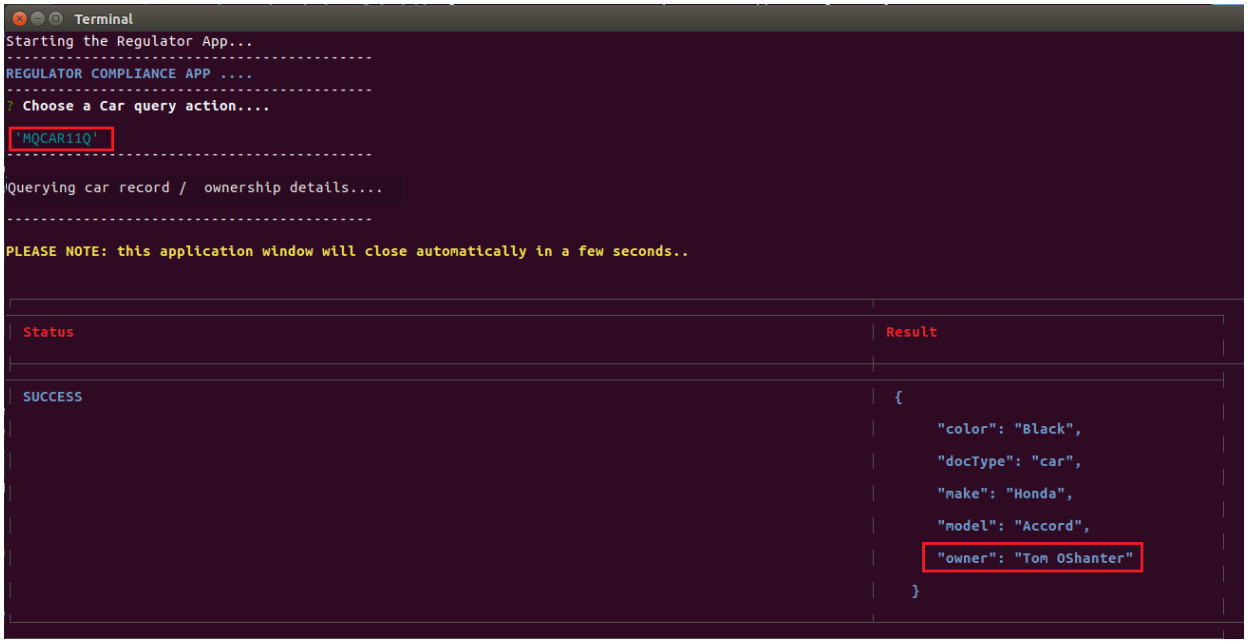
```
gnome-terminal --tab -e ./runRegApp.sh --geometry=160x40
```

```
blockchain@ubuntu:~/workspace/mqbridge/mqapp$ gnome-terminal --tab -e ./runRegApp.sh --geometry=160x40
```

__ **47.** In the Regulator App, choose to query the current ownership record of the first car, **MQCAR11** (Q for Query) and press **enter**



The output confirms the details of the car and ownership – take note of who owns the car at this present time (bottom right). (Note that the application window closes itself after approx. 5-6 seconds – you can re-run the query app at any time).



Just like the Dealer application earlier, query requests for the blockchain get posted to a designated Regulator INPUT queue in IBM MQ. Again, these are processed by the IBM MQ Bridge for Blockchain. After submitting the smart contract query transaction, the bridge returns the results for the car ID queried, and displays the information in the Regulator App.

That concludes this section of the lab.

Review

In this part of the Lab you have:

- Successfully examined the configuration of the Regulator organisation's IBM MQ Bridge for Blockchain.
- Successfully carried out end-to-end transactions as the Regulator user Ron and seen the end-to-end integration at play.

6 Change Car Ownership as Dealer, verify as Regulator

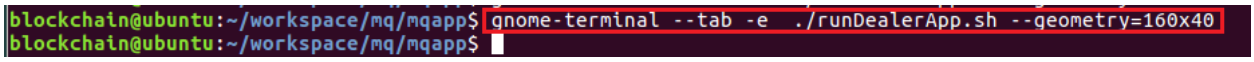
6.1 Introduction

This section shows a typical lifecycle change in the vehicle lifecycle network – i.e. car ownership changes. The Dealer App is also used to update the ownership records. The smart contract transaction that performs this in Fabcar is called `changeCarOwner` . Later, the Regulator will query the car's details on the blockchain, to see the change of ownership on the ledger.

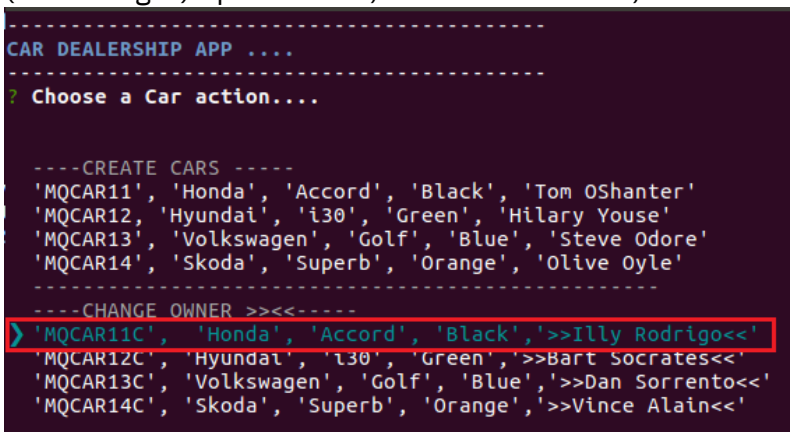
Steps:

__ **48.** From the main terminal window, launch the **Dealer App** this time:

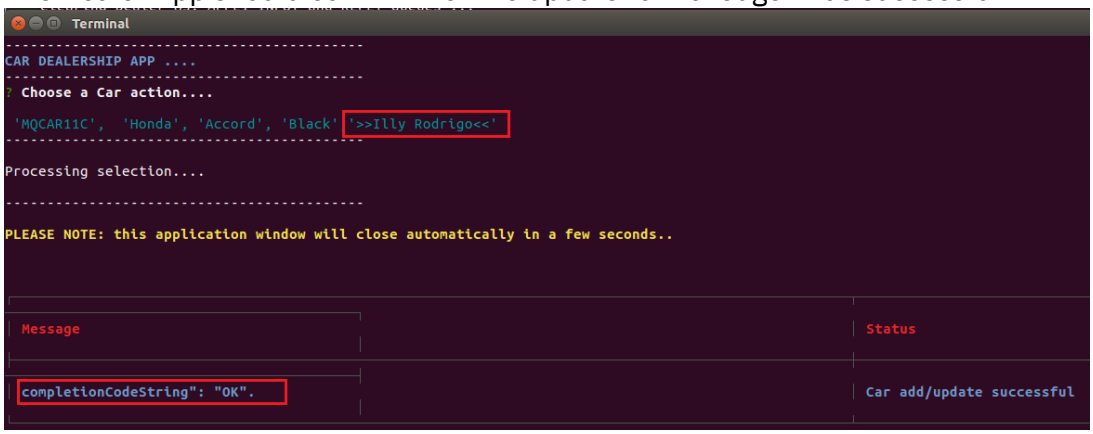
```
gnome-terminal --tab -e ./runDealerApp.sh --geometry=160x40
```



__ **49.** Under the **CHANGE OWNER** menu (the 2nd menu), select the first car **MQCAR11C** (for "change") - press **enter**; the >>>< chevrons, means the new owner is 'Illy Rodrigo'



The Dealer App should confirm that the update to the ledger was successful



The message window will close automatically after approx. 5 seconds.

Now let's check the car record as identity Ron, using the Regulator App

__ **50.** From the existing command line, launch the Regulator App as follows:

```
gnome-terminal --tab -e ./runRegApp.sh --geometry=160x40
```

```
blockchain@ubuntu:~/workspace/mq/mqapp$ gnome-terminal --tab -e  ./runRegApp.sh --geometry=160x40
```

__ **51.** In the Regulator App, once again, choose to query the record of the first car, **MQCAR11** and press **enter**

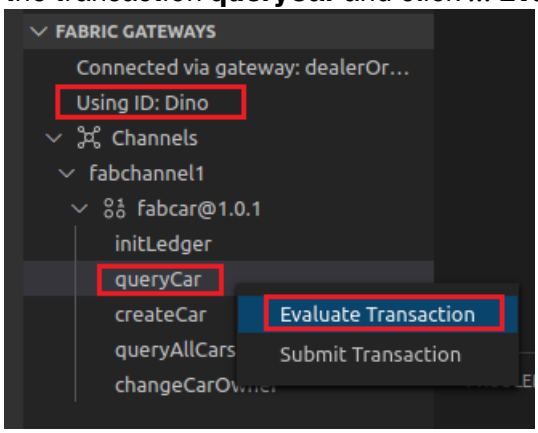
```
Starting the Regulator App...
----------------------------------------
REGULATOR COMPLIANCE APP ....
----------------------------------------
? Choose a Car query action....

  ----QUERY CAR DETAILS/CURRENT OWNERSHIP -----
❯ 'MQCAR11Q'
  'MQCAR12Q'
  'MQCAR13Q'
  'MQCAR14Q'
  --------------------------------------------------
  ----QUERY HISTORY OF PREVIOUS OWNERS -----
  'MQCAR11H'
  'MQCAR12H'
  'MQCAR13H'
  'MQCAR14H'
```

The output confirms the details of the car and ownership – it is now owned by its new owner, 'Illy Rodrigo'. Again, the application will close itself in approx. 5 seconds.

```
Terminal
Starting the Regulator App...
----------------------------------------
REGULATOR COMPLIANCE APP ....
----------------------------------------
? Choose a Car query action....

'MQCAR11Q'
----------------------------------------

Querying car record /  ownership details....

----------------------------------------

PLEASE NOTE: this application window will close automatically in a few seconds..

| Status  | Result
| SUCCESS | {
|         |     "color": "Black",
|         |     "docType": "car",
|         |     "make": "Honda",
|         |     "model": "Accord",
|         |     "owner": "Illy Rodrigo"
```

__ **52.** Let's verify this from the IBM Blockchain Platform VS Code extension, by executing a query on car 'MQCAR11' and verify the same change ownership transaction performed by the application matches what you would expect. **Return** to the VS Code extension.

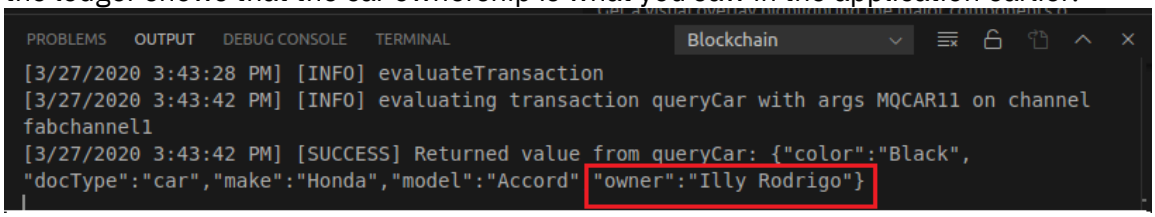
__ **53.** In the **Fabric Gateways** view, still connected as the Dealer gateway, right-click on the transaction **queryCar** and click ... **Evaluate Transaction**



__ **54.** Ensure the parameters you enter are as follows (in [] square brackets):

```
["MQCAR11"]
```

Press **enter twice** to accept the defaults for the transient data and peer targeting prompts. Review the output in the Output pane – you should see the query result from the ledger shows that the car ownership is what you saw in the application earlier.



You've now verified the end-to-end changes made by the application to the system of record (the ledger); once from the application and once using the smart contract transaction via the VS Code extension.

This concludes this section of the lab.

Review

In this part of the Lab you have:

- Successfully performed end-to-end transactions as the Dealer user Dino and seen the end-to-end integration at play to show the current car ownership as supplied by the blockchain system of record.

7 Audit History of Previous Ownership as Regulator

7.1 Introduction

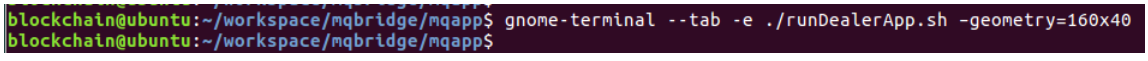
In the last section of this lab, you carry out another typical application function: querying the history of previous owners for a vehicle. The Regulator uses their application to check on the history of previous owners of a selected car.

In the steps below, you will use the Dealer App again, to update the ownership record of car **MQCAR11** , return ownership back to owner "Tom O Shanter", then perform another ownership change to "Illy Rodrigo" – this conveniently provides a trail of 3 previous owners (Tom, Illy, and Tom again) and the current owner – Illy Rodrigo. The Regulator can then query the car on the blockchain, to see the history of previous owners.

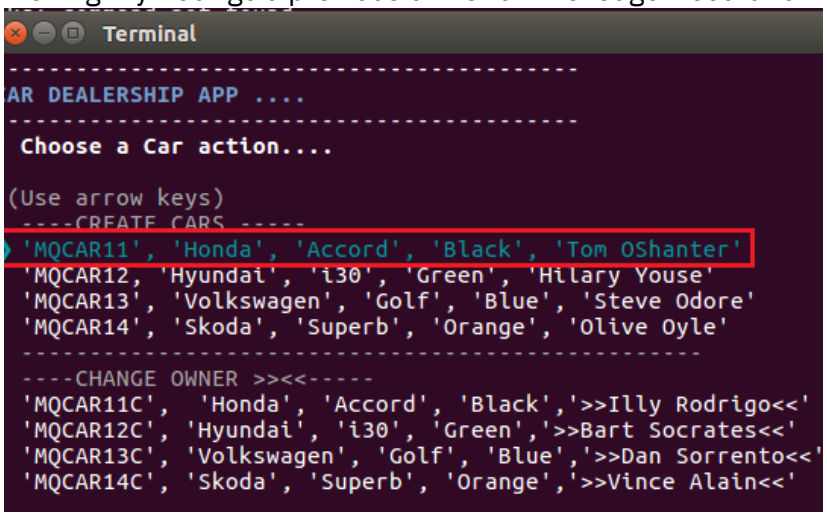
Steps:

__ **55.** From the main terminal window, still in the `mqapp` subdirectory, launch the **Dealer App**:

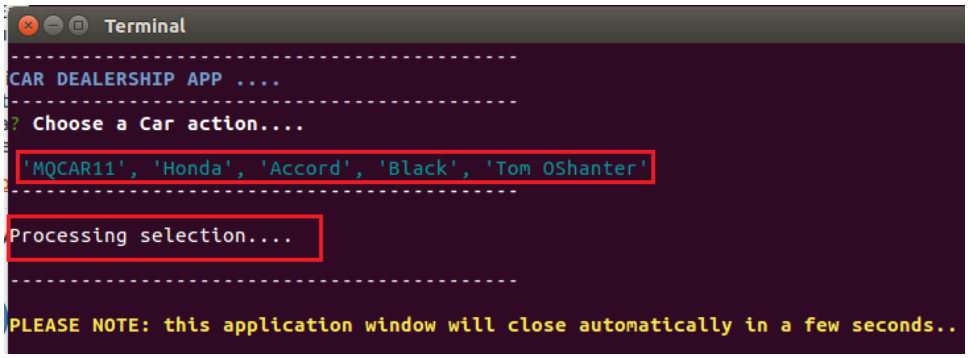
```
gnome-terminal --tab -e ./runDealerApp.sh --geometry=160x40
```



Select the car **MQCAR11** and hit **enter** (Note: this is simply running the "create" transaction again, but it has the effect of updating the existing car record for MQCAR11 – in the process, it will update the current owner back to **Tom OShanter** - and thus making Illy Rodrigo a previous owner on the ledger record for this car):



Car MQCAR11 gets updated on the ledger, and Illy Rodrigo becomes a previous owner in the car's history. Again, note that the application window closes itself.

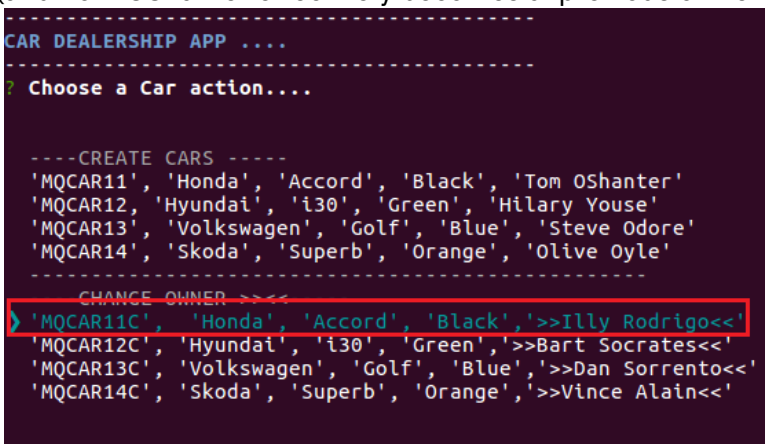


__ **56.** Launch the **Dealer app** again:

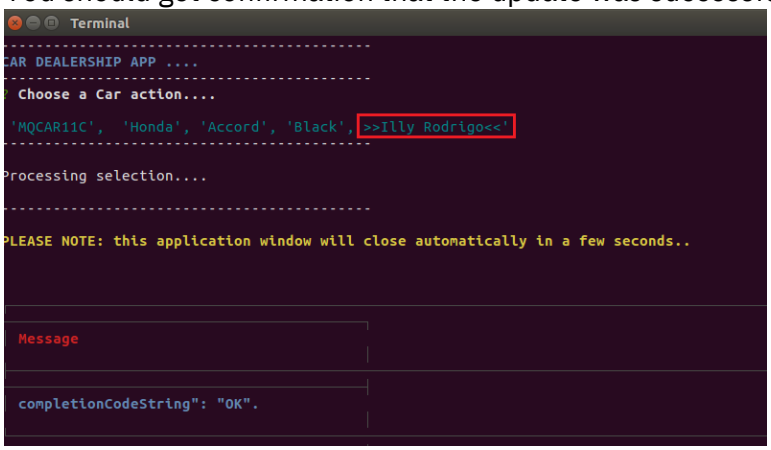
```
gnome-terminal --tab -e ./runDealerApp.sh --geometry=160x40
```

```
blockchain@ubuntu:~/workspace/mqbridge/mqapp$ gnome-terminal --tab -e ./runDealerApp.sh -geometry=160x40
blockchain@ubuntu:~/workspace/mqbridge/mqapp$
```

__ **57.** Select car **MQCAR11C** under the **Change Owner** sub-menu and hit **enter** – this time, the current owner becomes **Illy Rodrigo** once again, adding another ownership change (and Tom OShanter effectively becomes a 'previous owner').



You should get confirmation that the update was successful.



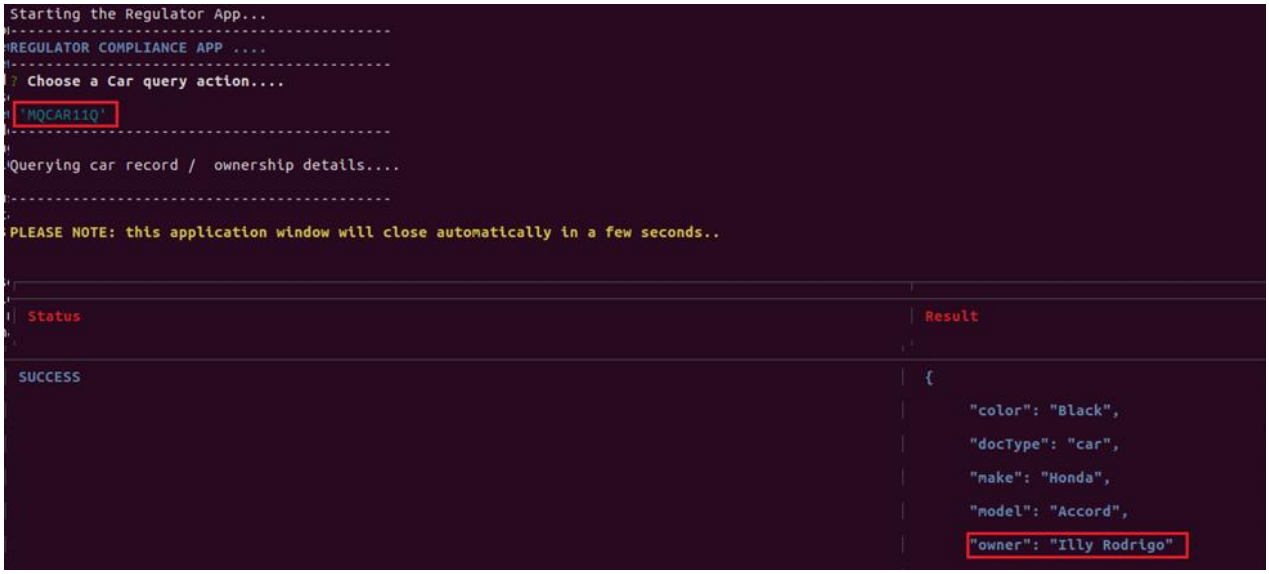
Next, you can check out the ownership history of car MQCAR11 - as the Regulator. It should reveal the history of previous owners, as a result of the transactions earlier.

__ **58.** Launch the Regulator App from the terminal window as follows:

```
gnome-terminal --tab -e ./runRegApp.sh --geometry=160x40
```

```
blockchain@ubuntu:~/workspace/mqbridge/mqapp$ gnome-terminal --tab -e ./runRegApp.sh --geometry=160x40
```

__ **59.** Inside the Regulator App, choose to query the current ownership record of the first car, **MQCAR11** (Q for Query) and press **enter** – it will reveal Illy Rodrigo as the current owner.

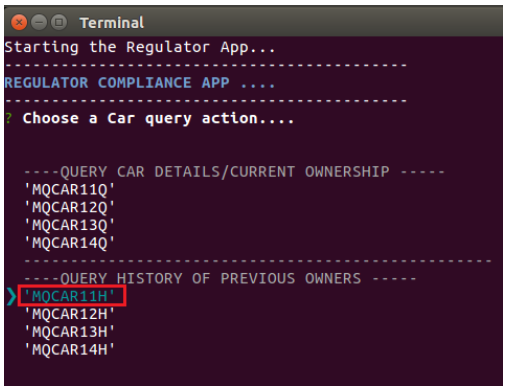


__ **60.** Finally, launch the Regulator App once again, to query the history of previous owners

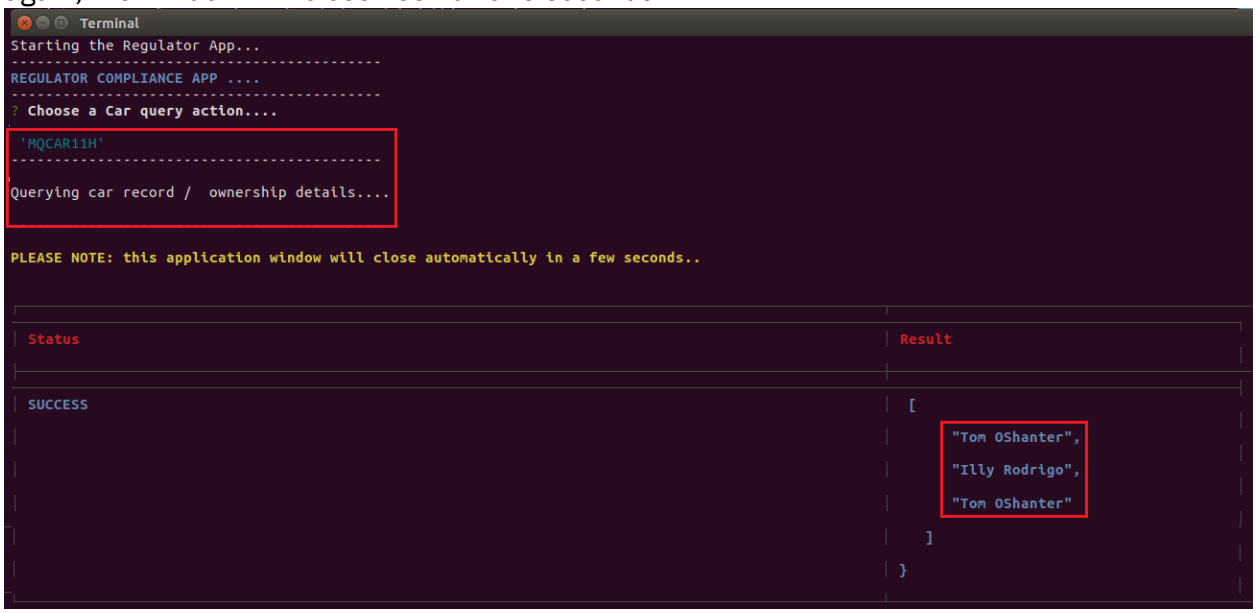
```
gnome-terminal --tab -e ./runRegApp.sh --geometry=160x40
```

```
blockchain@ubuntu:~/workspace/mqbridge/mqapp$ gnome-terminal --tab -e ./runRegApp.sh --geometry=160x40
```

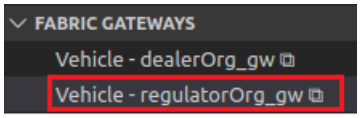
__ **61.** Under 'QUERY HISTORY OF PREVIOUS OWNERS' submenu – select 'MQCAR11H' (H for history) and hit **enter**:

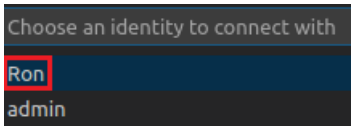


Once again, the query request gets processed, the application consumes the response provided by the bridge. You will get a history of previous owners displayed inside the application – there should be **three previous owners** on the right (see below). Note again, the window will close itself after 5 seconds

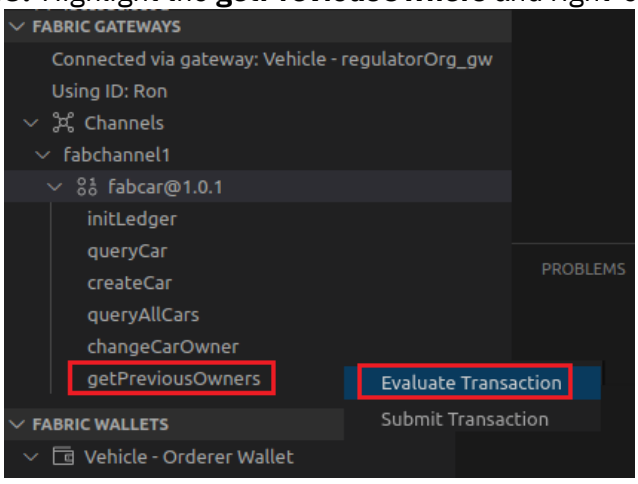


__ **62.** Return to the IBM Blockchain VS Code extension, and under the **Fabric Gateways** view, disconnect from the Dealer's gateway, then click on **Vehicle - regulatorOrg_gw**, and choose **Ron** as the identity to connect with:



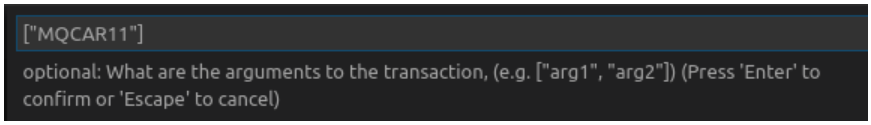


__ **63.** Highlight the **getPreviousOwners** and right-click....**Evaluate**

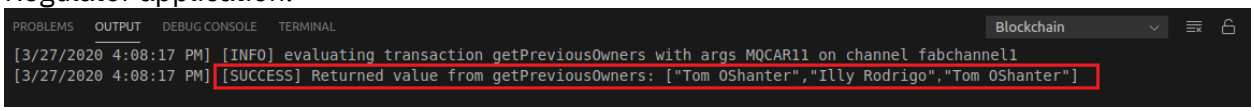


__ **64.** When prompted, Ensure the parameters you enter are as follows (in [] square brackets):

```
["MQCAR11"]
```



Press **enter twice** to accept the defaults for the transient data and peer targeting prompts. Review the output in the Output pane – you should see the query result shows that the previous ownership history matches exactly what you saw in the Regulator application.



You've now seen two perspectives to verify the application ownership history of car MQCAR11, as a result of ownership changes ; one via the application query response (that is returned via MQ); the other, by directly querying the ledger via smart contract transaction `getPreviousOwners`.

Optional Lab - try out a different car ID from the **Dealer** app: that is: 1) create a car (say MQCAR13 from the menu), then 2) change car ownership as shown and finally 3) verify current and previous ownership history, using the Regulator app menu and verify using the smart contract query using `getPreviousOwners`.

This concludes this section of the lab.

Review

In this part of the Lab you have:

- Successfully created a chain of car ownership history as Dino using the Dealer App.
- Successfully audited the car's ownership history on the blockchain ledger, firstly through the Regulator Application (routes requests via IBM MQ and the IBM MQ Bridge for Blockchain) and then querying the ledger directly using the IBM Blockchain Platform VS Code extension, and a smart contract query.

:

8 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: Lab Environment

This appendix provides more information on how this lab environment is configured in this environment.

As mentioned in the introduction of the lab guide, the sample applications (Dealer and Regulator App) are Node.JS based samples, that consume the IBM MQ APIs to be able to put and get messages from/to the IBM MQ Advanced Server queues – more info on that here https://github.com/ibm-messaging/mq-mqi-nodejs/blob/master/README.md .

The Regulator and Dealer (docker based) member networks are built and configured by Ansible and is all located local see more https://github.com/IBM-Blockchain/ansible-role-blockchain-platform-manager/blob/master/README.md .

The bridge configuration tool to create configurations is called **runmqbcb**– it asks a series of questions to create configuration files, and based on parameters set for the respective Dealer/Regulator bridge instances. See https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q130880_.htm#q130880_ and for more info on the configuration tool, see https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q130890_.htm

More information on the configuration tool and how to create IBM MQ Bridge for Blockchain configuration files is described in the next Appendix.

Appendix 2: Creating the MQ Bridge Configuration file

The IBM MQ Bridge MQ component has a configuration tool to generate its Bridge configuration file for an organisation's member network. To answer the questions asked by the interactive CLI tool, you need the parameters from your blockchain network credentials file, and from your IBM MQ Advanced queue manager that your application ultimately interacts with. Once the IBM Bridge Component is installed, you would run the tool using the following command (once you've set your environment using `setmqenv` etc) eg.

```
runmqbcb -o config_file_name.cfg
```

As you've seen for the Dealer bridge, it offers some default values, for given parameter fields - these are shown inside the square brackets []. As you answer the questions, you can press `Enter` to accept existing values, press `Space` then `Enter` to clear existing values (eg if you have re-run the tool, pointing at the same output config file), and type inside the brackets then press `Enter` to add new values. You can separate lists of values by commas, or by entering each value on a new line. A blank line ends the list. Note: You cannot edit the existing values. You can keep, replace, or clear them.

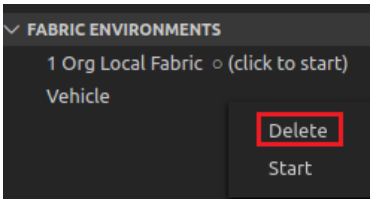
You'll need to enter values for the connection to your IBM MQ Advanced queue manager. Minimum values that are needed for the connection are the queue manager name, and the names of the bridge input and reply queues. For connections to remote queue managers, you will also need MQ Channel and MQ Conname (host address and port where the queue manager is running). To use TLS, for connecting to IBM MQ Advanced Server - you must use JNDI or CCDT and specify MQ CCDT URL or JNDI implementation class and JNDI provider URL accordingly.

Appendix 3: Teardown custom Vehicle Lifecycle network

The following steps are used to tear down the custom Fabric network that contains the Regulator / Dealer network.

1. In VS Code, disconnect any connected Fabric Environment.

2. Right click on any Fabric Environment for DealerOrg and RegulatorOrg and **Delete Environment** (Remember to click **yes** in the bottom corner of the screen)



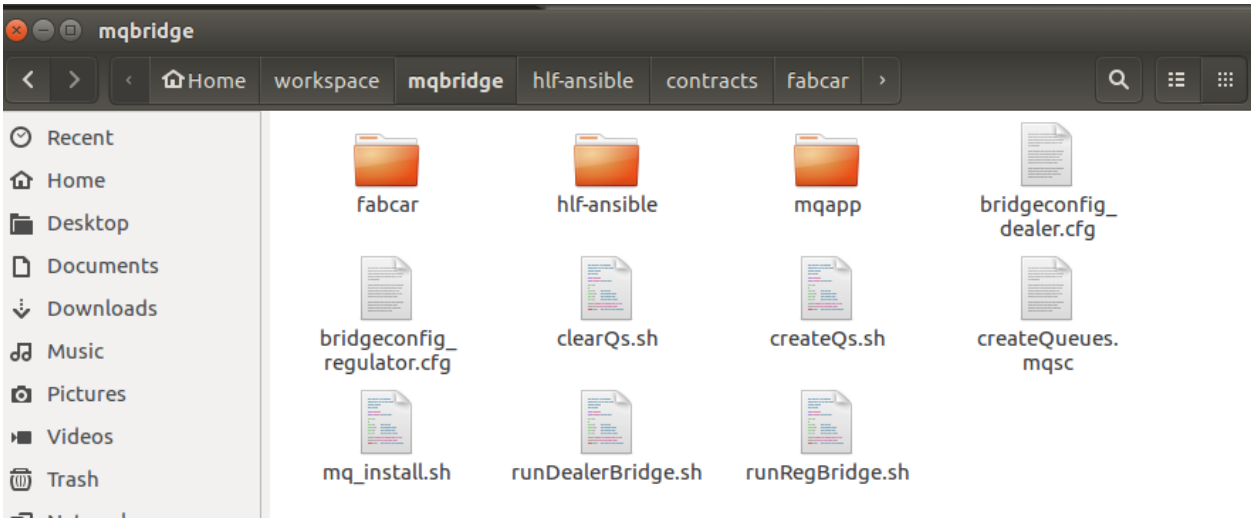
3. In a terminal window, run the following commands to clear up the “custom” Fabric:
   cd ~/workspace/mqbridge/hlf-ansible
   ./teardown.sh

4. Observe that the containers are stopped and removed.

Appendix 4: Description of files used in this lab

List of files and description of customisations applied by folder

Under the HOME/workspace/monitoring subdirectory, a few folders exist containing the configuration / client files used to complete this IBM Blockchain Platform monitoring lab



First let's describe the files/folders in the main directory (above) alphabetically, then their contents in turn:

File	Description	Comments
bridgeconfig_dealer.cfg	This is the IBM MQ Bridge for Blockchain configuration file for the Dealer organisation. It is created as a result of launching the IBM MQ Bridge for Blockchain configuration tool, as described in more detail in Appendix 2.	
bridgeconfig_regulator.cfg	As above, but for the Regulator organisation. Again, this is described in more detail in Appendix 2.	
clearQs.sh	Well, it clears Queues 😊 – there is a lab step where this script is launched – as at that point the Queue needs clearing before proceeding.	
createQMgrs.sh	Bash script to create the Queue Managers	
createQs.sh	This script runs a sequence of operations from an MQ script files, to	

	create queues automatically and conveniently, with the correct names and also the Listeners and Server Channel connection definitions.	
createQueues.mqsc	An MQ SCript file (.mqsc suffix), that carries out some queue creations and associated security / authorization updates that are required	
createLstnr_Dealer.mqsc	Creates the Listener (port 1414) and Server Channel for Dealer_QM1 queue manager	
createLstnr_Regulator.mqsc	Creates the Listener (port 1415) and Server Channel for Regulator_QM1 queue manager	createLstnr_Dealer.mqsc
mq_install.sh	This script was used to install the IBM MQ Advanced Server solution, including the IBM MQ Bridge for Blockchain component	
runDealerBridge.sh	This launches the IBM MQ Bridge for Blockchain listener, so that it can manage requests/responses between the Dealer's MQ queues, and the Dealer's blockchain network.	
runRegBridge.sh	Exactly as above, except launches the bridge component for the Regulator organisation and the Regulator's MQ queues and its blockchain network.	

Folder: fabcar: This folder contains the source Hyperledger Fabric Sample Fabcar client app. It has bash script wrappers, to generate a transaction workload against the smart contract deployed to IBM Blockchain Platform – all files are in the 'javascript' sub-folder

File	Description	Comments
fabcars.js	Fabcar Smart contract source code – contains a series of transactions to initialise the ledger, create or update cars, and query car ownership or the history of previous owners	The fabcar101.cds file is already packaged and this is imported from the hlf-ansible subdirectory under 'mqbridge' please note
package.json	The Node.JS package file, describing package name, npm dependencies etc etc.	

Folder: hlf-ansible: This is the ansible playbook directory – it contains the ansible playbook `site.yml` that builds the Hyperledger Fabric network containing two organisations.

File/Folder	Description	Comments
contracts	Contains the fabcar@101.cds file and the source code it was built from. ownership or the history of previous owners	The fabcar101.cds file is already packaged so it can be installed/instantiated by the ansible playbook.
README.md	Ansible readme file	
deploy.sh	The bash script that builds the Ansible docker image, launches it and performs the installation of the two-organisation Fabric network	Builds both the Dealer and the Regulator blockchain environments for this MQ Bridge lab.
requirements.yml	Describes the Ansible role used to interpret the playbook instructions	
site.yml	This the custom ansible playbook that created the Fabric network and associated artifacts. Can be torn down using **teardown.sh**	This creates the Nodes, gateways and wallets, to enable a single import of the Vehicle Lifecycle Fabric environment, comprising two organisations.
start.sh	Docker start the dockerized two organisation network, if it was previously stopped or if you are at the beginning of the lab	
stop.sh	Docker stop the dockerized two organisation network, if it was previously started.	
teardown.sh	Tears down the dockerized Fabric network, including its related images, and including pruning old chaincode images.	The script also removes the nodes, wallets and gateways and the ansible subdirectory ‘vehicle’ – so that a new deploy can be carried out.

Folder: mqapp: This contains all the Dealer and Regulator Application client code. It is from here that the Node.JS applications are launched and where the Node.JS dependencies are installed.

File/Folder	Description	Comments
clientcfg.json	This file contains the matching Fabric blockchain JSON requests, that are matched, when a user using the respective Dealer or Regulator Apps select a car (from the menu) to create or query: a selected car matches the corresponding operation in this .json file – and that operation (in an MQ message) is what gets posted by the IBM MQ Bridge for Blockchain component to the blockchain.	The Node.JS Apps use menu selection, then match on the entry in clientcfg.json, to issue the correct JSON operation to the IBM MQ Bridge for Blockchain
dealer.js	This is the Node.JS application for the Dealer App	This gets launched by runDealerApp.sh
inquirechoices.js	This contains the inquirer module that enables the menu and menu prompts to be created. The menu in this file is customised for the Dealer App	
package.json	The Node.JS application package definition	
reg-inquirechoices.js	Exactly the same as inquirechoices.js, except it has the Regulator App menu choices.	
regulator.js	This is the Node.JS application for the Regulator App.	This gets launched by runRegApp.sh
runDealerApp.sh	This is the bash 'wrapper' script that launches the Dealer App ; it also sets the MQ connection information, so the application knows how to connect to the Dealer's Queue Manager in MQ	

runRegApp.sh	This is the bash ‘wrapper’ script that launches the Regulator App ; it also sets the MQ connection information, so the application knows how to connect to the Regulator’s Queue Manager in MQ	