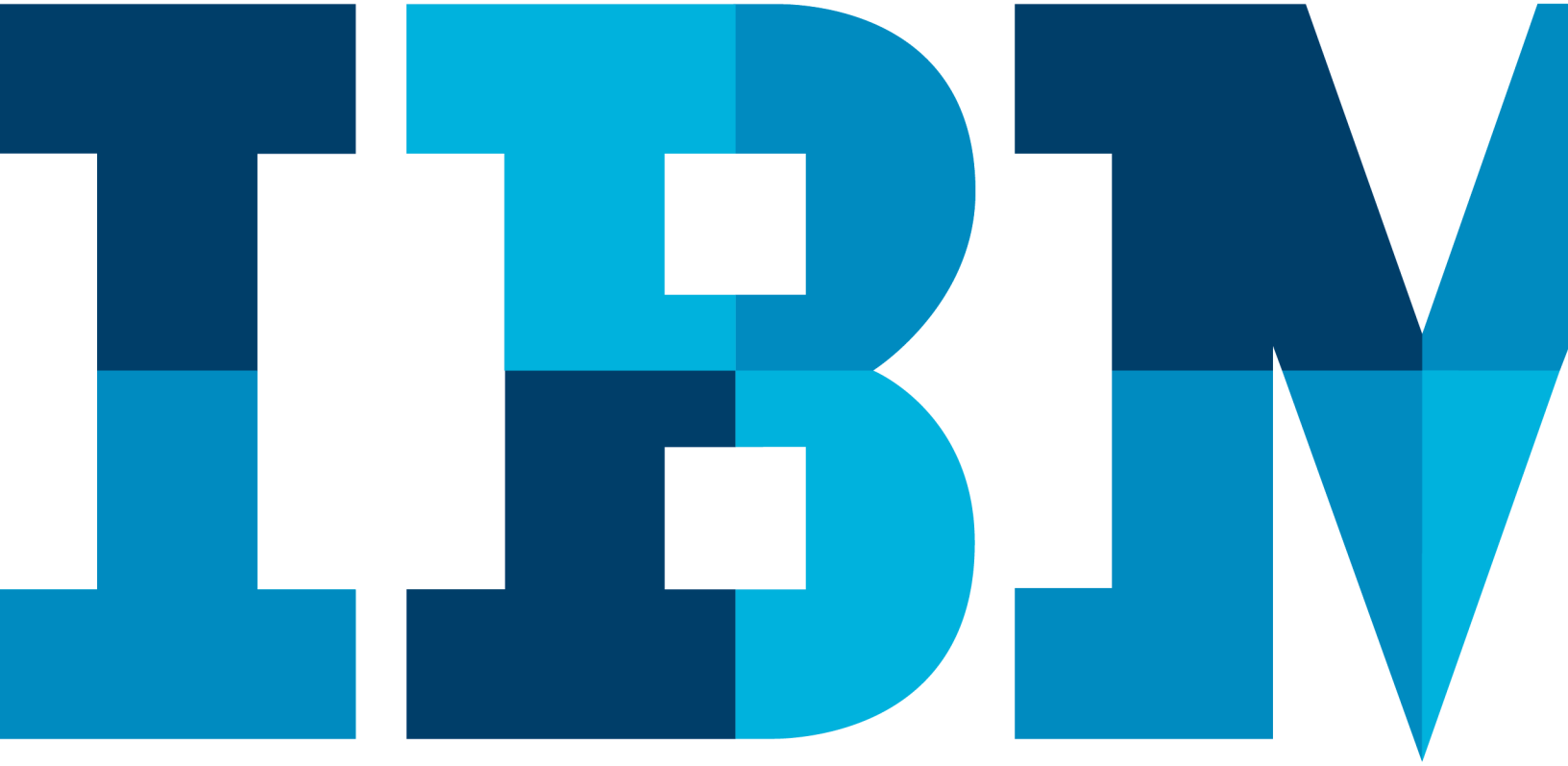
IBM Blockchain Hands-On Lab



IBM

Contents

OVERVIEW INTRODUCTION TO THE LAB 3

SECTION 1. DEPLOYING THE SAMPLE APPLICATION 4

1.1. CREATING A BLOCKCHAIN SERVICE 4
1.2. INITIALIZING THE ASSET TRANSFER DEMO 9

SECTION 2. ASSET TRANSFER AND DISPOSAL SCENARIOS 11

2.1. STARTING THE ASSET TRANSFER DEMO 11
2.2. TRANSFERRING AN ASSET TO A DEALERSHIP 12
2.2.1. VIEWING THE DEALERSHIP'S ASSETS 12
2.2.2. TRANSFERRING THE ASSET 13
2.2.3. VIEWING THE UPDATED SET OF MANUFACTURER'S ASSETS 17
2.3. TRANSFERRING AN ASSET TO A LEASING COMPANY 18
2.4. TRANSFERRING AN ASSET TO A LEASEE 20
2.5. TRANSFERRING AN ASSET TO A SCRAP MERCHANT 21
2.6. DISPOSING OF AN ASSET 22
2.7. VIEWING TRANSACTIONS 23
2.7.1. THE REGULATOR 23
2.7.2. OTHER USERS 24

SECTION 3. VIEWING THE BLOCKCHAIN 26

SECTION 4. MANAGING THE SAMPLE APPLICATION 28

4.1. VIEWING THE COMPONENTS OF THE BLOCKCHAIN SERVICE 28
4.2. VIEWING THE BLOCKCHAIN 29
4.3. UNDERSTANDING THE BLOCKCHAIN PEERS 32
4.4. INTERACTING WITH THE PEERS 33
4.5. VIEWING THE SERVICE STATUS, SUPPORT CONTACTS AND SAMPLES 39

SECTION 5. SECURITY FUNDAMENTALS 40

5.1. TRANSACTION SIGNING 40
5.2. MEMBERSHIP SERVICES CONCEPTS 43
5.3. USING MEMBERSHIP SERVICES 44

SECTION 6. REMOVING THE SAMPLE APPLICATION 47

APPENDIX A. NOTICE 48

APPENDIX B. TRADEMARKS AND COPYRIGHTS 50

Overview Introduction to the Lab

The purpose of this lab is to introduce you to the concepts of a blockchain by showing you how a blockchain transfers assets between participants in a business network. We will use car leasing as the scenario for the demo.

The lab runs inside the IBM® Bluemix environment; however, for this lab we will ignore Bluemix and focus on the car leasing demo itself. There is a follow-up lab that will properly introduce you to the Bluemix environment, and allows you to create and monitor the Blockchain service and application.

For Hyperledger Fabric V0.6 in Bluemix

Section 1. Deploying the Sample Application

In this section, we will log onto Bluemix and initiate the car leasing demo application.

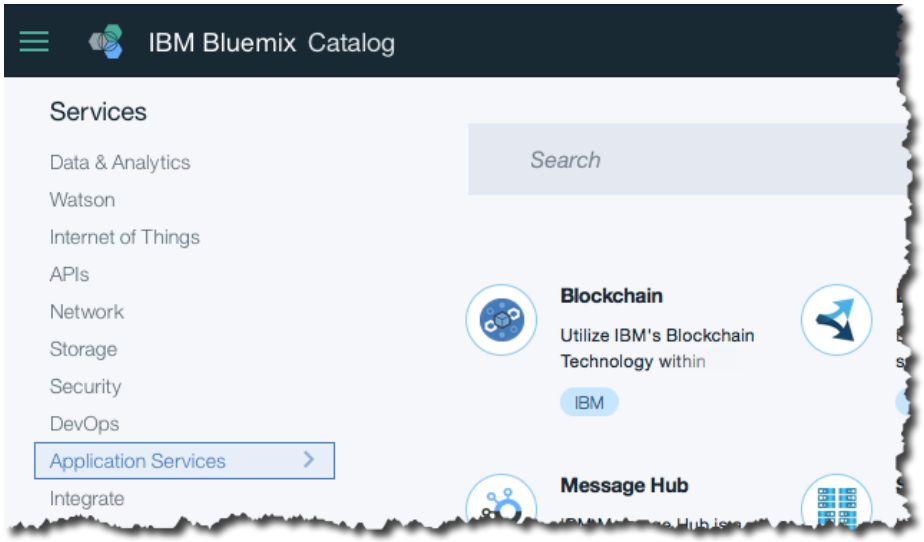
1.1. Creating a Blockchain Service

__1. Open a web browser (Firefox or Chrome are recommended) and go to www.bluemix.net.

__2. Click or '**Log In**' or log into an existing Bluemix account or '**Create a free account**' to create a new account.

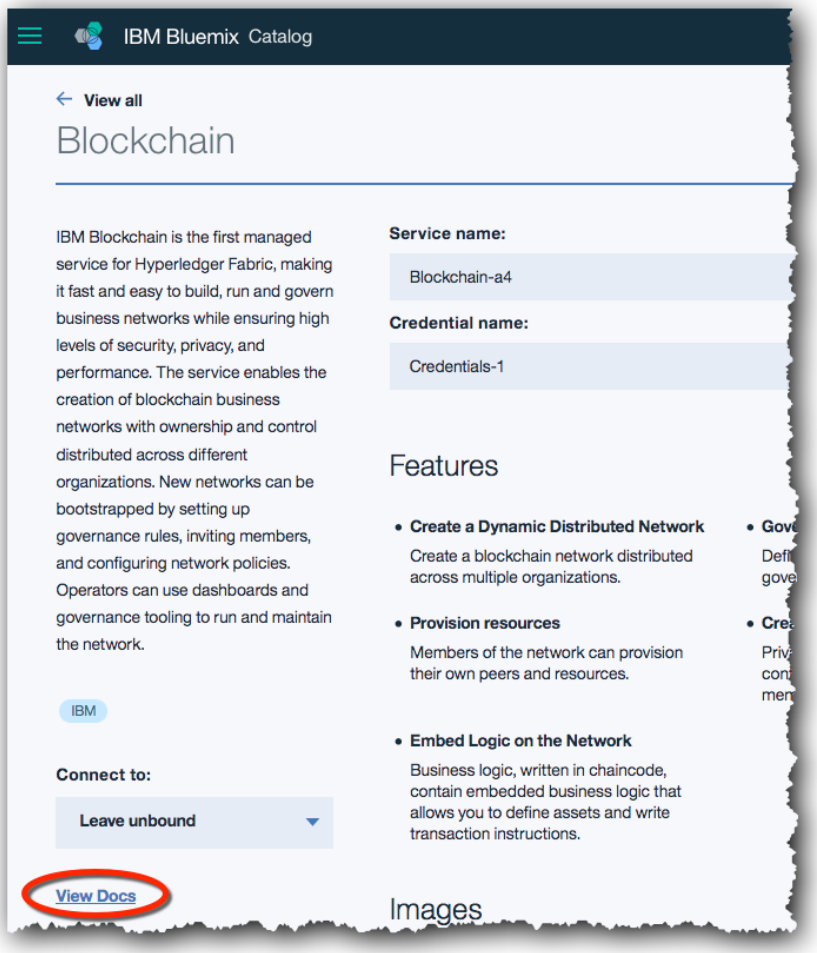
__3. Once you have successfully signed up and logged into Bluemix, select **Catalog** from the top bar.

__4. In the '**Services**' section of the sidebar, click '**Application Services**' and select **Blockchain**.



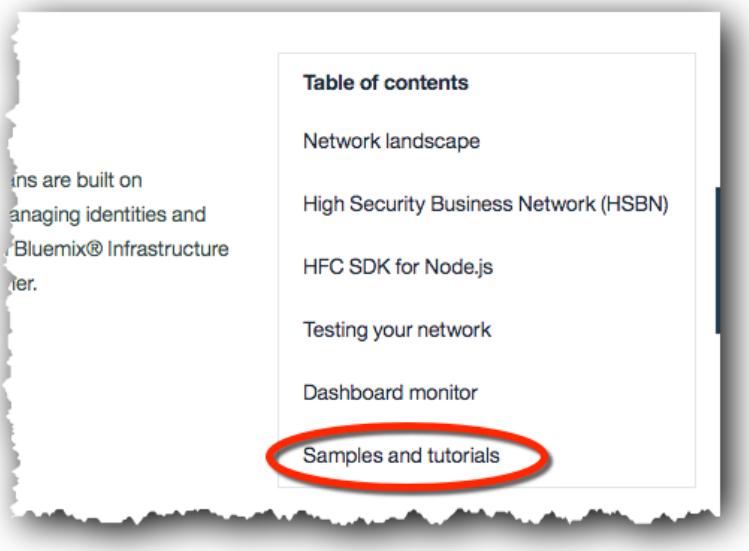
__5. Review the service description and information about the service.

__6. Click '**View Docs**' and learn about the process of creating a blockchain environment.



__7. Click '**HSBN and Starter plans**' on the left of the page.

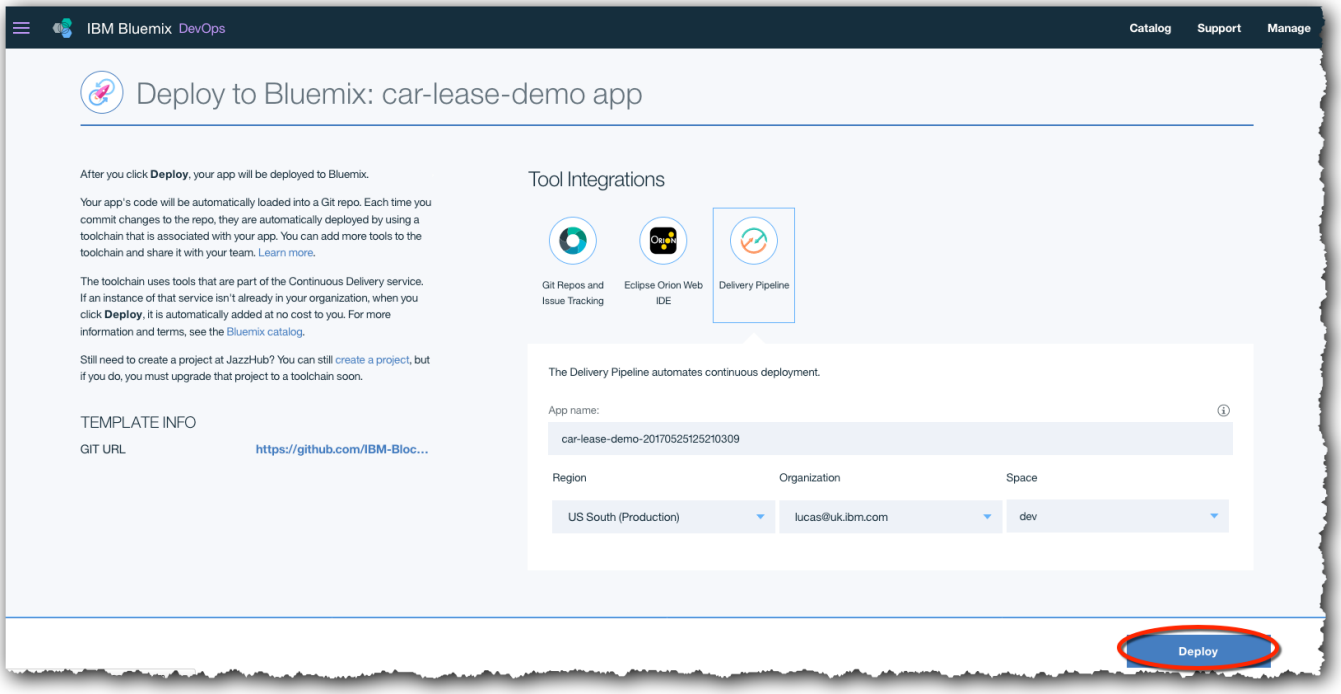
__8. On the right hand side of the page that appears, click '**Samples and Tutorials**'. (You may have to increase the size of your browser window to see the right hand table of links.)



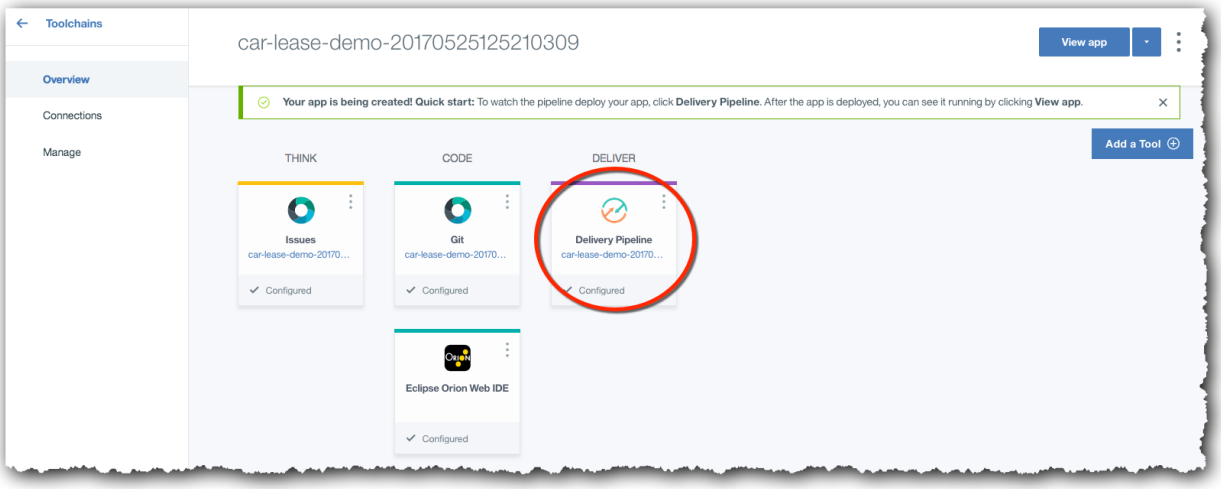
__9. Click 

against the Car Lease demo.

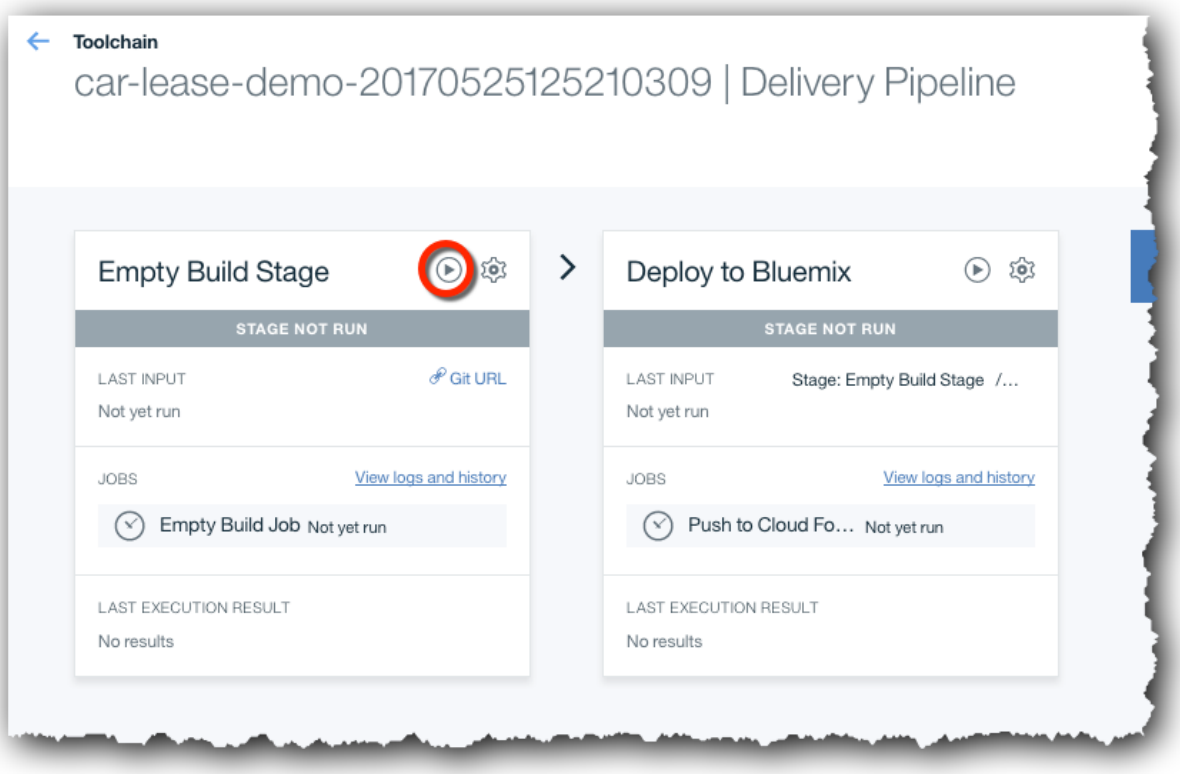
__10. Click '**Deploy**'.



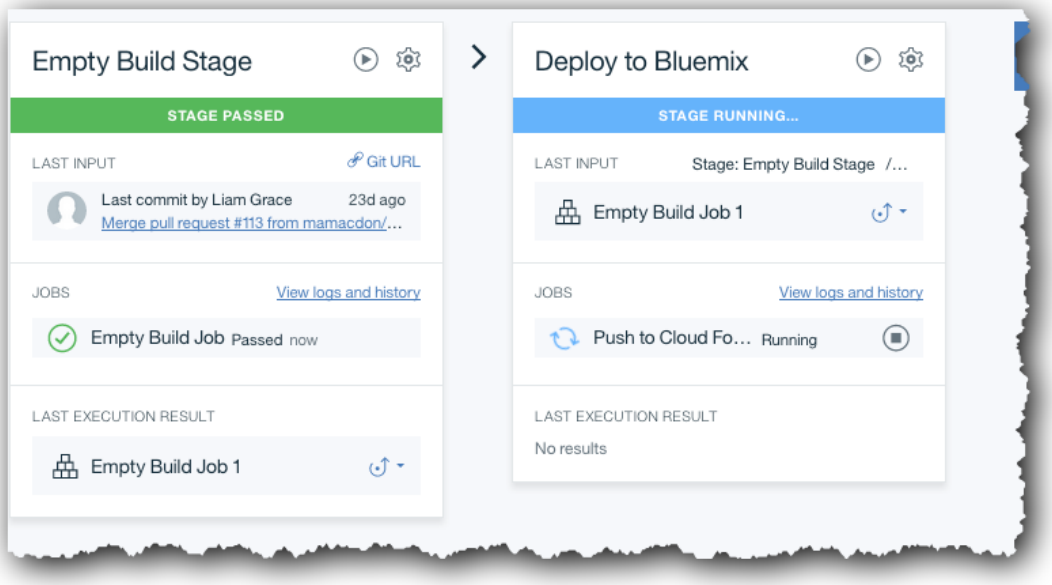
__11. Click '**Delivery Pipeline**'.



__12. Click the  button against '**Empty Build Stage**' to deploy the application to Bluemix.

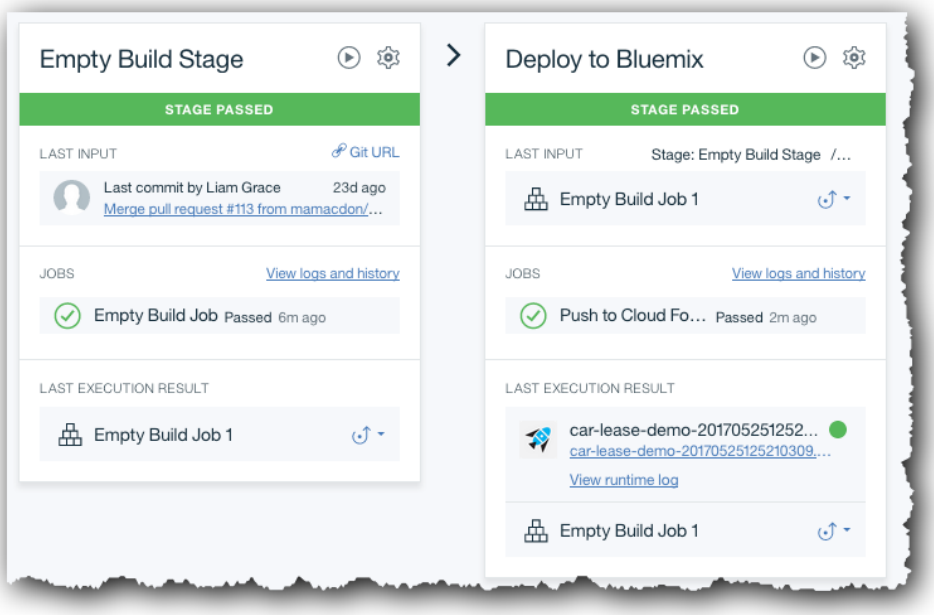


__13. Wait while the application is deployed. This can take around five minutes.

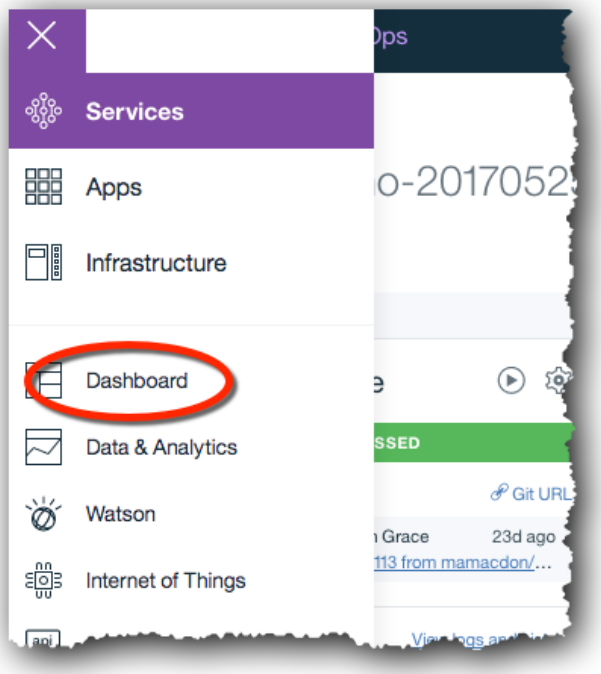


The source code is cloned from Github, built and pushed to the Cloud Foundry component of Bluemix.

These tasks are complete when the Build and Deploy stages have both passed.



__14. Once the application has been successfully deployed, click the  icon on the top left of the page to display the Bluemix menu and '**Dashboard**' to display the dashboard.

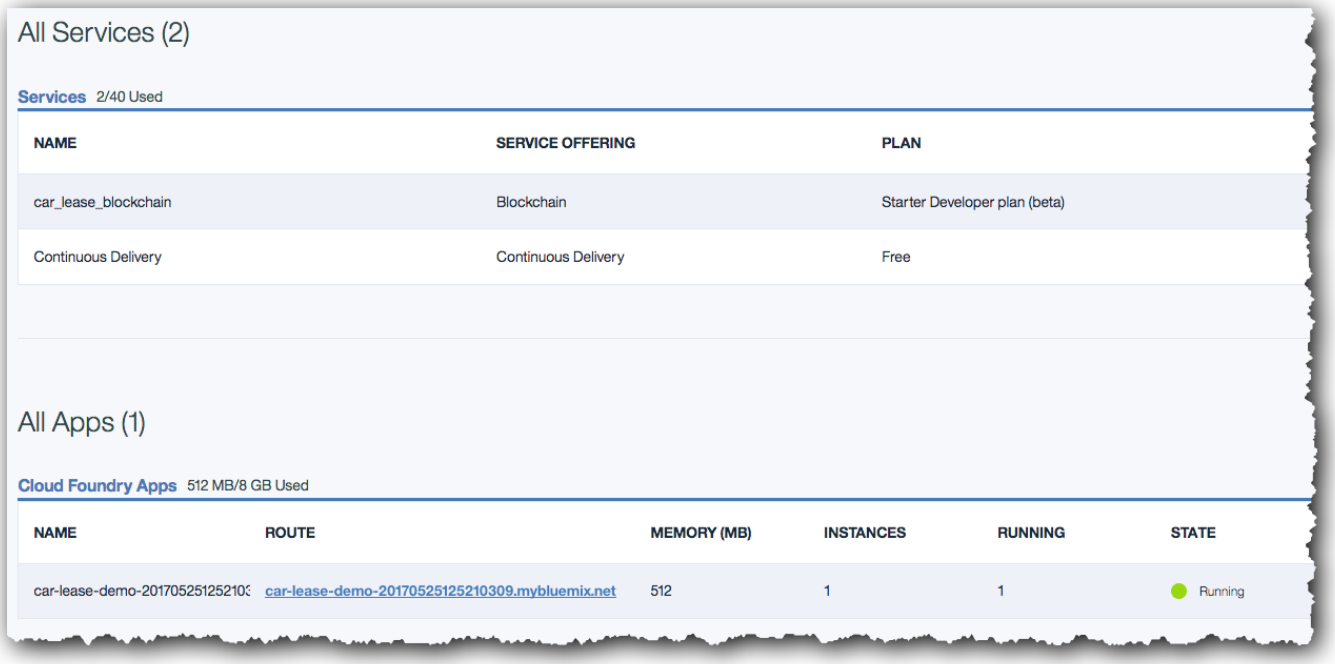


__15. Review the services and applications that have now been created and initialised.

You should see:

- a continuous delivery service, which can build and deploy and changes made to your copy of the car lease demo
- a blockchain service, which is running the instance of Hyperledger Fabric

- the Car Lease demo application, which should be running.

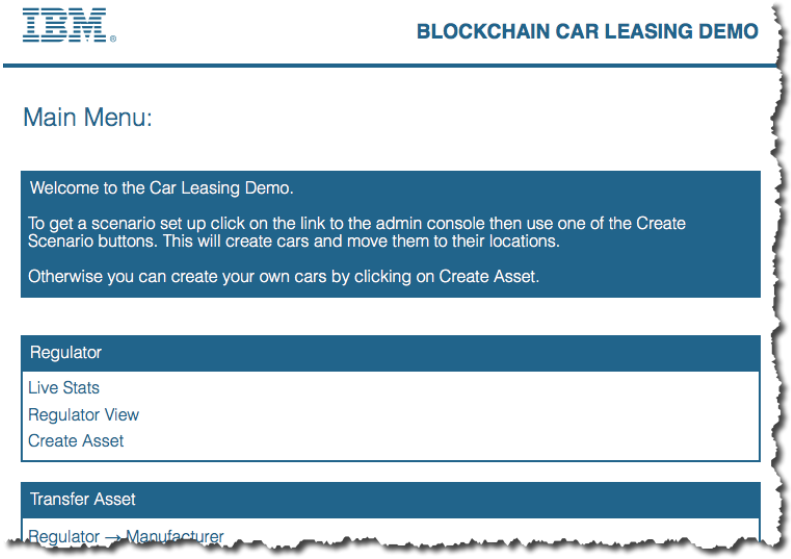


1.2. Initializing the Asset Transfer Demo

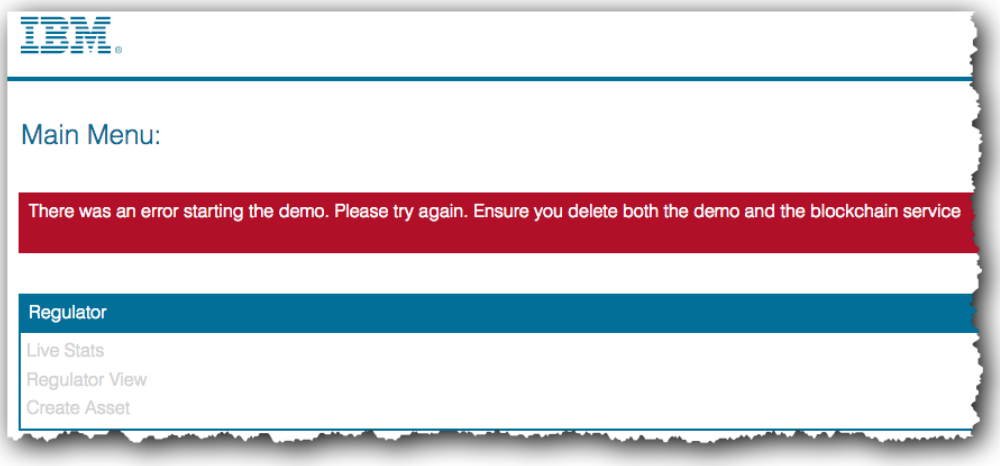
__16. Click the blue hyperlink under the 'ROUTE' column of your application (which will be of the form 'car-lease-demo-2017MMDDhhmmssxxx.mybluemix.net') and this will load the demo webpage.

(Do not click elsewhere on this line, as this will load the administration interface for the application, which we will look at later).

You will now see the front page of the car leasing demo.



If you see the following error, first try reloading the web page. If the problem persists please contact your instructor.

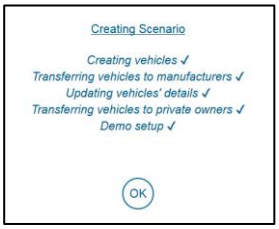


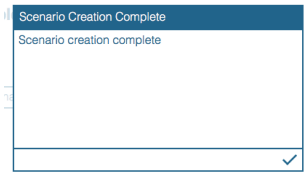
__17. From the Car Leasing demo front page, click '**Admin Console**' and '**Create Simple Scenario**'.

This will preload the blockchain with a set of transactions. (The Full Scenario works fine too; the difference between the Simple Scenario and the Full Scenario is that in the Full Scenario more assets are initially loaded onto the blockchain; this takes a few minutes longer to initialize, however.)

Wait for the initialization to complete.

__18. Click '**OK**' to close the Creating Scenario log, and then dismiss the 'Scenario Creation complete' box by clicking the check mark.





__19. Finally click '**Home**' to return to the main menu.

Section 2. Asset Transfer and Disposal Scenarios

In the following sections, we will discover how blockchain is used to track ownership of an asset across multiple participants in a business network. The scenario describes how blockchain is used to model the lifecycle of vehicle ownership and control between the following participants:

1) Manufacturer to Dealership
2) Dealership to Leasing Company
3) Leasing Company to Leasee
4) Leasing Company to Scrap Merchant

The Scrap Merchant's role in this scenario will also demonstrate how asset disposal can be represented on the blockchain.

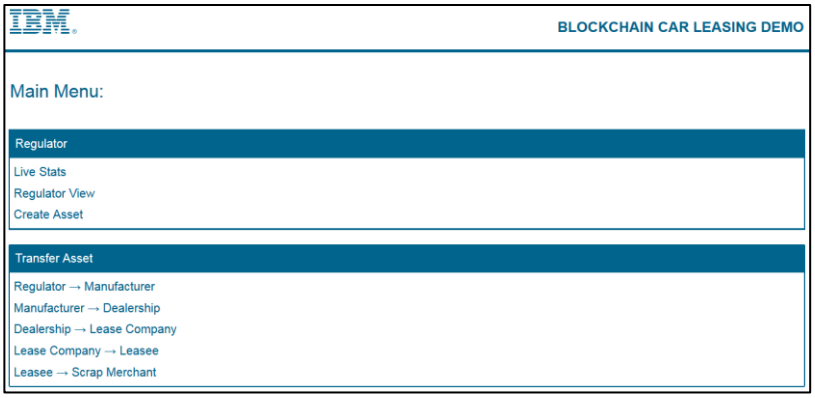
In this scenario each participant has entered into a business agreement with each other and all parties are known and trusted by each other. The above process of transferring vehicles has been negotiated and agreed with all participants. The order in which the above processes take place is strictly defined within the demo showing that for example a Manufacturer cannot transfer directly to a Leasee by missing out the dealership and Leasing company transfers.

These rules have been defined in the smart contract which has been written and signed by the regulator (the DVLA).

2.1. Starting the Asset Transfer Demo

__20. Bring up a web browser (Firefox or Chrome are recommended) and go to the URL that your instructor has provided. If you completed Section 1, just use the URL route of the application that you already created.

You should be able to see the Car Leasing main menu.



2.2. Transferring an Asset to a Dealership

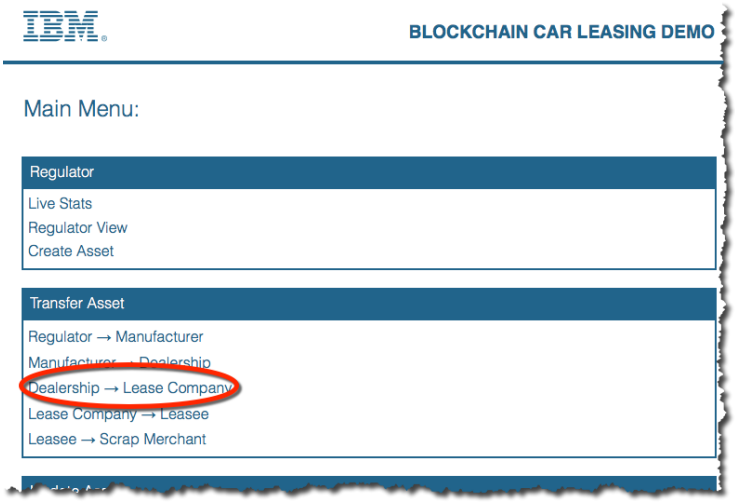
In the following section we will transfer the ownership of a vehicle from a dealership to a leasing company (known as “Beechvale Group”) using the blockchain.

Before transferring the vehicle to the dealership we will verify which assets the target dealership currently owns.

2.2.1. Viewing the Dealership’s Assets

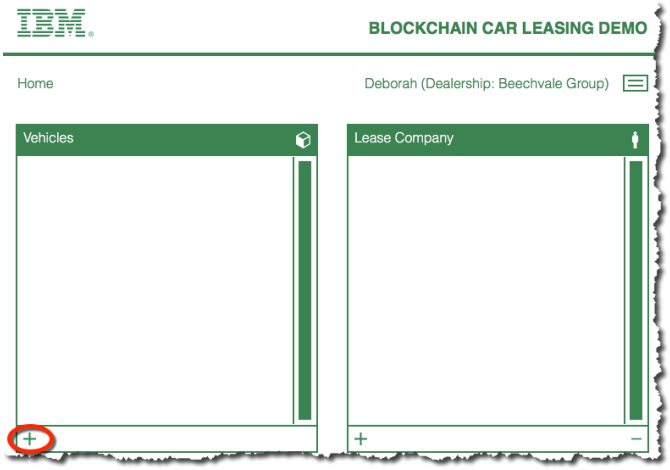
In this section, we will first act as a dealership to verify which assets the Beechvale dealership owns.

__21. From the Main Menu, click ‘**Dealership -> Lease Company**’.

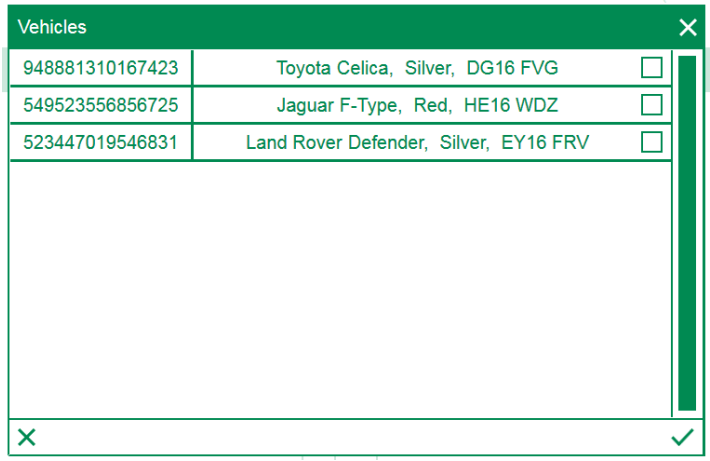


We now see the application from the point of view of Deborah, who works for the Beechvale Group (a dealership).

__22. Click the plus sign in the “**Vehicles**” window to list the vehicles that are owned by this dealership according to the blockchain.



You should see a number of vehicles displayed. (There might be more or fewer depending on the scenario that has been set up.)



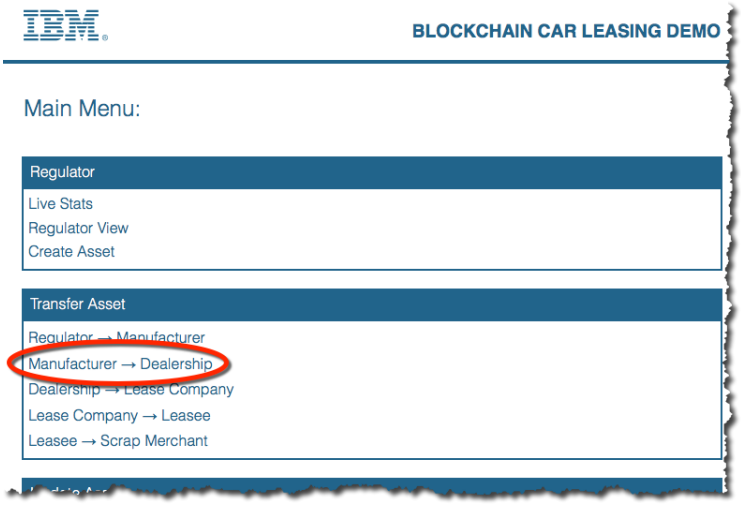
	If you see no cars at all, this might be due to a timing issue in the lab environment. Try waiting a few seconds and try the previous step again. If the problem persists, ask the instructor.

__23. Click the '**X**' to dismiss the window and 'Home' to return to the main menu.

2.2.2. Transferring the Asset

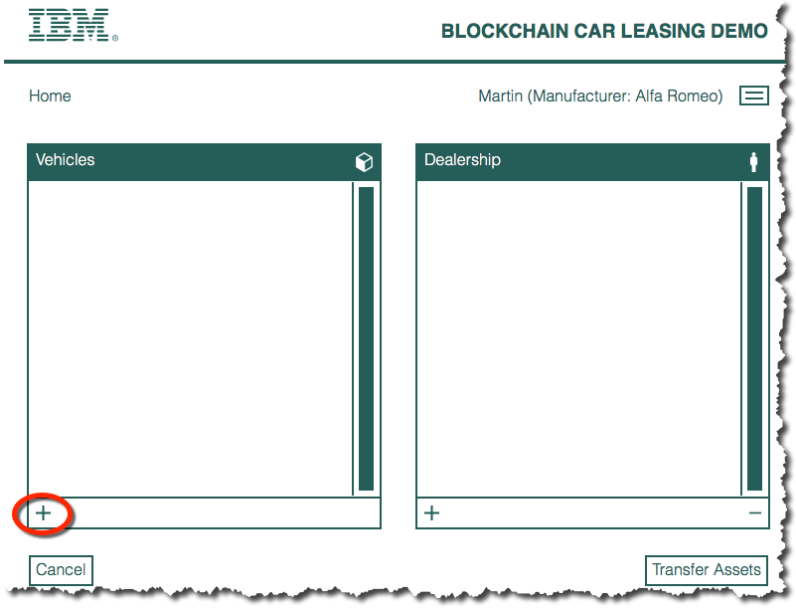
We will now transfer an Alfa Romeo car to the Beechvale Dealership from Alfa Romeo.

__24. From the demo main menu, click the '**Manufacturer -> Dealership**' link in the Transfer Asset section.



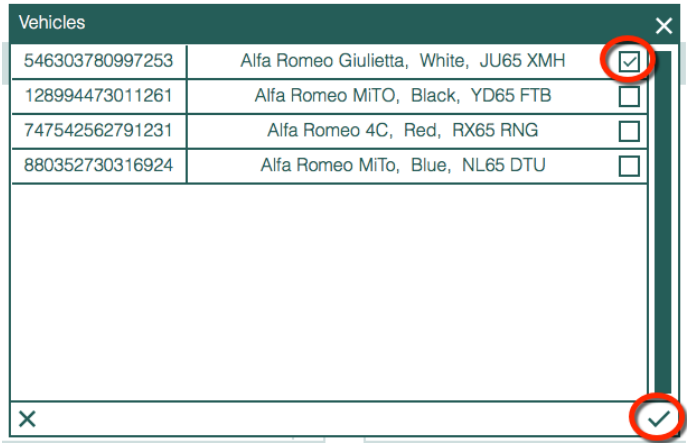
You are now viewing the application as Martin, who works for Alfa Romeo.

__25. Click the '**+**' sign in the vehicles box.



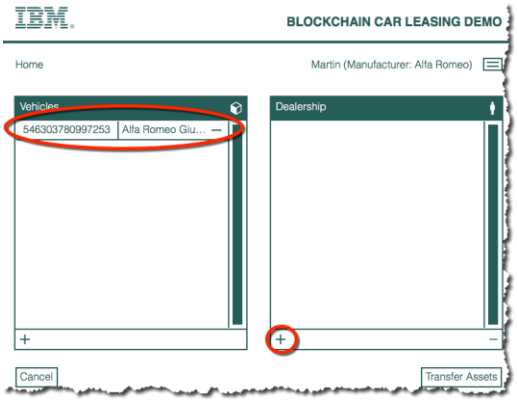
This queries the blockchain for the cars that are owned by Alfa Romeo.

__26. Click the checkbox against the first car to add it to the transfer request, then click the checkmark (tick) to save the choice.

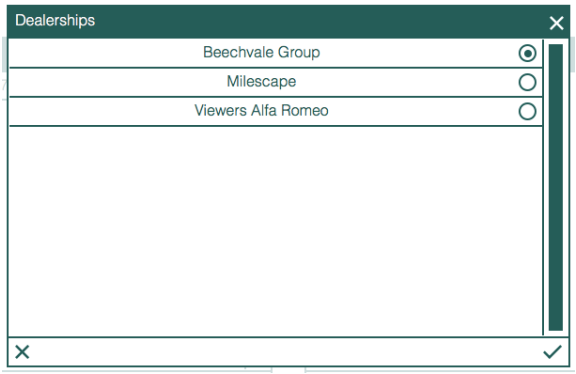


The Alfa Romeo you selected now appears in the list of vehicles to be transferred.

__27. Click the '**+**' sign in the Dealership box.



__28. From the list of Dealerships, choose '**Beechvale Group**' then, click the checkmark to confirm your choice):

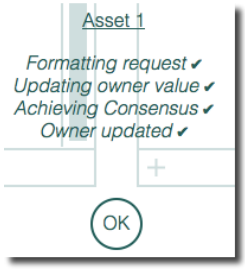


__29. Click '**Transfer Assets**'.

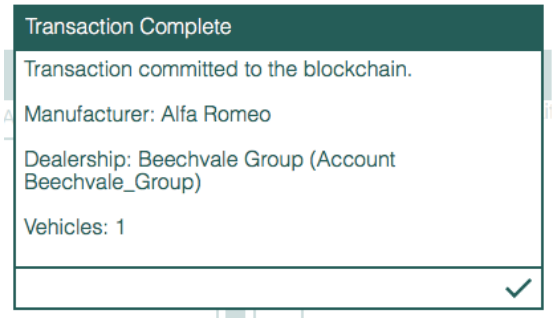
This adds a transaction to the blockchain that will transfer ownership of the Alfa Romeo car to the Beechvale Group.

The nodes in the blockchain network will now confirm the transaction; this takes a few seconds to complete.

__30. Click **OK** when the transaction has been validated by the blockchain network.



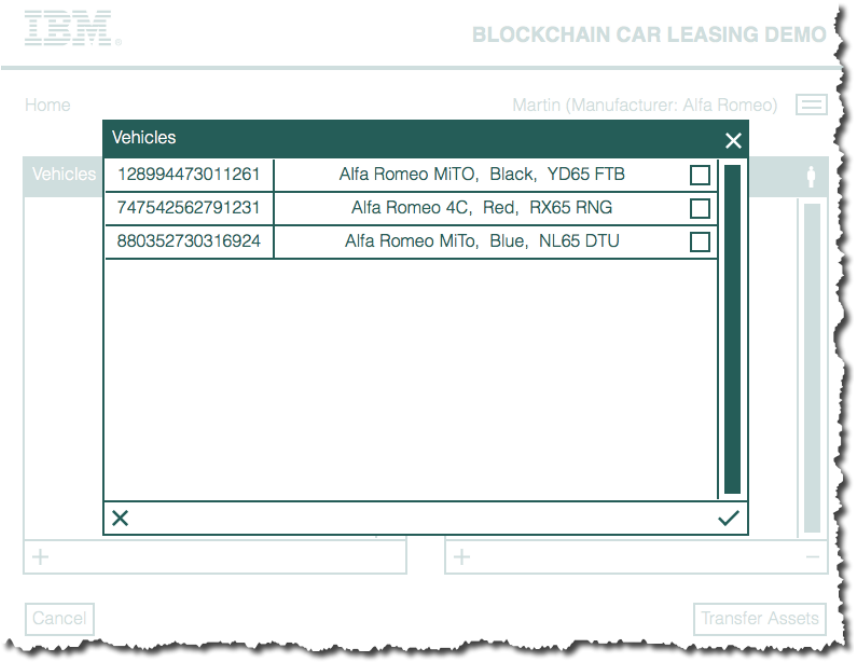
__31. Dismiss the transaction confirmation message.


Transaction Complete
Transaction committed to the blockchain.
Manufacturer: Alfa Romeo
Dealership: Beechvale Group (Account Beechvale_Group)
Vehicles: 1

2.2.3. Viewing the Updated Set of Manufacturer's Assets

The manufacturer's ability to control the asset has now been removed.

__32. Click the '**+**' sign on the Vehicles box to verify that the manufacturer can no longer see the asset you transferred:



The manufacturer now controls one asset fewer; the transferred vehicle is no longer visible to the manufacturer.

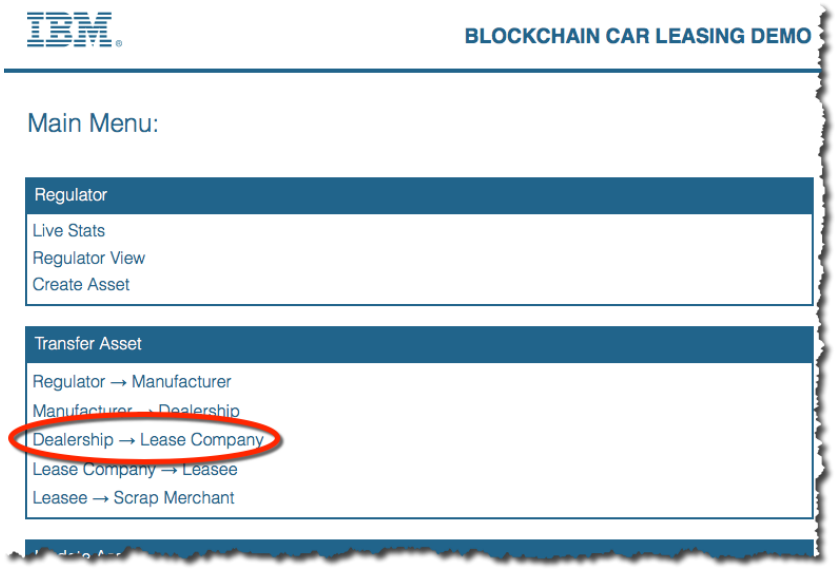
__33. Click the '**X**' to dismiss the window.

2.3. Transferring an Asset to a Leasing Company

In this section we will act as Deborah, who works for the Beechvale Group dealer. First we will verify that the asset you transferred earlier is now available to you to transfer; you will then transfer the asset to a leasing company.

In the previous section we transferred the ownership of a vehicle from the Alfa Romeo manufacturer to the dealership “Beechvale Group”. The vehicle will now appear in the list of vehicles Beechvale Group are able to control.

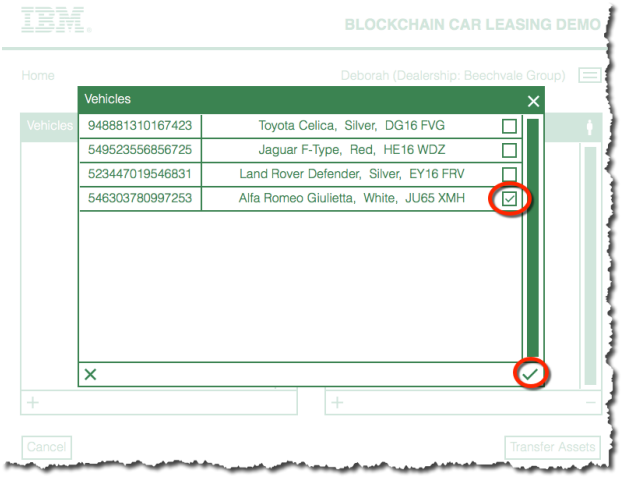
__34. From the main menu, click ‘**Dealership -> Lease Company**’.



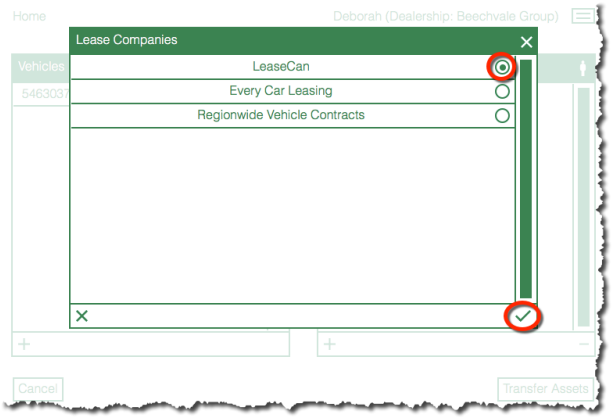
You are now experiencing the application as Deborah again.

__35. Click the ‘**+**’ icon in the “Vehicles” box to show the list of vehicles that the dealer can see.

__36. Select the Alfa Romeo car and click the check mark (tick).

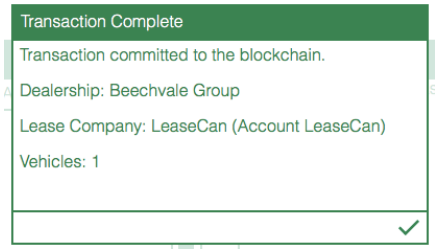


__37. In the Lease Company window click the plus sign to select ‘LeaseCan’. Click the check mark to confirm.

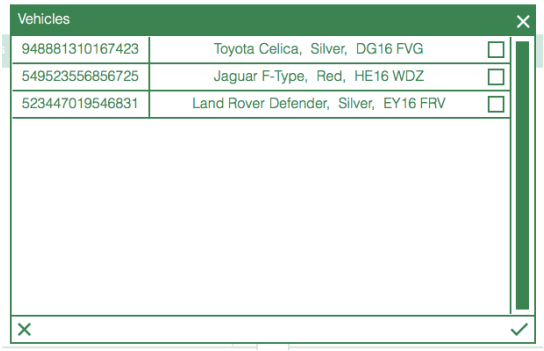


__38. Click the Transfer Assets button and wait for the transaction to be validated.

__39. Click **OK** and then dismiss the Transaction complete window.



__40. Click the ‘**+**’ icon in the “Vehicles” box to verify that Deborah no longer has visibility of the car she just transferred. Click X to close the window.



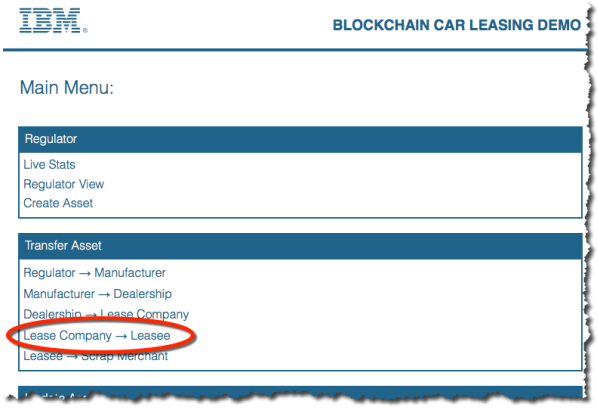
__41. Return to the main menu.

2.4. Transferring an Asset to a Leasee

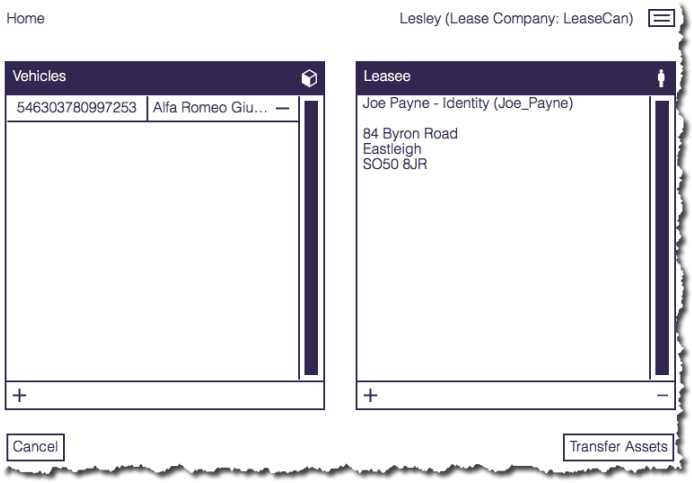
In this section, we will act as a representative of the lease company. First, we will verify that the asset you transferred earlier is now available to you acting as the lease company to transfer; we will then transfer the asset to a leasee.

In the previous section, we transferred the ownership of a vehicle from the dealership “Beechvale Group” to the lease company “LeaseCan”. The vehicle will now appear in the list of vehicles LeaseCan is able to control.

__42. From the main menu, click ‘**Lease Company -> Leasee**’.



__43. Use the two panels to prepare a transfer of the Alfa Romeo car to Joe Payne.



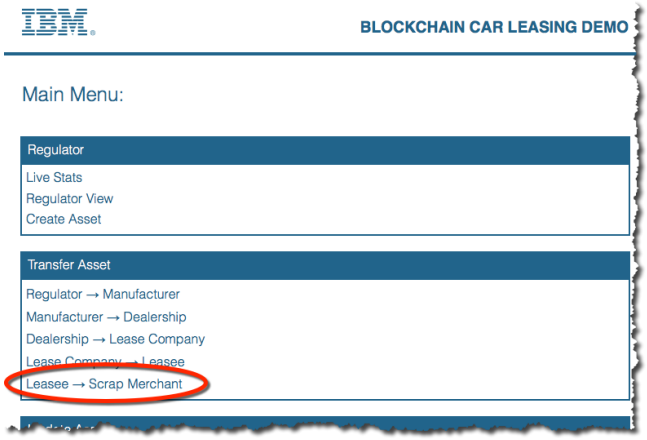
__44. Click the Transfer Assets button and wait for the transaction to be validated. Dismiss the confirmation prompts.

2.5. Transferring an Asset to a Scrap Merchant

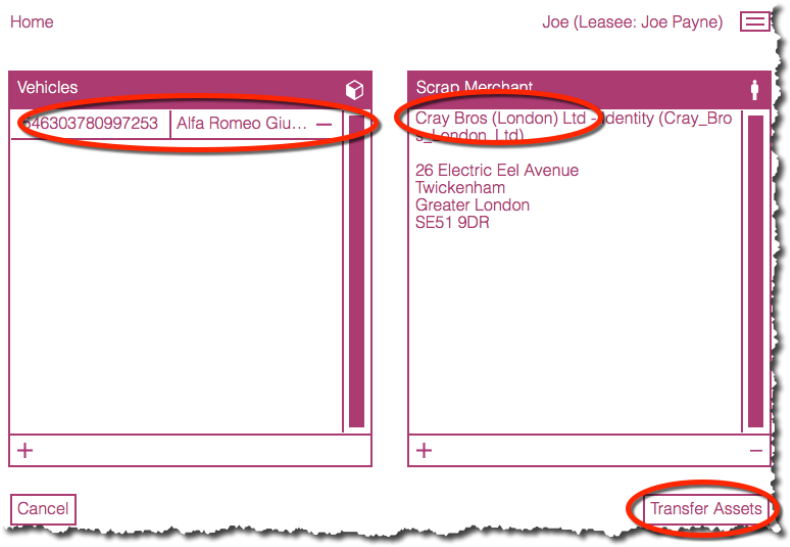
In this section we will act as the leasee, who in this greatly simplified scenario has the authority to send the vehicle to a scrap merchant. First, we will verify that the asset you transferred earlier is now available to us acting as the leasee; we will then transfer the asset to a scrap merchant.

In the previous section, we transferred the ownership of a vehicle from the lease company “LeaseCan” to Joe Payne. The vehicle will now appear in the list of vehicles Joe is able to control.

__45. From the main menu, click ‘**Leasee -> Scrap Merchant**’.



__46. Transfer the car to the Cray Bros (London) Ltd.



__47. When the transaction has been validated, return to the main menu.

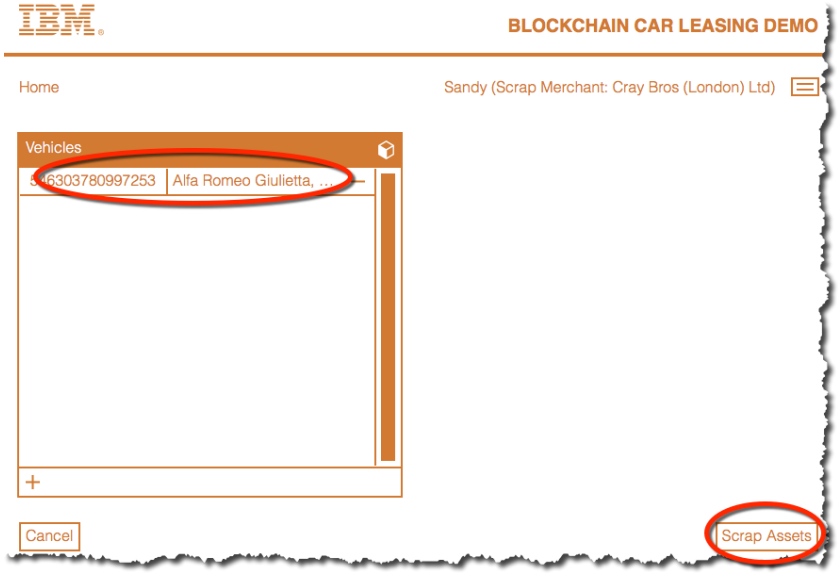
2.6. Disposing of an Asset

In this section, we will act as the scrap merchant and dispose of the asset. First, we will verify that the asset you transferred earlier is now available to you acting as the scrap merchant. We will then dispose of the asset.

In the previous section, we transferred the ownership of an Alfa Romeo car from “Joe Payne” to the scrap merchant. The vehicle will now appear in the list of vehicles that the scrap merchant is able to control.

__48. From the main menu, click ‘**Scrap Merchant -> Scrap**’.

__49. Use the ‘**+**’ sign to prepare the asset for scrapping and click ‘**Scrap Assets**’ when ready. Note that there is no destination panel for this operation.



__50. Dismiss the confirmation dialogs once the asset has been scrapped.

__51. Verify that the asset can no longer be viewed by the scrap merchant.

__52. Return to the main menu.

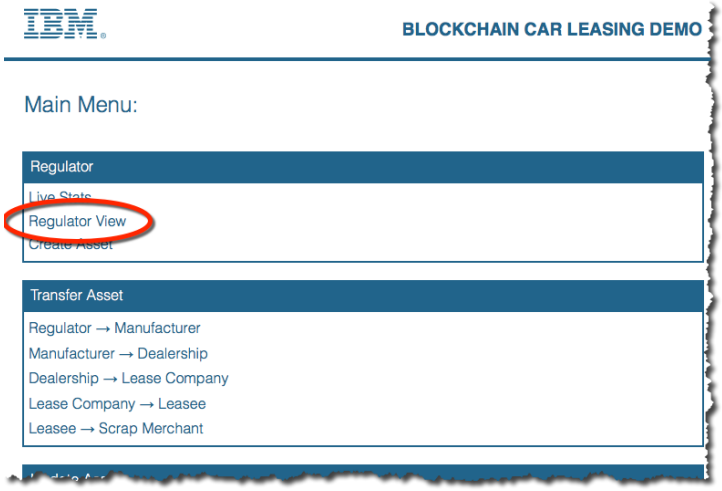
Once the asset has been transferred it is not removed from the blockchain; it has merely been marked as “scrapped”. In the next section we will demonstrate that the asset can still be viewed in the transaction logs.

2.7. Viewing Transactions

The regulator view has unrestricted access to all activities on the Blockchain. In this section we will act as the Regulator and view all asset transfer and disposal activity. We will then act as another user who has a more restricted view of the transactions.

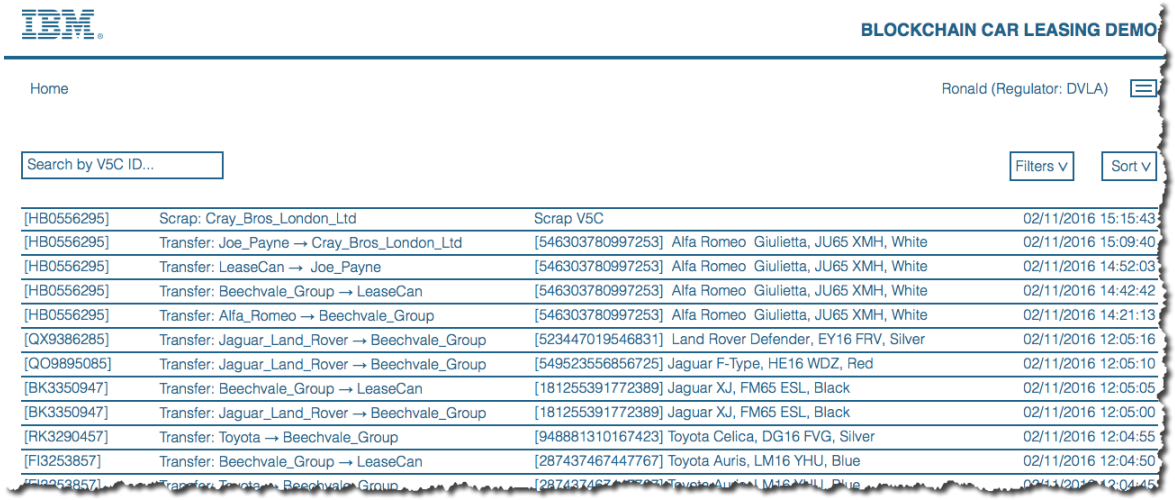
2.7.1. The Regulator

__53. From the main menu, click '**Regulator View**'.



__54. From the main menu, click '**Regulator View'**.

You will see the activity in chronological order, with the most recent activity at the top of the list of transactions.



__55. In the "Search by V5C ID…" box, start typing the vehicle identifier of the Alfa Romeo you have been working with. In the example here this is **HB0556295** but your ID might be different.

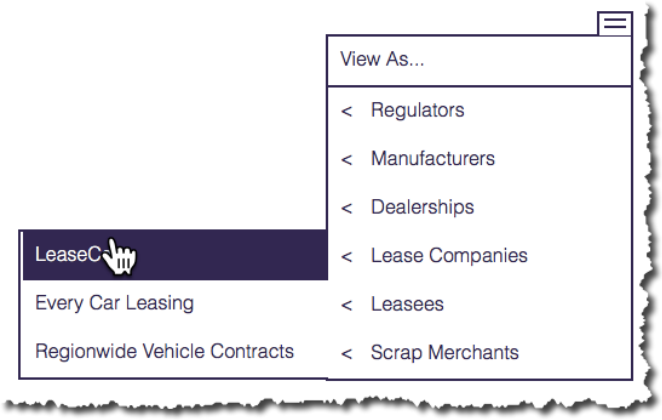
This will filter the view so that only the transactions for this car are shown.

__56. View the complete set of transactions again by using the backspace key to delete the characters you just entered.

2.7.2. Other users

Other users can only see part of the lifecycle of the vehicle. They are able to see what happened to the vehicle prior to their ownership and whilst they owned it but cannot see what happened to the vehicle after they transferred it.

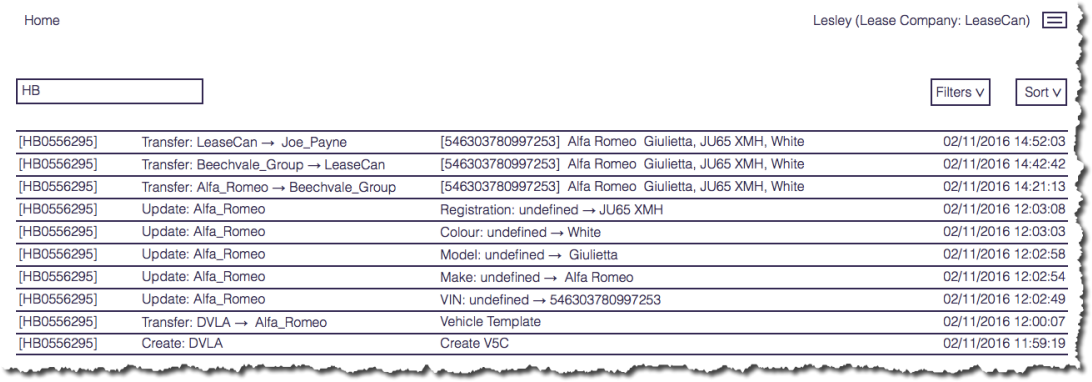
__57. Click the three lines in the top right corner of the Regulator view to see the set of transactions through the eyes of another user. In the dropdown that appears hover over “Lease Companies” then click ’**Lease Can**’.



The view now changes to show all transactions that:
(a) relate to cars currently owned by LeaseCan, or
(b) relate to cars once owned by LeaseCan, up to the point that they were transferred away.

__58. Start typing the identifier of the Alfa Romeo once more (**HB0556295** in the example, but again your ID will vary).

Note how the transactions shown against this car are restricted to the ones up to the point that LeaseCan transferred the car to Joe Payne.

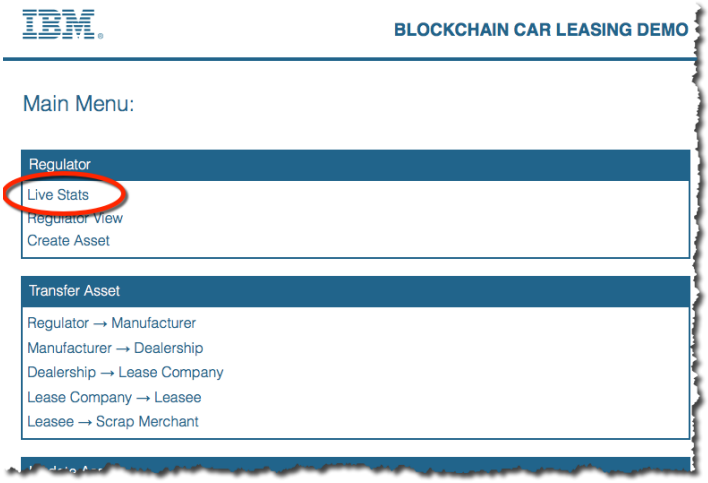


__59. Return to the main menu.

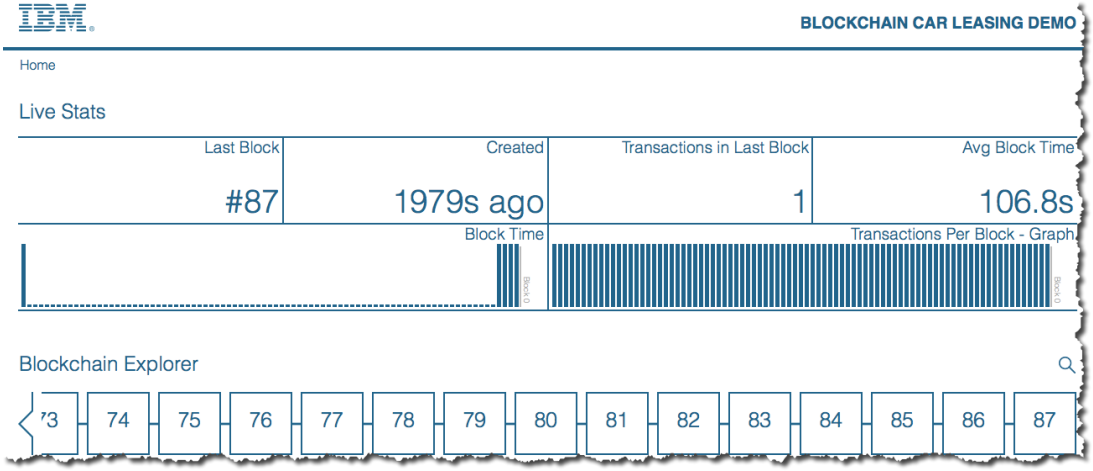
Section 3. Viewing the Blockchain

In this final section of the lab, we will introduce one of the key data structures that makes up the blockchain. The follow-on lab ("Blockchain Explored") will cover this topic in more detail.

__60. From the main menu, click '**Live Stats**'.



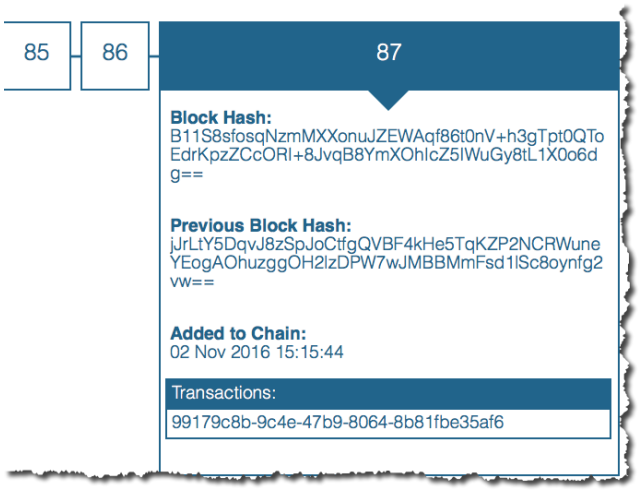
__61. Wait for the screen to be populated. (The blockchain is being queried, and this will take longer the more transactions there are.)



__62. Review the various pieces of information being shown.

Last Block:	The block number of the last committed block (higher numbers are more recent)
Created:	How long ago since the last block was committed
Transactions in Last Block:	The number of transactions in the last block; in this demo, this is usually one.
Avg Block Time:	The average time between each block being committed
Block Time:	A graph showing how much time was between each block
Transactions Per Block - Graph:	How many transactions were in each block (again, this is usually one)
Blockchain Explorer:	Allows you to look at a specific block's details in the blockchain.

__63. Click a block in the Blockchain Explorer pane to see more information about it.



We will look at the blocks in more detail in the follow-on lab: “Blockchain Explored”.

Section 4. Managing the sample application

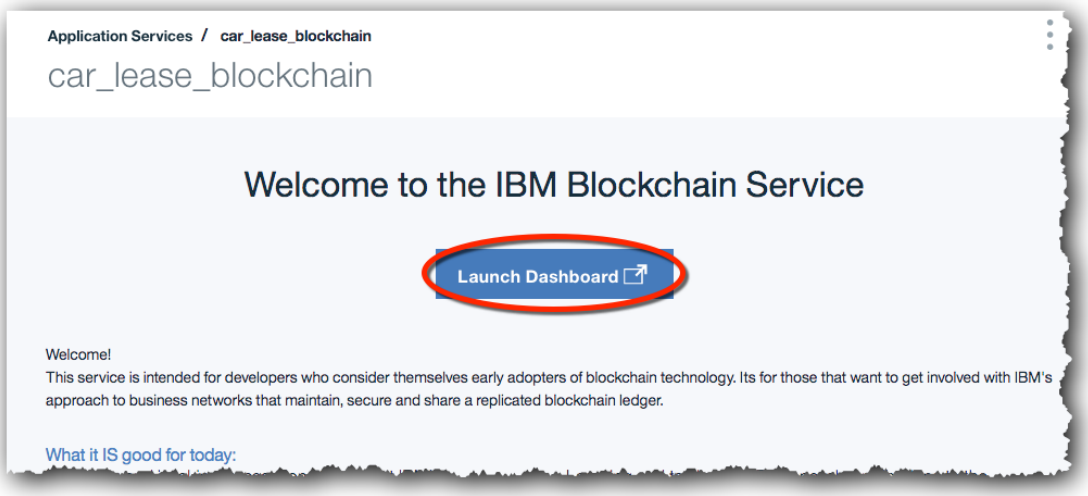
In this section we will use the monitoring tools available inside the Bluemix environment to view and manage the blockchain.

4.1. Viewing the components of the Blockchain service

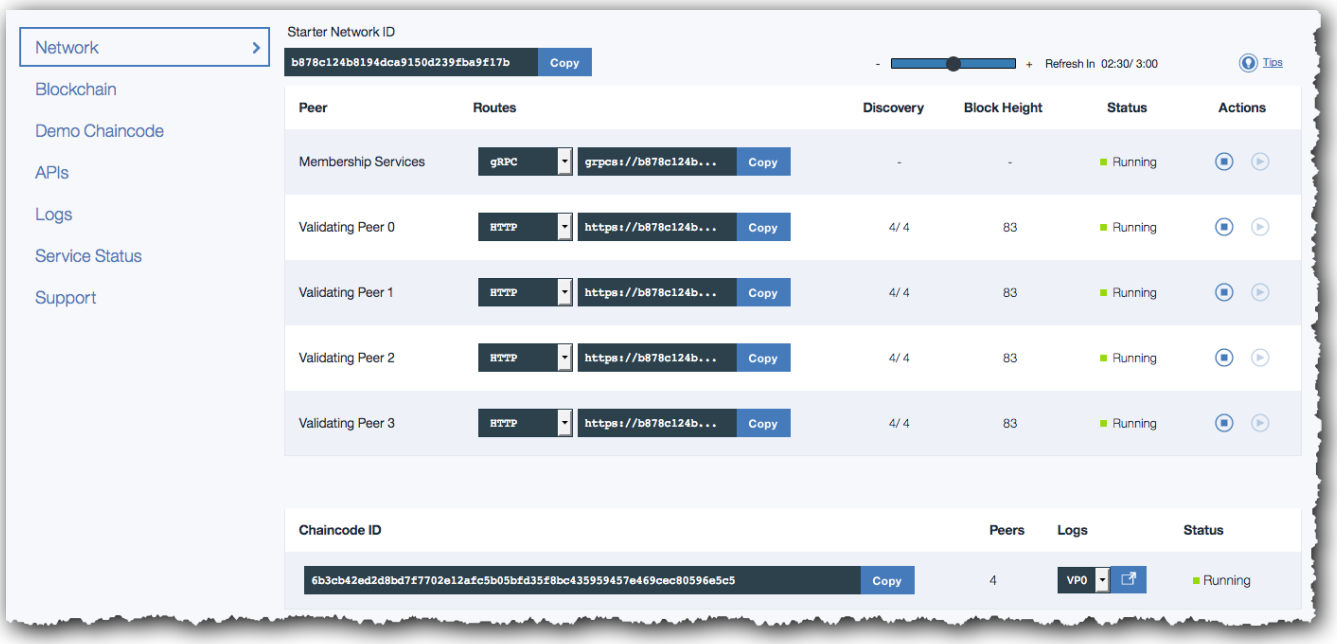
__64. Return to the Bluemix dashboard, either by selecting the icon in the top left of Bluemix and selecting '**Dashboard**' or by going directly to https://console.ng.bluemix.net/dashboard/applications.

__65. Click on the **car_lease_blockchain** service in the Services section of the dashboard.

__66. Review the details and select 'Launch Dashboard' to launch the dashboard.



You should now see the dashboard with seven tabs down the left-hand side. The '**Network**' tab will be selected by default.



The blockchain is a replicated, shared ledger. This blockchain is shared among all the participants of the network. Each participant still has their own copy of the ledger, and replication ensures that the copies are kept synchronised.

The blockchain network that has been set up for us in this demo contains four participants ("Validating Peers") as well as a Membership Services component that we will look at later. Applications submit transactions into just one validating peer, and peer-to-peer technology is used to replicate the transaction elsewhere.

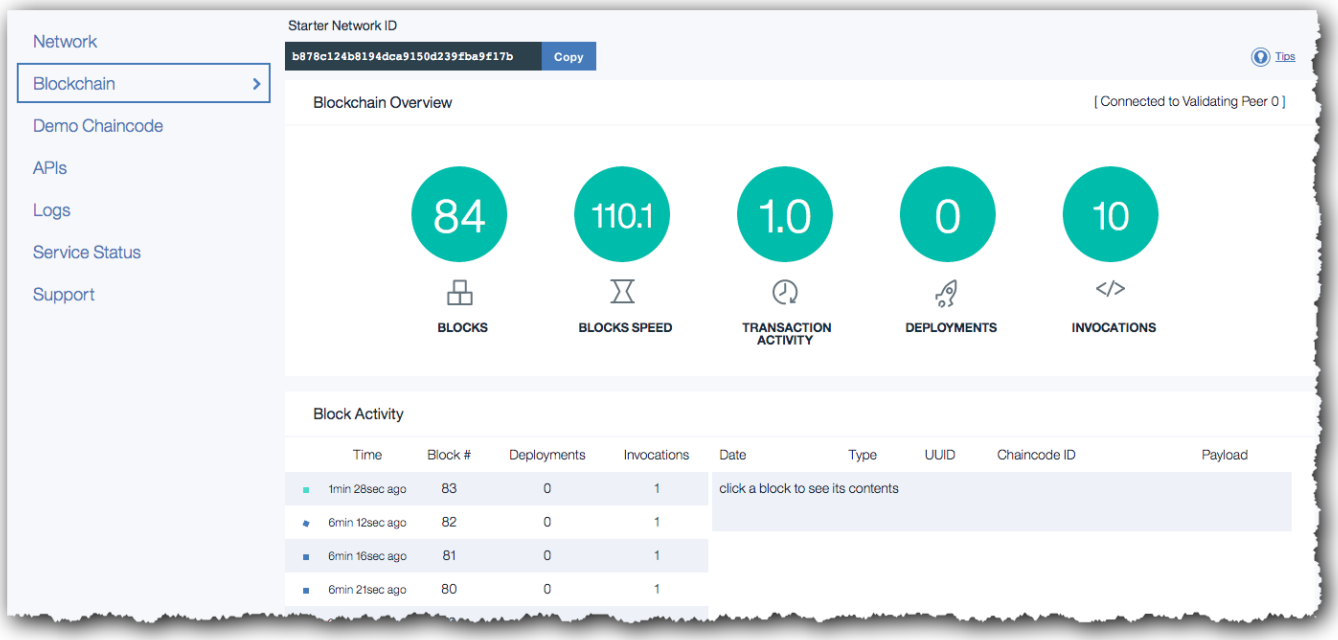
__67. Verify that the four validating peers each have the same block height.

__68. Return to the car leasing demo and invoke another transaction. Verify that the block height increases by one for all four validating peers. (Refresh the web page if necessary.)

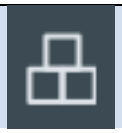
4.2. Viewing the Blockchain

The Blockchain tab shows a visual representation of the state of the Blockchain.

__69. Click the '**Blockchain**' tab at the left of the page.



The icons show:

	Total number of blocks in the chain
	Average number of blocks per hour
	Average number of transactions per block
	Number of deployment calls made to deploy chaincode
	Number of invoke requests made within this blockchain

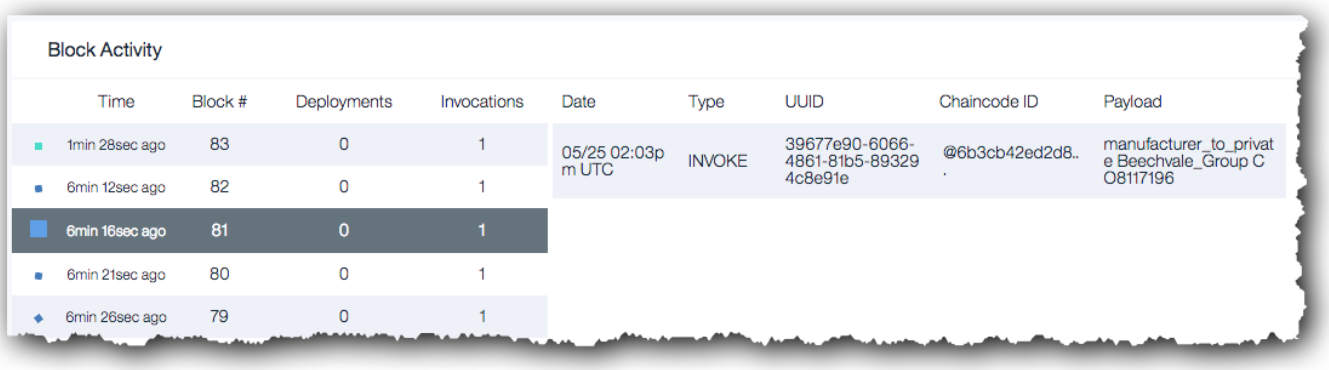
Each block contains a set of transactions. In Hyperledger Fabric V0.6, a transaction is the record of the request to interact with chaincode (a smart contract). The two most important transaction types are:

- *DEPLOY*: The request to deploy a piece of chaincode across all validating peers, so that it can be executed at a later date.
- *INVOKE*: The request to invoke a piece of chaincode (for example, invoke the chaincode to transfer the ownership of a car)

Other request types exist (e.g. query). Not all request types are recorded on the blockchain.

The blocks also include when that block was committed to the blockchain.

__70. Click on a block that contains at least one invocation request.



__71. Look through the list of transactions that are contained within the block.

Date	Type	UUID	Chaincode ID	Payload
05/25 02:03pm UTC	INVOKE	39677e90-6066-4861-81b5-893294c8e91e	@6b3cb42ed2d8...	manufacturer_to_private Beechvale_Group CO8117196

Each line of information is a transaction stored within the block. A block may contain multiple transactions but in this demo there will often only be one transaction per block due to the low frequency of transactions being made. The information displayed is:

Date	The date the transaction was submitted.
Type	The type of transaction taking place (e.g. INVOKE or DEPLOY).
UUID	The unique identifier for each transaction.
Chaincode ID	Refers to the chaincode that is being invoked or deployed.
Payload	The input parameters to the chaincode.

__72. Repeat this for other blocks to understand how the transactions are stored.

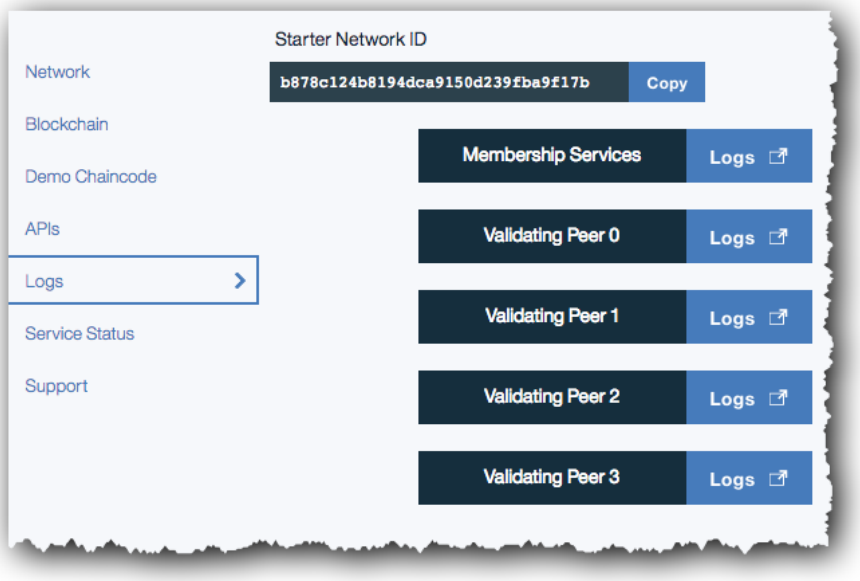
When the Blockchain service is initialised for the car leasing application, the first block in the chain should contain a 'DEPLOY' transaction, where the chaincode is deployed to the validating peers.

View these blocks If you're willing to scroll down the Blockchain explorer that far!

4.3. Understanding the Blockchain Peers

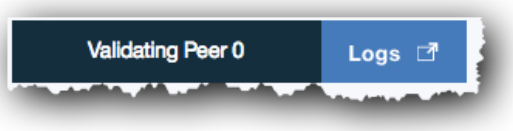
We are now going to review the logs associated with the peers. This is useful for understanding how the blockchain works, and for diagnosing problems.

__73. Click on the '**Logs**' tab.



By looking at the logs for each peer you can verify that every node has executed every transaction.

__74. Click the Logs button against one of the validating peers.



This will show the logs for the selected peer in a new window.

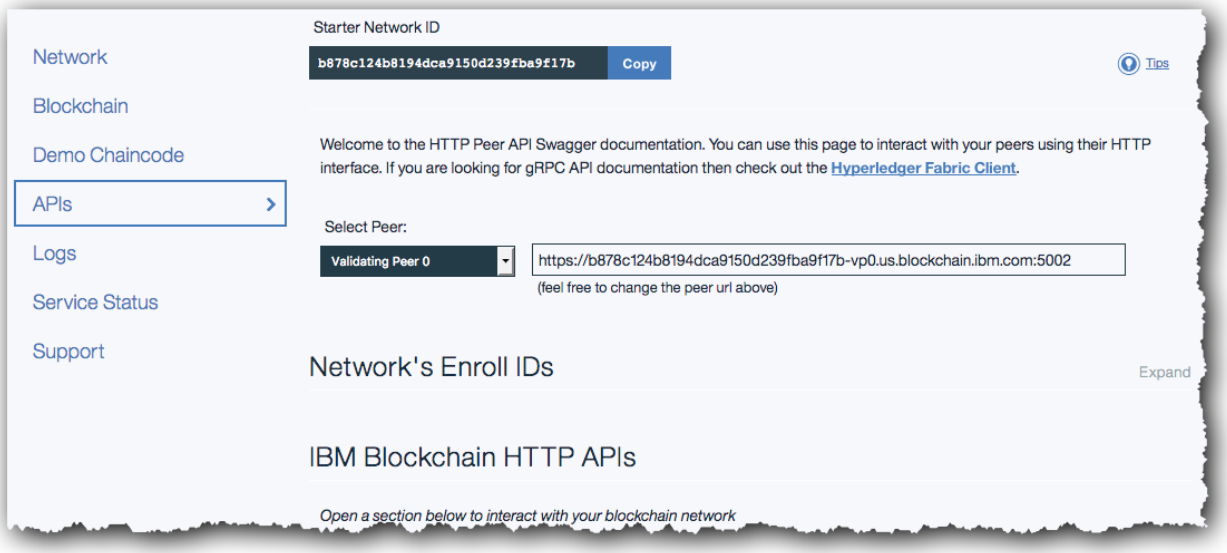
```
OUT - /scripts/start.sh -network_id b878c124b8194dca9150d239fba9f17b -peer_id vp0 -chaincode_host
prod-us-02-chaincode-swarm-vp0.us.blockchain.ibm.com -chaincode_port 3380 -network_name
us.blockchain.ibm.com -port_discovery 30002 -port_rest 5002 -port_event 31002 -peer_enrollid peer
-chaincode_tls true -peer_tls true -num_peers 4
OUT - Enrollment secret is not passed calculating the default
OUT -
CORE_PEER_ID="vp0",CORE_PEER_NETWORKID="b878c124b8194dca9150d239fba9f17b",CORE_PEER_ADDRESSAUTODE
="false",CORE_PEER_LISTENADDRESS="0.0.0.0:30002",CORE_REST_ADDRESS="0.0.0.0:5002",CORE_CLI_ADDRES
.0.0.0:30404",CORE_PEER_VALIDATOR_EVENTS_ADDRESS="0.0.0.0:31002",CORE_PEER_ADDRESS="b878c124b8194d
150d239fba9f17b-
vp0.us.blockchain.ibm.com:30002",CORE_LOGGING_PEER="warning",CORE_LOGGING_CRYPTO="warning",CORE_LO
NG_STATUS="warning",CORE_LOGGING_STOP="warning",CORE_LOGGING_LOGIN="warning",CORE_LOGGING_VM="debu
CORE_LOGGING_CHAINCODE="debug",CORE_PEER_LOGGING_LEVEL="warning",CORE_VM_ENDPOINT="tcp://prod-us-0
chaincode-swarm-
vp0.us.blockchain.ibm.com:3380",CORE_VM_DOCKER_TLS_ENABLED="true",CORE_VM_DOCKER_TLS_CERT_FILE="
/certs/chaincode_host/cert.pem",CORE_VM_DOCKER_TLS_KEY_FILE="/certs/chaincode_host
/key.pem",CORE_VM_DOCKER_TLS_CA_FILE="/certs/chaincode_host
/ca.pem",CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE="us.blockchain.ibm.com",CORE_PEER_TLS_ENABLED="tru
ORE_PEER_TLS_CERT_FILE="/certs/peer/cert.pem",CORE_PEER_TLS_KEY_FILE="/certs
/peer/key.pem",CORE_PEER_TLS_SERVERHOSTOVERRIDE="b878c124b8194dca9150d239fba9f17b-
vp0.us.blockchain.ibm.com",CORE_PEER_PKI_TLS_ENABLED="true",CORE_PEER_PKI_TLS_ROOTCERT_FILE="/cer
/peer/cert.pem",CORE_PEER_PKI_TLS_SERVERHOSTOVERRIDE="b878c124b8194dca9150d239fba9f17b-
vp0.us.blockchain.ibm.com",CORE_PEER_DISCOVERY_PERIOD="60s",CORE_PEER_DISCOVERY_TOUCHPERIOD="60s"
E_CHAINCODE_DEPLOYTIMEOUT="180000",CORE_CHAINCODE_STARTUPTIMEOUT="30000",CORE_PEER_VALIDATOR_CONSE
S_PLUGIN="pbft",CORE_PBFT_GENERAL_MODE="batch",CORE_PBFT_GENERAL_BATCHSIZE="1000",CORE_PBFT_GENERA
IMEOUT_BATCH="1s",CORE_PBFT_GENERAL_TIMEOUT_REQUEST="30s",CORE_PBFT_GENERAL_TIMEOUT_VIEWCHANGE="30
CORE_PBFT_GENERAL_TIMEOUT_RESENDVIEWCHANGE="30s",CORE_PBFT_GENERAL_TIMEOUT_NULLREQUEST="0s",CORE_S
ETRANSFER_TIMEOUT_SINGLEBLOCK="600s",CORE_STATETRANSFER_TIMEOUT_SINGLESTATEDELTA="600s",CORE_STATE
NSFER_TIMEOUT_FULLSTATE="600s",CORE_PEER_DISCOVERY_ROOTNODE="b878c124b8194dca9150d239fba9f17b-
vp1.us.blockchain.ibm.com:30002,b878c124b8194dca9150d239fba9f17b-
vp2.us.blockchain.ibm.com:30002,b878c124b8194dca9150d239fba9f17b-
vp3.us.blockchain.ibm.com:30002",CORE_SECURITY_ENABLED="true",CORE_SECURITY_ENROLLID="peer0",CORE
URITY_ENROLLSECRET="4c832819af",CORE_PEER_PKI_ECA_PADDR="b878c124b8194dca9150d239fba9f17b-
ca.us.blockchain.ibm.com:30002",CORE_PEER_PKI_TCA_PADDR="b878c124b8194dca9150d239fba9f17b-
ca.us.blockchain.ibm.com:30002",CORE_PEER_PKI_TLSCA_PADDR="b878c124b8194dca9150d239fba9f17b-
ca.us.blockchain.ibm.com:30002"
OUT - 2017-05-24 20:22:43,582 CRIT Supervisor running as root (no user in config file)
OUT - 2017-05-24 20:22:43,584 INFO supervisord started with pid 14
OUT - 2017-05-24 20:22:44,586 INFO spawned: 'start_peer' with pid 17
```

4.4. Interacting with the peers

It is possible to invoke the management APIs that interact directly with the peers. In this section we will be trying out these APIs directly from the Bluemix environment.

Note that the APIs concern *operationally managing* the Blockchain service – this is not the same as adding and invoking transactions through chaincode!

__75. Click on the '**APIs**' tab on the dashboard.



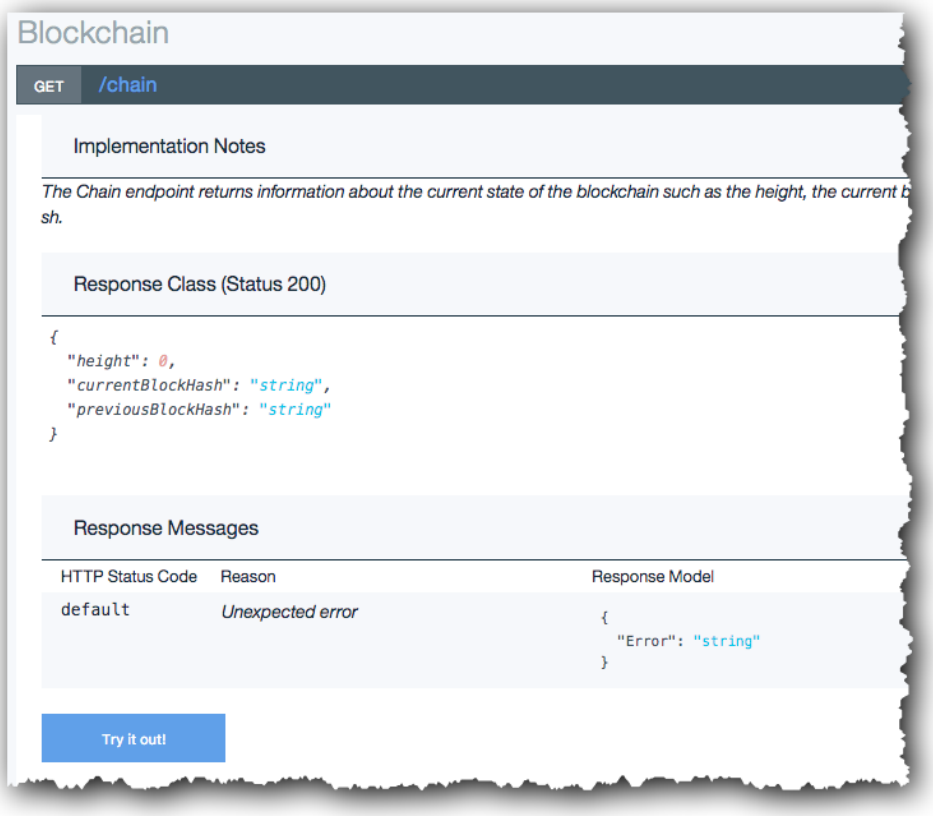
This page allows you to invoke APIs that will directly interrogate and manage the blockchain. First we will use the API interface to query the height of the Blockchain (the number of blocks).

__76. Click the '**Blockchain**' section.

This reveals the **GET /chain** operation which is a valid operation to call on the peer.

__77. Select the operation to view information about it.

This reveals the input and output data formats.



__78. Click '**Try It Out**' to invoke the API.

Review the displayed fields:

- The *Curl* field shows how to perform the same request from a command-line or script.
- The *Request URL* shows the URL that was invoked, including the endpoint information of the peer (hostname:port) and the method call (/chain).
- The *Response Body* shows the information that was returned including, importantly, the height of the blockchain.
- The *Response Code* 200 shows that the request was successful.
- The *Response Headers* confirms that the response body has been returned in a JSON data structure.

The blockchain is immutable: it is append-only and transactions cannot be modified or deleted once committed to the blockchain. Hash functions are used to link the blocks in the chain together; each block is linked to the previous block by a hash of the previous block's contents. If transactions are tampered with, the hash function returns a different value which renders the blockchain un-navigable.

A hash function is simply a function that is applied to a data set that produces a consistent output. It is usually used to map data of an arbitrary size to data of a fixed size. Importantly for blockchain, any change to the input data set will produce a different hash output, which can be used to easily detect any modifications to a block.

In the JSON response body, the '*height*' field shows the number of blocks in the blockchain; the '*currentBlockHash*' is a hash function that has run over the most recent block and '*previousBlockHash*' is the same for the block before it.

__79. Note the first few characters of the value of the *currentBlockHash* ("JIS21…" in the previous screenshot).

__80. Invoke another transaction in the car leasing demo to force another block to be created.

__81. Re-run the GET /chain operation; verify that the height has increased by 1 and that the new *previousBlockHash* is the same as the previous block's *currentBlockHash* ("JIS21…").

__82. Click the 'Block' section and click on the 'GET /chain/blocks/{Block}' operation. In the 'Block' text field, enter the number *one less* than the current height of the chain (for example if the height was 85, enter 84).

```
{
 "transactions": [
  {
   "type": 2,
   "chaincodeID": "EkA2YjNjYjQyZWQyZDhiZDdmNzcwMmUxMmFmYzViMDViZmQzNWY4YmM0MzU5NTk
   "payload": "Cn0IARJCEkA2YjNjYjQyZWQyZDhiZDdmNzcwMmUxMmFmYzViMDViZmQzNWY4YmM0Mz
   "txid": "352a7417-5d28-4199-b490-38e583ab0d9b",
   "timestamp": {
    "seconds": 1495721844
   },
   "nonce": "/9xRMDyuvhHO9BetrkGj10FCiJnvij6Z",
   "cert": "MIICmDCCAj6gAwIBAgIRAP4mqp72201Yq+7F6Dgh/EUwCgYIKoZIzj0EAwMwKTELMAkGA1UEB
   "signature": "MEUCIB8dics+LyfITYO0Sg2mHiS+W5roQRM3K53E50MzHr6KAiEAnLbPZapxNod1Z958NU
  }
 ],
 "stateHash": "wEIXSjQ5X4E0RqgQPwG1QKDf41smPY2fbECWwCFfa8aLDc62WDNxT0FkpRV06RwVHMB
 "previousBlockHash": "JIS21vw/U6bxLloXaUNBr/sAO54NqpZKHYVTCd+oZyUnnTeHFKfm9a/55nvkecG2Tr
 "consensusMetadata": "CFO=",
```


The returned JSON structure contains several elements, including:

(a) a *transactions* array, which describes the set of transactions in the block. The description of each transaction includes its type (1/invoke or 2/deploy), the unique identifier of the associated chaincode and the encoded input parameters to it (payload).

(b) a *stateHash*, which is the result of running a hash function over the transaction output,

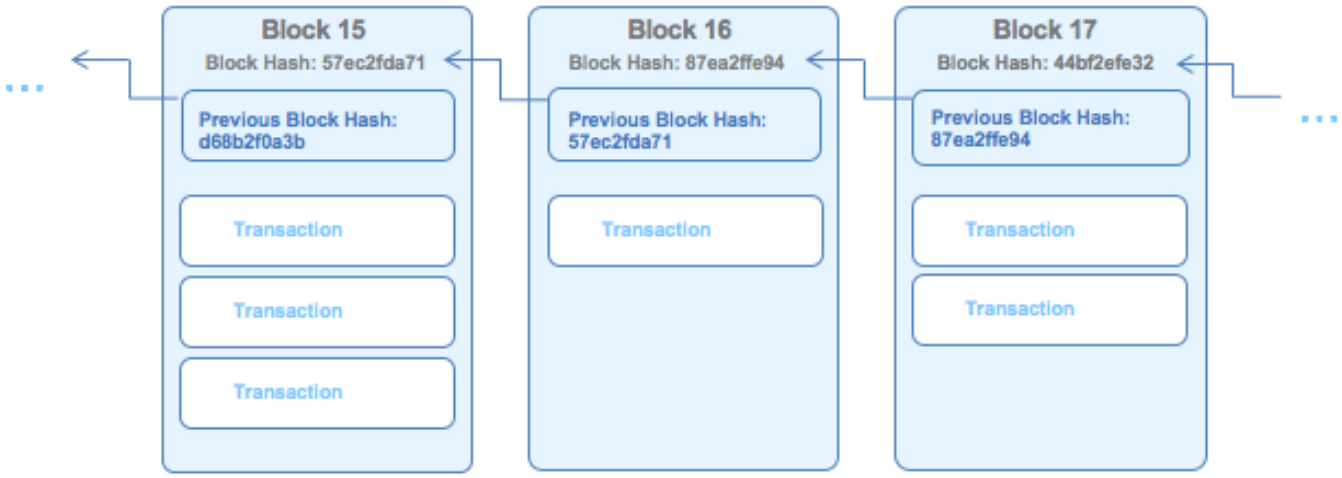
(c) a *previousBlockHash*, which is the result of running a hash function of the previous block in the chain, and

(d) a *nonHashData* element, which contains data that will *not* be used in the computation of the next block's *previousBlockHash*.

Note that the *previousBlockHash* field matches the *previousBlockHash* returned by the GET /chain operation.

When a new block is created, a hash function is run over the entire previous block (except the *nonHashData* element) and the result stored in the *previousBlockHash* element. This way, if any earlier block in the chain is tampered with, subsequent blocks will be invalid.

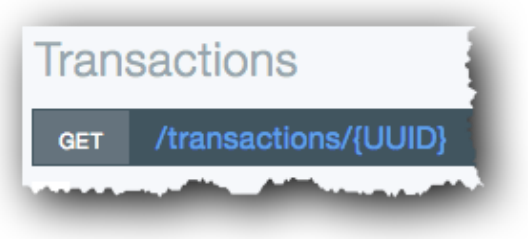
Particularly, note that the *previousBlockHash* element is *itself* used to calculate the hash of subsequent blocks. This means that even a small change to one of the first blocks in the chain can be detected in any future block.



We will look at some other fields in this data structure in the next section.

__83. Copy the *txid* field of a transaction from a block; this will be a unique identifier of the form "04421f7d-652a-491d-90b0-7bc9f29b2d85".

__84. Click the '**Transactions**' section.



This reveals the **GET /transactions/{UUID}** operation which is a valid method to call on the peer.

__85. Paste the transaction UUID and click '**Try it out!**'.

The 'payload' field is base64 encoded (use a web tool such as http://www.base64decode.org for decoding this information); when decoded you'll see that the payload includes the chaincode ID of the smart contract being called together with its input parameters. For example:

Note that this application does not encrypt the transactions, so the payloads are visible (albeit base64 encoded) to all.

4.5. Viewing the Service Status, Support Contacts and Samples

__86. Click on the '**Status**' tab at the top of the service page.

This page shows you the recent availability of the Blockchain service on Bluemix, and also the version of Hyperledger Fabric that is being used by your network.

__87. Click on the '**Support**' tab at the top of the service page.

This page shows you how to get more help with IBM Bluemix and the Blockchain service.

__88. Click on the '**Demo Chaincode**' tab at the top of the service page.

This page gives the opportunity to deploy more samples to the Blockchain service, and also some how to get started with writing your own blockchain applications and chaincode.

We will look at chaincode development in more detail in the follow-on lab "Blockchain Unchained".

Section 5. Security Fundamentals

Despite not using or requiring cryptographic mining, Hyperledger Fabric uses cryptography and other security principals to great effect. In this section of the lab we will look at how the underlying technology uses concepts such as signing and certificates.

5.1. Transaction Signing

__89. In the blockchain dashboard, click ‘APIs’.

__90. If necessary, re-run the GET /chain/blocks/{Block} operation to return the details of a single block.


```
{
  "transactions": [
    {
      "type": 2,
      "chaincodeID":
"EoABMTZlNjU1YzBmY2U2YTk4ODI4OTZkM2Q2ZDExZjdkY2Q0ZjQ1MDI3ZmQ0NzY0MDA0NDQwZmYxZTYxMzQwOTEwYTlkNjc2ODVjNGJiNzIzMjcy
YTQ5N2YzY2Y0MjhlNmNmNmIwMDk2MTg2MTIyMjBlMTQ3MWUwM2I2YzBhYTc2Y2I=",
      "payload":
"CqQBCAESgwESgAExNmU2NTVjMGZjZTZhOTg4Mjg5NmQzZDZkMTFmN2RjZDRmNDUwMjdmZDQ3NjQwMDQ0NDBmZjFlNjEzNDA5MTBhOWQ2NzY4NWM0
YmI3MjMyNzJhNDk3ZjNjZjQyOGU2Y2Y2YjAwOTYxODYxMjIyMGUxNDcxZTAzYjZjMGFhNzZjYhoaCghzZXRfdXNlcgoHcnZ5OTIwZQoFbGVyb3k="
,
      "txid": "256cb7a9-1c50-47a8-a078-0bb5bf96fc79",
      "timestamp": {
        "seconds": 1479732624,
        "nanos": 860673029
      },
      "nonce": "Hil0TKfQb80X6WuAzF4L2f+O1WtDZ81r",
      "cert":
"MIICQDCCAeegAwIBAgIQDBdV+AtiS7SQwj+UhA36rTAKBggqhkjOPQQDAzApMQswCQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMQwwCgYDVQQDEwN0
Y2EwHhcNMTYxMTIxMTIzNjQyWhcNMTcwMjE5MTIzNjQyWjA9MQswCQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMSAwHgYDVQQDExdUcmFuc2FjdGlvb
iBDZXJ0aWZpY2F0ZTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABLrxmY9TA2KWhSe0G7jwvLT7hEF55sTQyQZB0s5ifLcMmSlpqzqrzNgwaLeMgf
rp3iOMmMCwAVc9ylfWU30eILejgdwwgdkwDgYDVR0PAQH/BAQDAgeAMAwGA1UdEwEB/wQCMAAwDQYDVR0OBAYEBAECAwQwDwYDVR0jBAgwBoAEAQI
DBDBNBgYqAwQFBgcBAf8EQPR58QFNovEdqgHctksWMJ++AKg5rsWINnJVLnlVPyocrTXehS7HHMSDl+stv1+GKsVasbFmQIY1PhStJLQ31a4wSgYG
KgMEBQYIBECZj3vpacz9rlXCfQy6nTLKEhc1HQjDJGIR5jrHaK7jpoYItcQA2Ae6nFlJRqkePunKg5c7wV4RkdI0pz+rqkvfMAoGCCqGSM49BAMDA
0cAMEQCIFIpIruyutTlbKXLyNzzfC8N4hxzOQvcABzDU926i8wYAiAJudsovUZketRmEzR/CwpAZqTJ6f1pjZ/FG9Qa4V8Geg==",
      "signature":
"MEQCIHkIzR2C8phQB2/sla/MVg9bZtHYZ/rUJJKFehJEZ7DWAiAQAni1NZ7QtGhOU7hnNdcpkPPSWWnqgBzQPQklrdzu1g=="
    }
  ],
  "stateHash": "HpBslIqtVTXTi4tVOKp7gKD6WVrLs9mjM7Z3O7p0mYHIiXkcHrseQNdCRuEqkFjYBfBcOyt1ykSXXR0s5o5sHA==",
  "previousBlockHash":
"LSK+4Im1LZB0HimLyzTeQnsK3FUdB6dWf9PjyvQtJFst5t6Mbkuzn1fVDKFvuQLU3qnIW1ggSzbrnlfxu1qaeg==",
  "consensusMetadata": "CAQ=",
  "nonHashData": {
    "localLedgerCommitTimestamp": {
      "seconds": 1479732625,
      "nanos": 865115723
    },
    "chaincodeEvents": [
      {}
    ]
  }
}
```


The block also reveals information related to privacy and traceability.

Unlike the pseudonymous Bitcoin network, this blockchain can also be private and permissioned. Public key cryptography can ensure that the membership of the network is restricted to authorised participants, and that individual transactions are traceable (i.e. can be signed) and have appropriate privacy (i.e. can be encrypted).

This demo does not encrypt its transaction payloads but the transactions are signed.

__91. Review the ‘cert’ and ‘signature’ elements of an individual transaction.


```
"cert":
"MIICQDCCAeegAwIBAgIQDBdV+AtiS7SQwj+UhA36rTAKBggqhkjOPQQDAzApMQswCQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMQwwCgYDVQQDEwN0
Y2EwHhcNMTYxMTIxMTIzNjQyWhcNMTcwMjE5MTIzNjQyWjA9MQswCQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMSAwHgYDVQQDExdUcmFuc2FjdGlvb
iBDZXJ0aWZpY2F0ZTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABLrxmY9TA2KWhSe0G7jwvLT7hEF55sTQyQZB0s5ifLcMmSlpqzqrzNgwaLeMgf
rp3iOMmMCwAVc9ylfWU30eILejgdwwgdkwDgYDVR0PAQH/BAQDAgeAMAwGA1UdEwEB/wQCMAAwDQYDVR0OBAYEBAECAwQwDwYDVR0jBAgwBoAEAQI
DBDBNBgYqAwQFBgcBAf8EQPR58QFNovEdqgHctksWMJ++AKg5rsWINnJVLnlVPyocrTXehS7HHMSDl+stv1+GKsVasbFmQIY1PhStJLQ31a4wSgYG
KgMEBQYIBECZj3vpacz9rlXCfQy6nTLKEhc1HQjDJGIR5jrHaK7jpoYItcQA2Ae6nFlJRqkePunKg5c7wV4RkdI0pz+rqkvfMAoGCCqGSM49BAMDA
0cAMEQCIFIpIruyutTlbKXLyNzzfC8N4hxzOQvcABzDU926i8wYAiAJudsovUZketRmEzR/CwpAZqTJ6f1pjZ/FG9Qa4V8Geg==",
      "signature":
"MEQCIHkIzR2C8phQB2/sla/MVg9bZtHYZ/rUJJKFehJEZ7DWAiAQAni1NZ7QtGhOU7hnNdcpkPPSWWnqgBzQPQklrdzu1g=="
```


The “*signature*” element is the output of running a function over the transaction input data using the private key of the initiator of the transaction. As the private key is a secret known only to the transaction initiator, it has the effect of proving who initiated the transaction; it is a *digital signature*.

The “*cert*” element is the public transaction certificate of the transaction initiator.

It is possible to base64 decode the *cert* element (e.g. using www.base64decode.org) to see the human-readable fields of the certificate. It is also possible to format the data in a form that can be read by the *OpenSSL* tools (www.openssl.org), as shown in the next step.

__92. [OPTIONAL] To view the certificate details, create a file called test.pem that contains the certificate information bounded by -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----, and separate into 65-column lines as per the following screenshot:


```
-----BEGIN CERTIFICATE-----
MIICQDCCAeegAwIBAgIQDBdV+AtiS7SQwj+UhA36rTAKBggqhkjOPQQDAzApMQsw
CQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMQwwCgYDVQQDEwN0Y2EwHhcNMTYxMTIx
MTIzNjQyWhcNMTcwMjE5MTIzNjQyWjA9MQswCQYDVQQGEwJVUzEMMAoGA1UEChMD
SUJNMSAwHgYDVQQDExdUcmFuc2FjdGlvbiBDZXJ0aWZpY2F0ZTBZMBMGByqGSM49
AgEGCCqGSM49AwEHA0IABLrxmY9TA2KWhSe0G7jwvLT7hEF55sTQyQZB0s5ifLcM
mSlpqzqrzNgwaLeMgfrp3iOMmMCwAVc9ylfWU30eILejgdwwgdkwDgYDVR0PAQH/
BAQDAgeAMAwGA1UdEwEB/wQCMAAwDQYDVR0OBAYEBAECAwQwDwYDVR0jBAgwBoAE
AQIDBDBNBgYqAwQFBgcBAf8EQPR58QFNovEdqgHctksWMJ++AKg5rsWINnJVLnlV
PyocrTXehS7HHMSDl+stv1+GKsVasbFmQIY1PhStJLQ31a4wSgYGKgMEBQYIBECZ
j3vpacz9rlXCfQy6nTLKEhc1HQjDJGIR5jrHaK7jpoYItcQA2Ae6nFlJRqkePunK
g5c7wV4RkdI0pz+rqkvfMAoGCCqGSM49BAMDA0cAMEQCIFIpIruyutTlbKXLyNzz
fC8N4hxzOQvcABzDU926i8wYAiAJudsovUZketRmEzR/CwpAZqTJ6f1pjZ/FG9Qa
4V8Geg==
-----END CERTIFICATE-----
```


Then in a shell window run the command:

```
openssl x509 -in test.pem -text
```

The output will show the certificate information:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0c:17:55:f8:0b:62:4b:b4:90:c2:3f:94:84:0d:fa:ad
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: C=US, O=IBM, CN=tca
        Validity
            Not Before: Nov 21 12:36:42 2016 GMT
            Not After : Feb 19 12:36:42 2017 GMT
        Subject: C=US, O=IBM, CN=Transaction Certificate
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
            EC Public Key:
                pub:
                    04:ba:f1:99:8f:53:03:62:96:85:27:b4:1b:b8:f0:
                    bc:b4:fb:84:41:79:e6:c4:d0:c9:06:41:d2:ce:62:
                    7c:b7:0c:99:29:69:ab:3a:ab:cc:d8:30:68:b7:8c:
                    81:fa:e9:de:23:8c:98:c0:b0:01:57:3d:ca:57:d6:
                    53:7d:1e:20:b7
                ASN1 OID: prime256v1
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                01:02:03:04
            X509v3 Authority Key Identifier:
                keyid:01:02:03:04

            1.2.3.4.5.6.7: critical
                .y..M.......K.0....9...6rU.yU?*..5.........-._.*.Z..f@.5>..$.7..
            1.2.3.4.5.6.8:
                ..{.i...U.}...2...5...$b..:.h...........YIF..>....;.^...4.?..K.
    Signature Algorithm: ecdsa-with-SHA384
        30:44:02:20:52:29:22:bb:b2:ba:d4:e5:6c:a5:cb:c8:dc:f3:
        7c:2f:0d:e2:1c:73:39:0b:dc:00:1c:c3:53:dd:ba:8b:cc:18:
        02:20:09:b9:db:28:bd:46:64:7a:d4:66:13:34:7f:0b:0a:40:
        66:a4:c9:e9:fd:69:8d:9f:c5:1b:d4:1a:e1:5f:06:7a
```

More information on the signing and encryption methods used can be found here:
https://github.com/hyperledger/fabric/blob/master/docs/protocol-spec.md

5.2. Membership Services Concepts

Now we will use the Membership Services component of the blockchain. This is a built-in Certificate Authority that provides us with the public/private key pairs that we will use on the blockchain. These were the certificates that we saw in the previous section.

Encryption and signing on the blockchain is done using standard public/private key pairs. It is assumed that everyone on the network can view public keys, and that only a single authorised user holds the private key. A public/private key pair is distributed in the form of a certificate.

Data that is signed with a private key can be verified by anyone with the public key, and is used on blockchain like a real-world signature - to prove the origin of data or code, and/or to state agreement to something.

Data that is encrypted with a public key can only be decrypted by the holder of the private key. This is used on blockchain to ensure privacy. Data can be encrypted multiple times in such a way that any one of a set of users can decrypt it (for example, both beneficiaries of a transaction).

The certificate authority on the blockchain, also known as the membership services component, is able to provide the user with two types of certificate:

- Enrollment certificate: A public/private key pair that is used to provide the user with an identity on the network. It is typically long lasting, like a user-name.

- Transaction certificate: A public/private key pair that is used to sign a transaction. It is typically used only once, like a one-time-use throwaway credit card number. It is not possible for all users to infer the owner of a transaction certificate.

A given user might have one enrollment certificate but many hundreds of transaction certificates, depending on how many transactions they wish to commit to the blockchain.

It is important to understand why the blockchain does this. Why does the blockchain have these two types of certificate? Why does the user not simply use the enrollment certificate for the signing of all transactions?

The reason is that the transaction certificate gives us something called *unlinkable identity*.

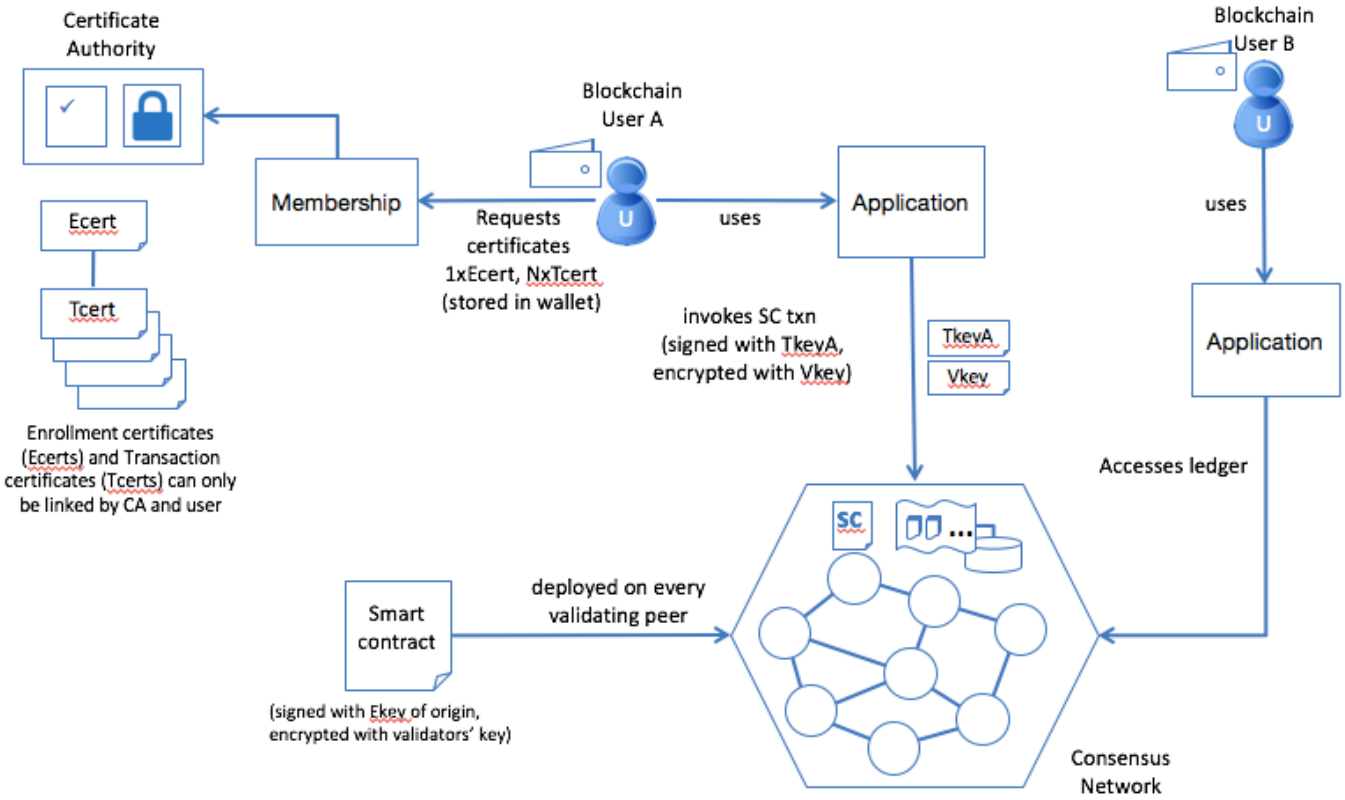
Let's assume Alice, Bob and Charlie form part of a business network and share a blockchain for transactions. Alice is dealing with Bob, and she is also dealing with Charlie. Crucially, Alice should never know that Bob and Charlie have done business together, as this knowledge could violate confidentiality between Bob and Charlie, and thereby give Alice a competitive advantage. For example, if every time Alice transacted with Bob, Alice knew that a transaction subsequently occurred between Bob and Charlie, she could assume that Bob is a middle-man and could avoid working with Bob in the future.

If we only ever used enrollment certificates for signing transactions on the blockchain, Alice could infer all the transactions involving either Bob or Charlie, because we know she has access to their public enrollment keys. By using transaction certificates on the other hand, no identity can be inferred because the certificates are used only once.

There is nothing stopping users from using transaction certificates multiple times, or indeed from using their enrollment certificates to sign transactions, but doing so increases the risk that their identity can be inferred.

Blockchains for business are generally regulated in some way. A regulator might have priviledged access to the certificate authority, and thus be able to access the private keys of participants, and also be able to see which transaction certificates are owned by which users. For this reason, the certificate authority is a trusted environment, in much the same way that a passport office or drivers license authority is trusted. A compromised certificate authority will compromise all transactions that used certificates from it.

The security concepts of the blockchain can be summarised in the following diagram. A smart contract is simply a piece of user code deployed to the blockchain, and a transaction is defined as a set of input parameters to that code.



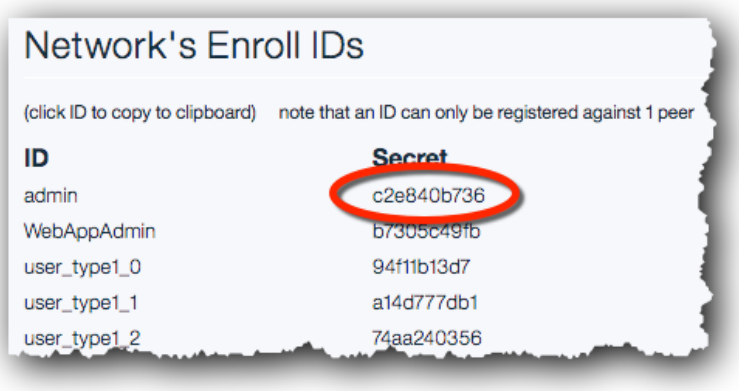
We will now use the membership services component to enroll on the blockchain network, provide us our enrollment certificate and issue some new sample transaction certificates.

5.3. Using Membership Services

__93. In the blockchain dashboard 'APIs' tab, click "Network's Enroll IDs".

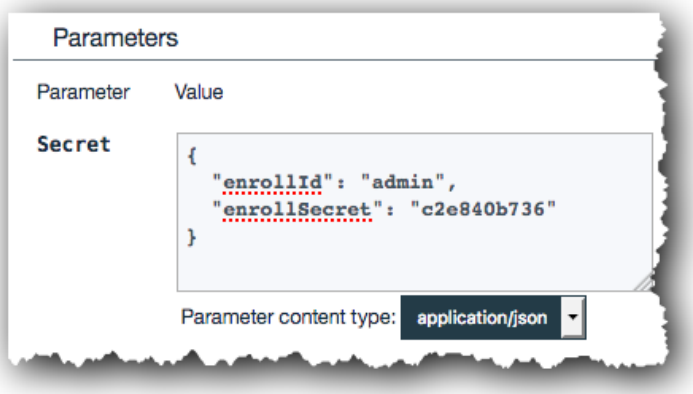
This reveals the available user IDs and passwords (secrets). This is a convenience for the purposes of this demo, and clearly would not be available in a real-world blockchain.

__94. Click the secret next to the 'admin' user to copy it to the clipboard.



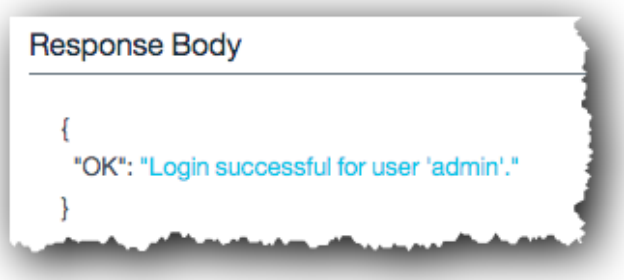
__95. In the Registrar section, click the POST /registrar operation.

__96. Fill in the JSON structure for the 'Secret' parameter, as follows. The enrollSecret parameter should match the secret you just copied to the clipboard.



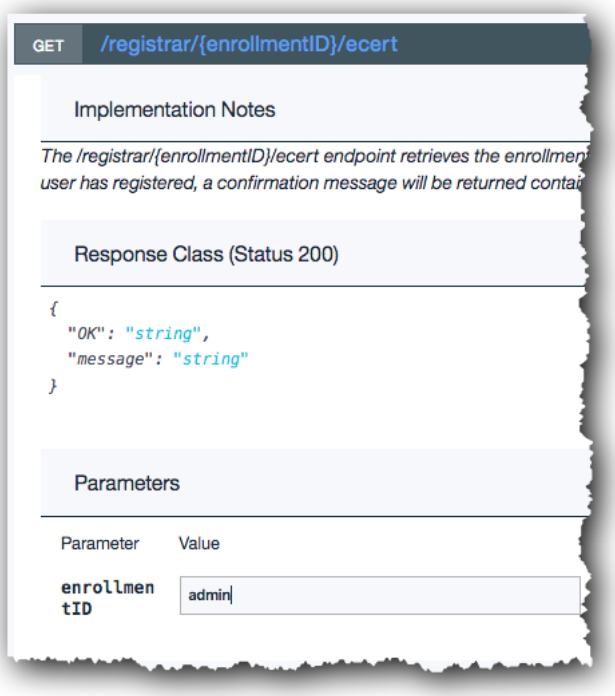
__97. Click 'Try it out!'.

You should get an OK response back from the certificate authority.



__98. Select the 'GET /registrar/{enrollmentID}/ecert' operation.

__99. Enter 'admin' as the enrollmentID.



__100. Click ‘Try it out!’.

The enrollment certificate for the user will be returned.

__101. Do the same for the ‘GET /registrar/{enrollmentID}/tcert’ operation. Use ‘admin’ as the enrollmentID and ‘5’ as the count.

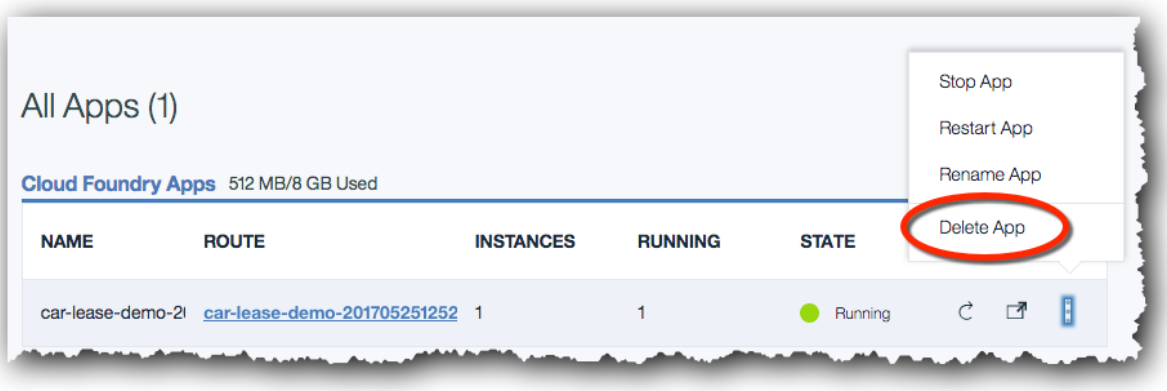
You will receive a set of five transaction certificates. As the developer of a blockchain application, you could then use these certificates for signing new transactions.

Section 6. Removing the sample application

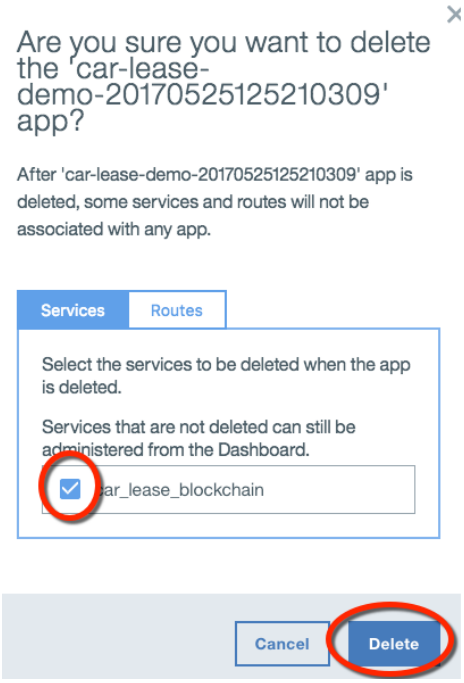
The final section of this lab aims to stop and remove the Blockchain service you created.

__102. Return to the Bluemix Dashboard (https://console.ng.bluemix.net/dashboard/applications).

__103. Click the three vertical dots at the right of the Car Leasing application Settings icon in the car lease demo application and select '**Delete App**' from the menu.



__104. Ensure that the 'car_lease_blockchain' service is selected for deletion and click '**Delete**'.



__105. Wait for the items to be stopped and deleted. Once this is done, both the application and the associated blockchain service will no longer be visible in the Bluemix dashboard. The Continuous Delivery service will remain and be deleted if required.

Appendix A. Notice

This information was developed for products and services offered in the U.S.A.
IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.
For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.
Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated

Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	*Tivoli*	*WebSphere*	*Workplace*	*System p*	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.