IBM Blockchain Hands-On Blockchain Explored

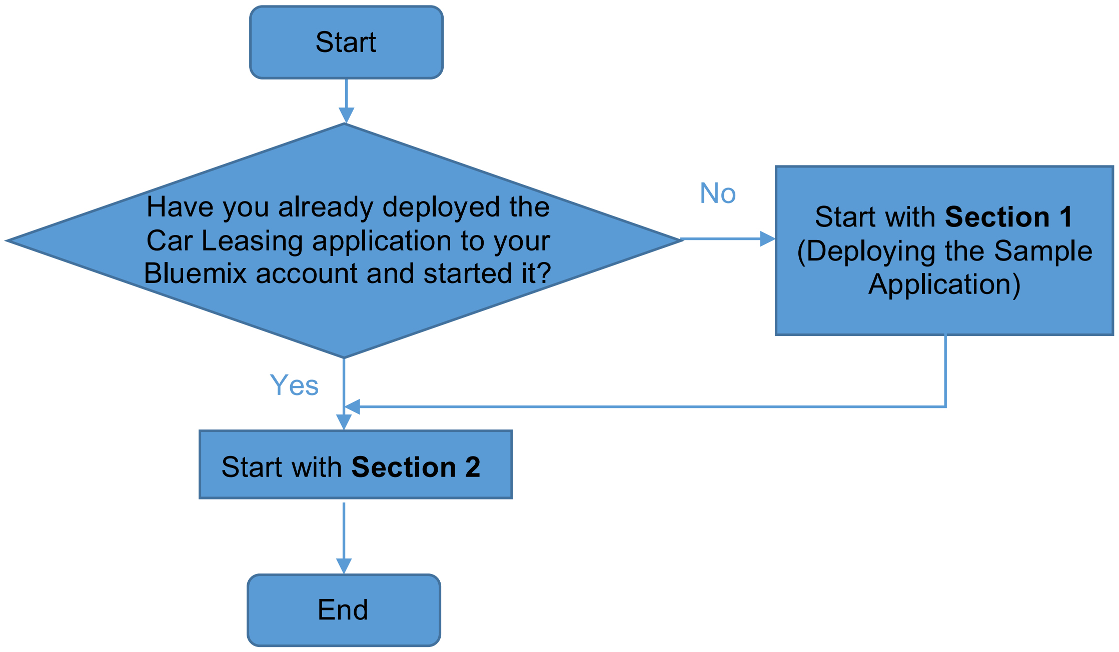
Lab Two – Bluemix – Exercises

Contents

Overview Introduction to the Lab

The purpose of this lab is to introduce you to the Blockchain service on Bluemix. We will build on the car leasing scenario that was introduced in the “Blockchain Explained” lab.

If you are using your own Bluemix account and have already completed the previous lab, you will have already deployed the car leasing application to your account. You can skip section 1 and re-use your existing application:


Start
Have you already deployed the Car Leasing application to your Bluemix account and started it?
No
Start with **Section 1** (Deploying the Sample Application)
Yes
Start with **Section 2**
End

For Hyperledger Fabric V0.6 in Bluemix

Section 1. Deploying the sample application

In this section we will use Bluemix to deploy a copy of the car leasing demo application.

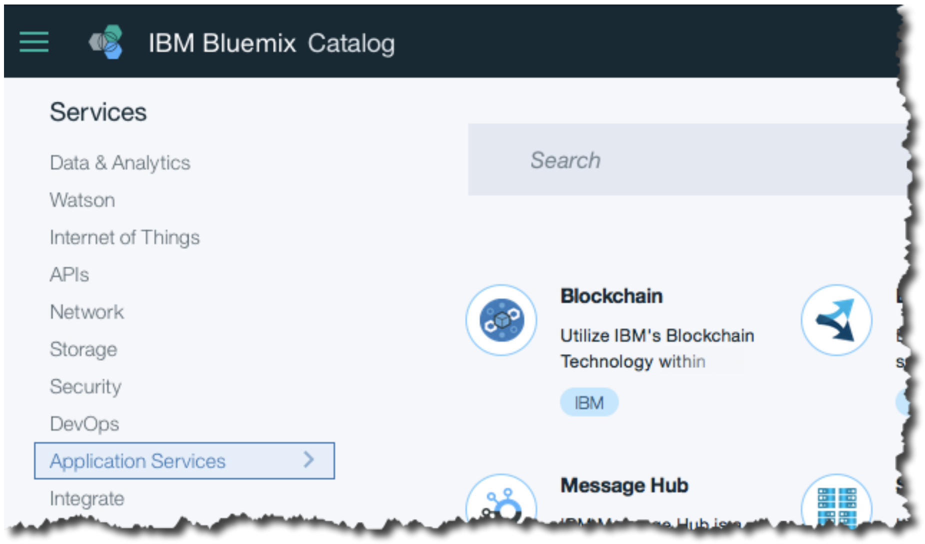
1.1. Creating a Blockchain Service

__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.

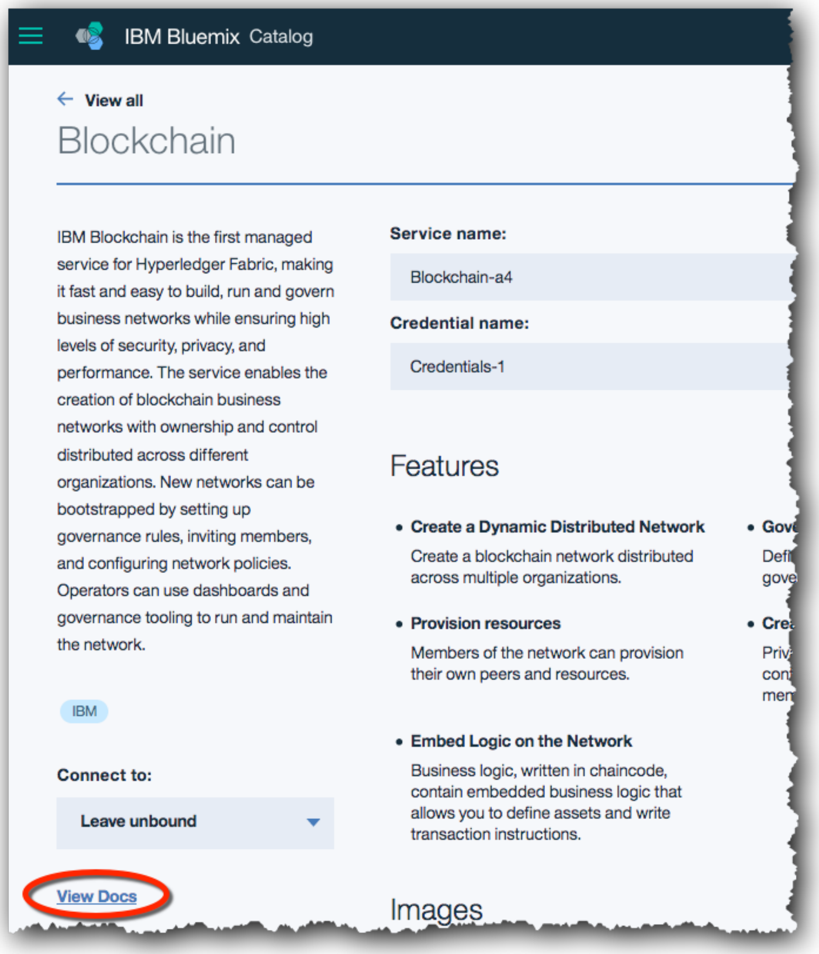
__3. Once you have successfully signed up and logged into Bluemix, select Catalog from the top bar.

__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.



Blockchain

IBM Blockchain is the first managed service for Hyperledger Fabric, making it fast and easy to build, run and govern business networks while ensuring high levels of security, privacy, and performance. The service enables the creation of blockchain business networks with ownership and control distributed across different organizations. New networks can be bootstrapped by setting up governance rules, inviting members, and configuring network policies. Operators can use dashboards and governance tooling to run and maintain the network.

Connect to: Leave unbound

Service name: Blockchain-a4

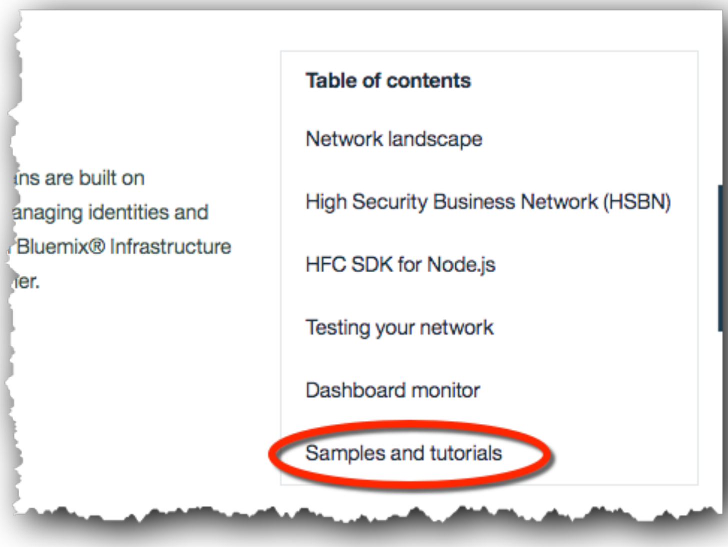
Credential name: Credentials-1

Features

- **Create a Dynamic Distributed Network**
  Create a blockchain network distributed across multiple organizations.
- **Provision resources**
  Members of the network can provision their own peers and resources.
- **Embed Logic on the Network**
  Business logic, written in chaincode, contain embedded business logic that allows you to define assets and write transaction instructions.

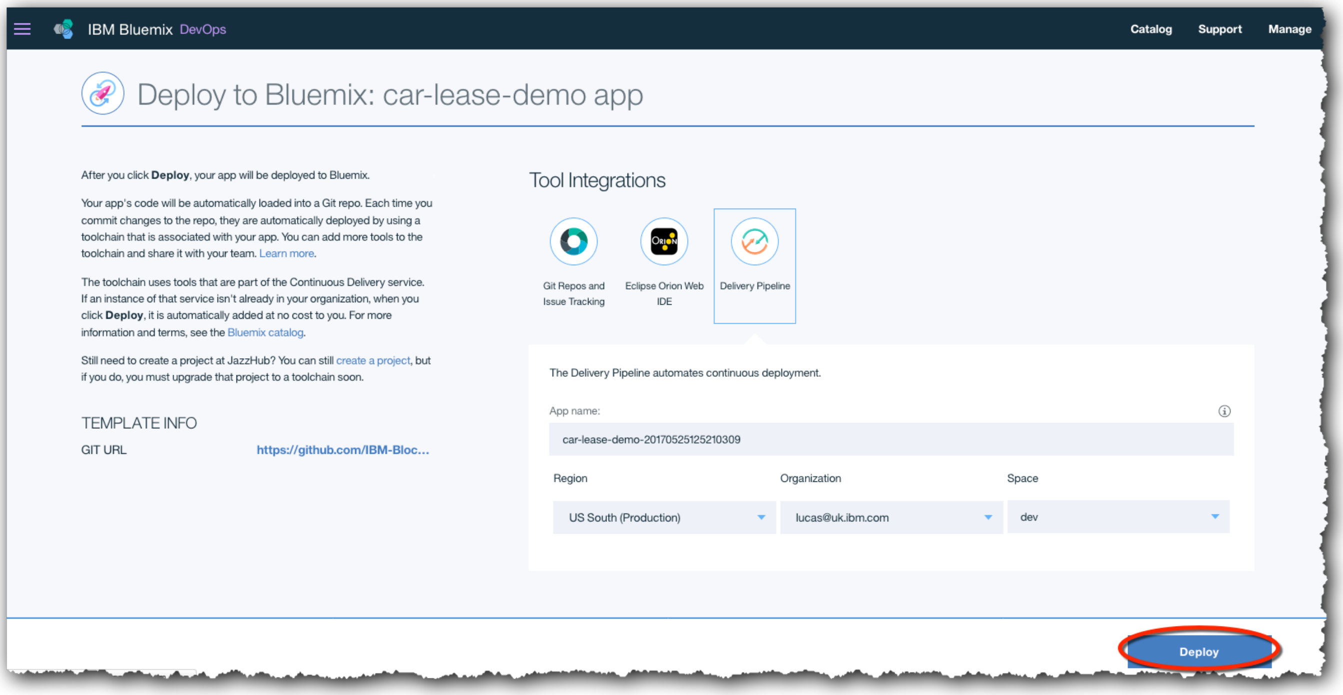
__7. Click ‘**HSBN and Starter plans**’ on the left of the page.

__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 **Deploy to Bluemix** against the Car Lease demo.

__10. Click '**Deploy**'.



Deploy to Bluemix: car-lease-demo app

After you click **Deploy**, your app will be deployed to Bluemix.

Your app's code will be automatically loaded into a Git repo. Each time you commit changes to the repo, they are automatically deployed by using a toolchain that is associated with your app. You can add more tools to the toolchain and share it with your team. Learn more.

The toolchain uses tools that are part of the Continuous Delivery service. If an instance of that service isn't already in your organization, when you click **Deploy**, it is automatically added at no cost to you. For more information and terms, see the Bluemix catalog.

Still need to create a project at JazzHub? You can still create a project, but if you do, you must upgrade that project to a toolchain soon.

TEMPLATE INFO

GIT URL https://github.com/IBM-Bloc...

Tool Integrations

The Delivery Pipeline automates continuous deployment.

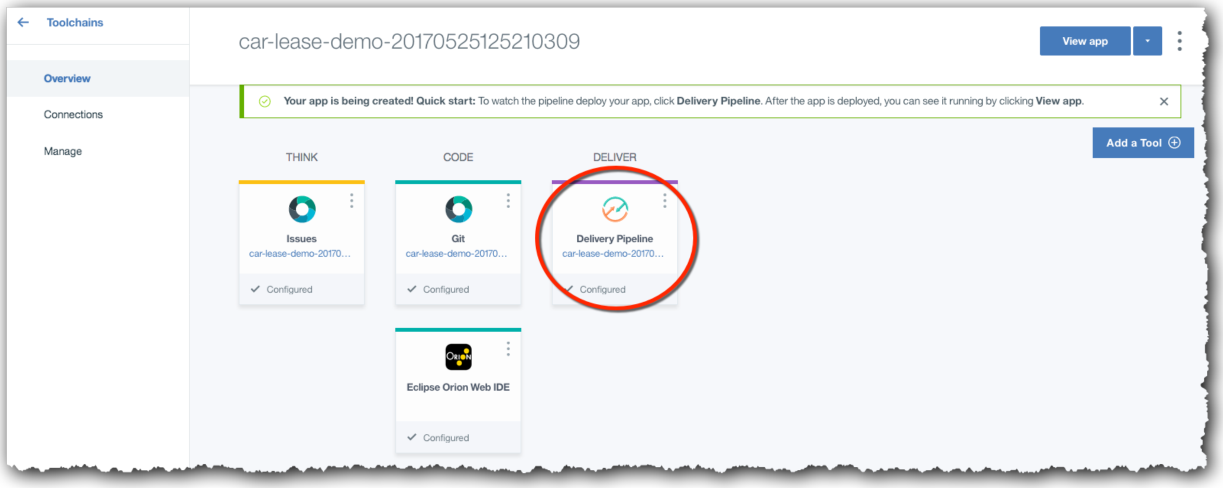
App name: car-lease-demo-20170525125210309

Region: US South (Production)

Organization: lucas@uk.ibm.com

Space: dev

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

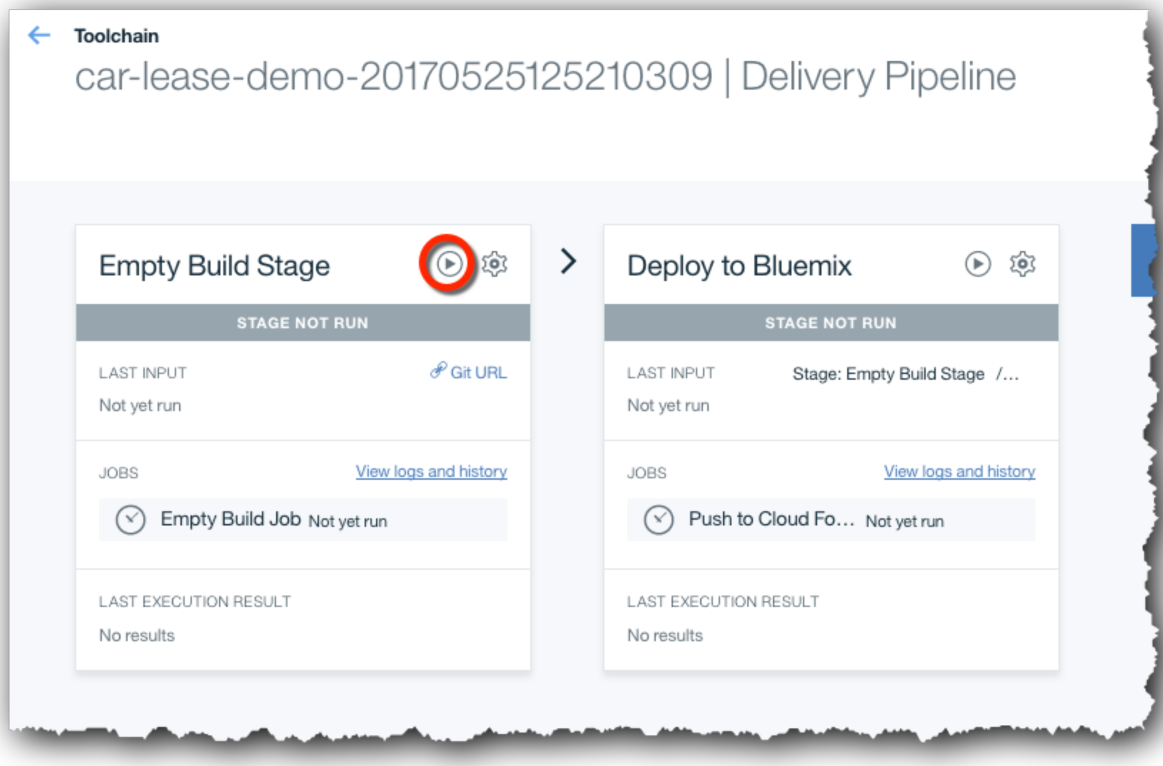


car-lease-demo-20170525125210309

Your app is being created! Quick start: To watch the pipeline deploy your app, click **Delivery Pipeline**. After the app is deployed, you can see it running by clicking **View app**.

THINK	CODE	DELIVER
Issues car-lease-demo-20170... ✓ Configured	Git car-lease-demo-20170... ✓ Configured	Delivery Pipeline car-lease-demo-20170... ✓ Configured
	Eclipse Orion Web IDE ✓ Configured	

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


Toolchain

car-lease-demo-20170525125210309 | Delivery Pipeline

Empty Build Stage

STAGE NOT RUN

LAST INPUT Git URL

Not yet run

JOBS View logs and history

Empty Build Job Not yet run

LAST EXECUTION RESULT

No results

Deploy to Bluemix

STAGE NOT RUN

LAST INPUT Stage: Empty Build Stage /...

Not yet run

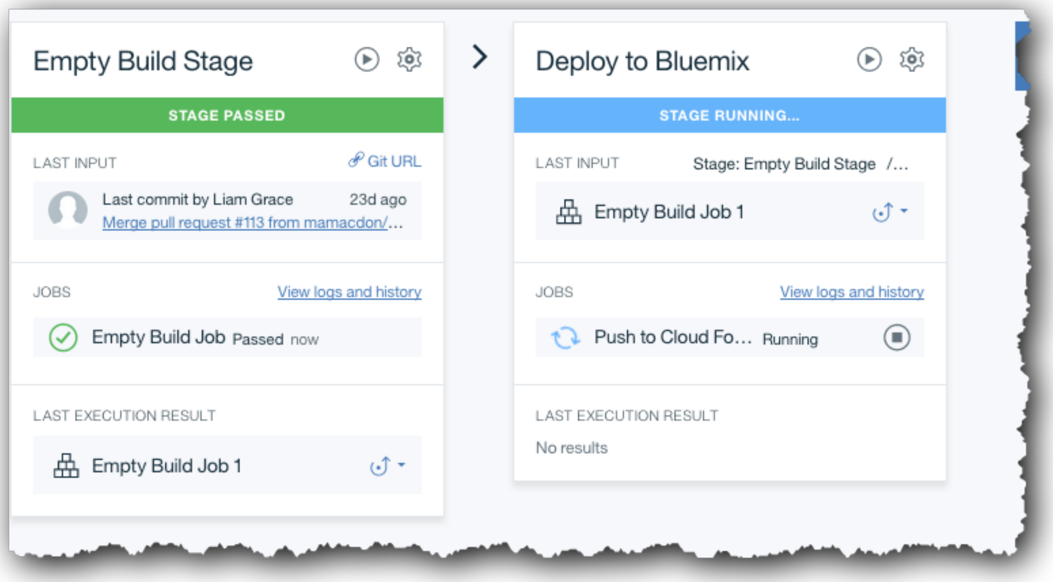
JOBS View logs and history

Push to Cloud Fo... Not yet run

LAST EXECUTION RESULT

No results

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


Empty Build Stage

STAGE PASSED

LAST INPUT Git URL

Last commit by Liam Grace 23d ago

Merge pull request #113 from mamacdon/...

JOBS View logs and history

Empty Build Job Passed now

LAST EXECUTION RESULT

Empty Build Job 1

Deploy to Bluemix

STAGE RUNNING...

LAST INPUT Stage: Empty Build Stage /...

Empty Build Job 1

JOBS View logs and history

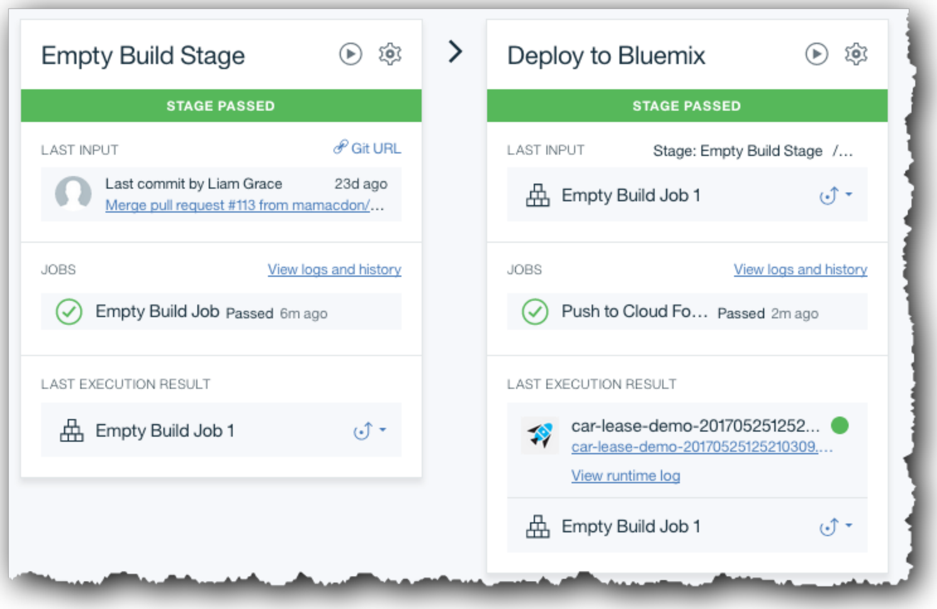
Push to Cloud Fo... Running

LAST EXECUTION RESULT

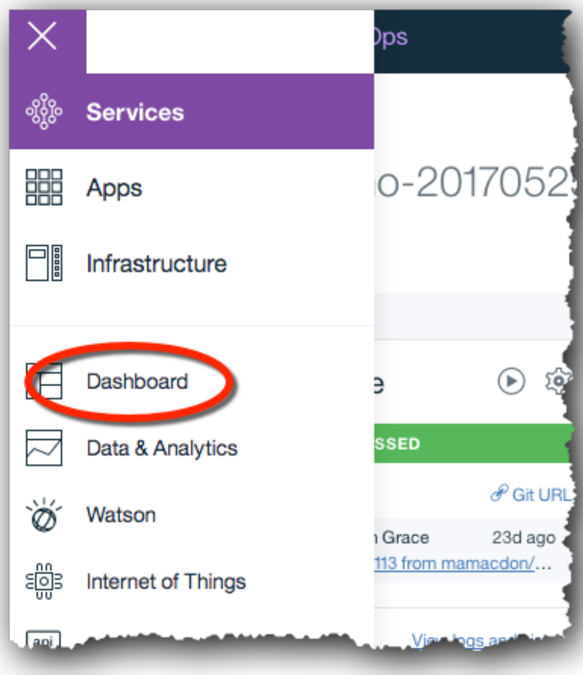
No results

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.

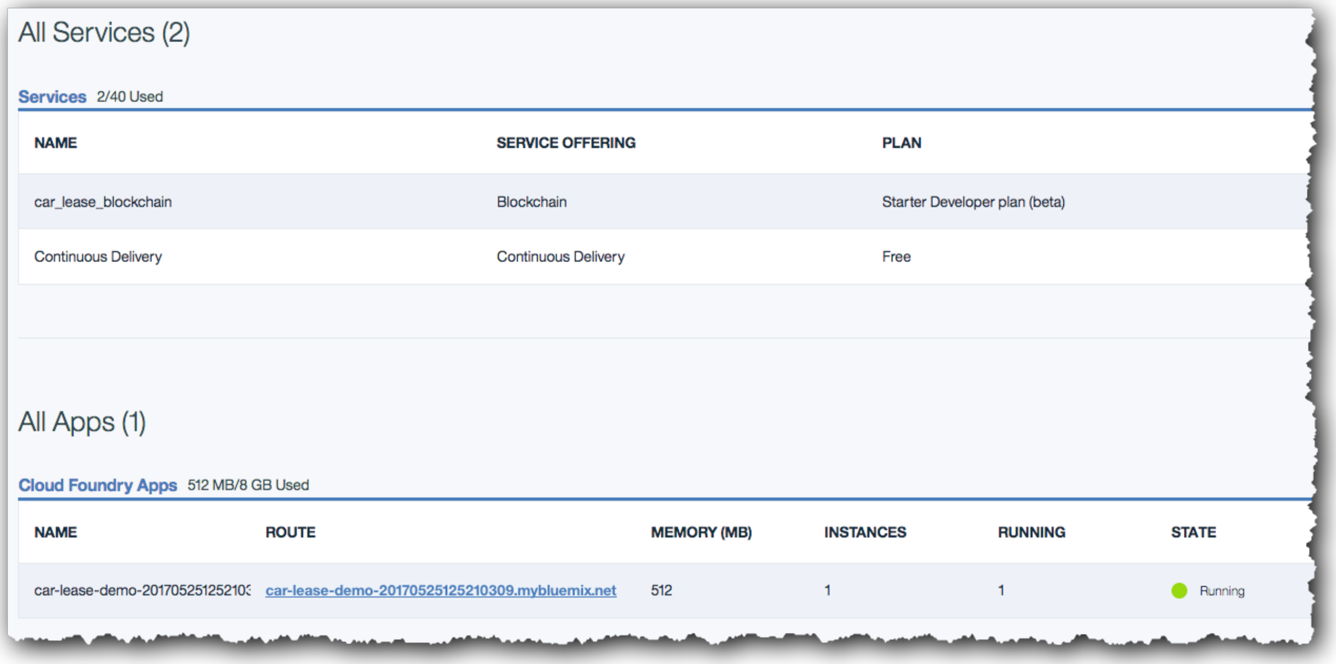


__15. Review the services and applications that have now been created and initialised.

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.



All Services (2)

Services 2/40 Used

NAME	SERVICE OFFERING	PLAN
car_lease_blockchain	Blockchain	Starter Developer plan (beta)
Continuous Delivery	Continuous Delivery	Free

All Apps (1)

Cloud Foundry Apps 512 MB/8 GB Used

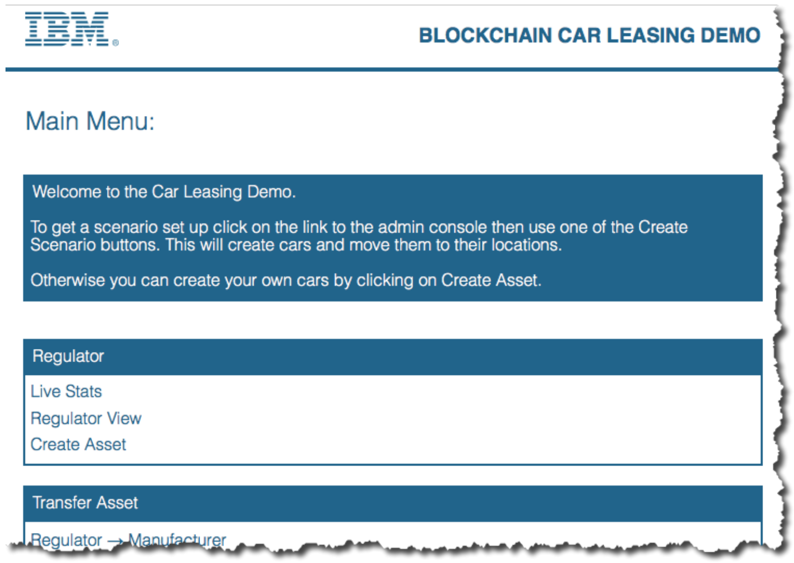
NAME	ROUTE	MEMORY (MB)	INSTANCES	RUNNING	STATE
car-lease-demo-20170525125210	car-lease-demo-20170525125210309.mybluemix.net	512	1	1	● Running

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.



BLOCKCHAIN CAR LEASING DEMO

Main Menu:

Welcome to the Car Leasing Demo.

To get a scenario set up click on the link to the admin console then use one of the Create Scenario buttons. This will create cars and move them to their locations.

Otherwise you can create your own cars by clicking on Create Asset.

Regulator

Live Stats

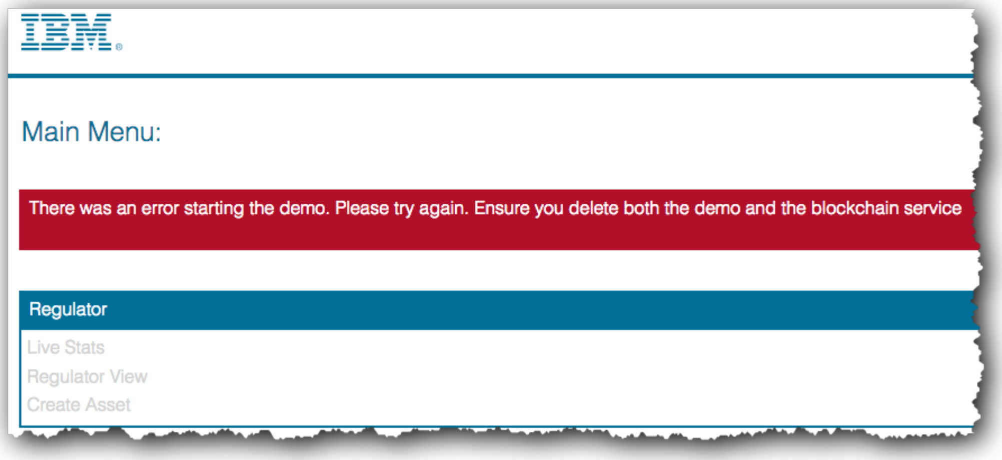
Regulator View

Create Asset

Transfer Asset

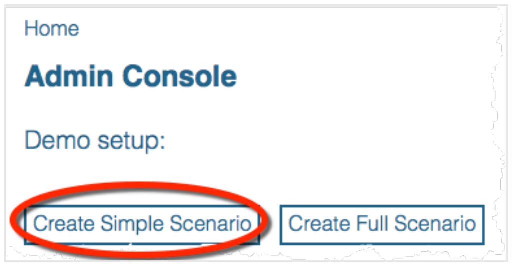
Regulator → Manufacturer

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



There was an error starting the demo. Please try again. Ensure you delete both the demo and the blockchain service

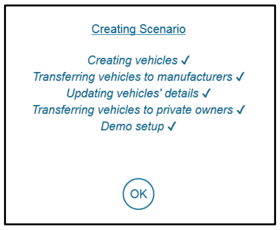
__17. From the Car Leasing demo front page, click '**Admin Console**' and '**Create Simple Scenario**'.



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.)

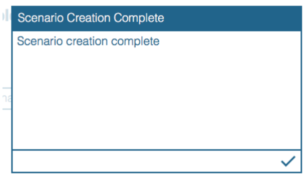
Wait for the initialization to complete.

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



Creating Scenario

Creating vehicles ✓
Transferring vehicles to manufacturers ✓
Updating vehicles' details ✓
Transferring vehicles to private owners ✓
Demo setup ✓



Scenario Creation Complete

Scenario creation complete

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

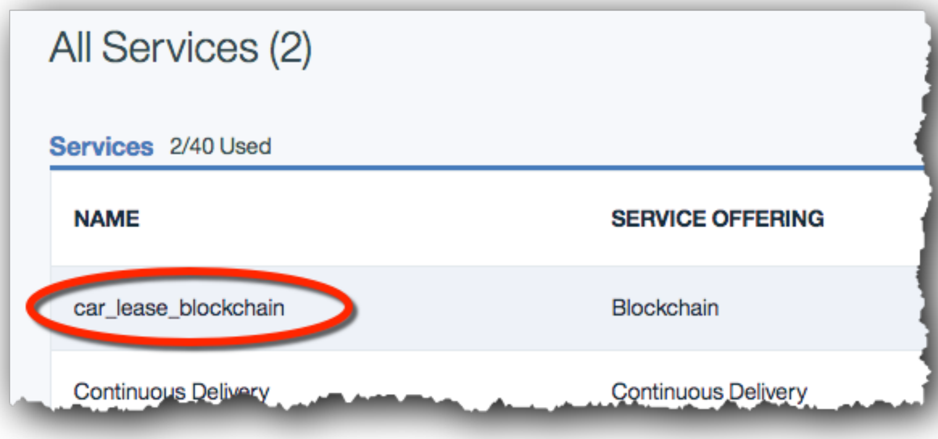
Section 2. Managing the sample application

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

2.1. Viewing the components of the Blockchain service

__20. 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.

__21. Click on the **car_lease_blockchain** service in the Services section of the dashboard.

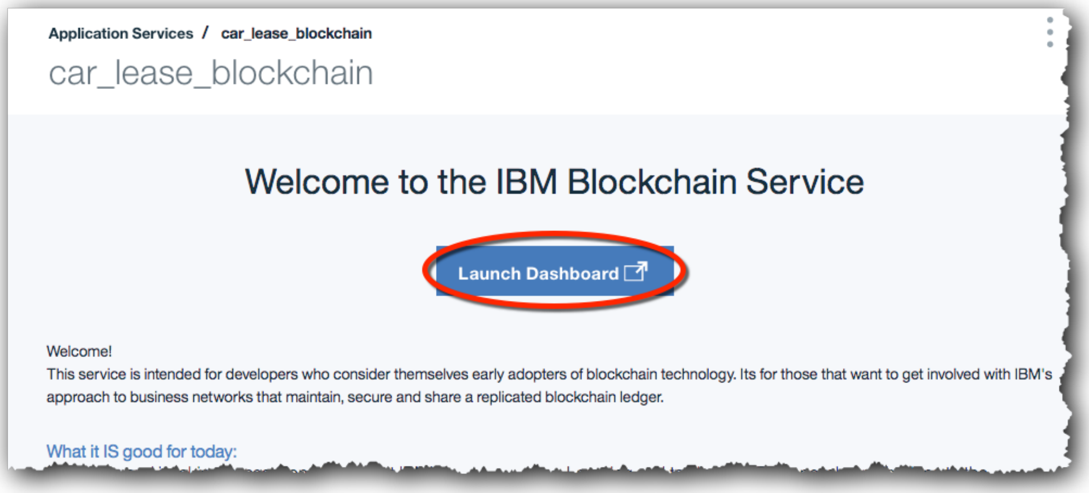


All Services (2)

Services 2/40 Used

NAME	SERVICE OFFERING
car_lease_blockchain	Blockchain
Continuous Delivery	Continuous Delivery

__22. Review the details and select 'Launch Dashboard' to launch the dashboard.



Application Services / car_lease_blockchain

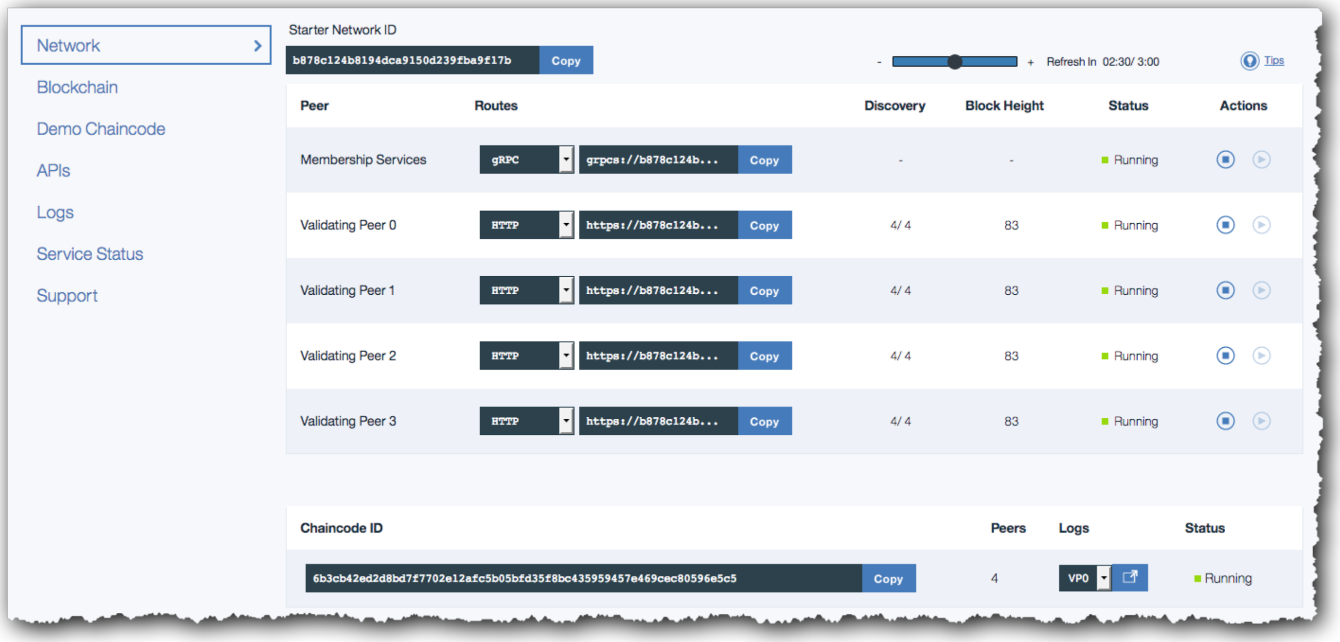
car_lease_blockchain

Welcome to the IBM Blockchain Service

Welcome!
This service is intended for developers who consider themselves early adopters of blockchain technology. Its for those that want to get involved with IBM's approach to business networks that maintain, secure and share a replicated blockchain ledger.

What it IS good for today:

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



Starter Network ID

b878c124b8194dca9150d239fba9f17b

Peer	Routes		Discovery	Block Height	Status
Membership Services	gRPC	grpcs://b878c124b...	-	-	🟩 Running
Validating Peer 0	HTTP	https://b878c124b...	4/ 4	83	🟩 Running
Validating Peer 1	HTTP	https://b878c124b...	4/ 4	83	🟩 Running
Validating Peer 2	HTTP	https://b878c124b...	4/ 4	83	🟩 Running
Validating Peer 3	HTTP	https://b878c124b...	4/ 4	83	🟩 Running

Chaincode ID	Peers	Logs	Status
6b3cb42ed2d8bd7f7702e12afc5b05bfd35f8bc435959457e469cec80596e5c5	4	VP0	🟩 Running

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.

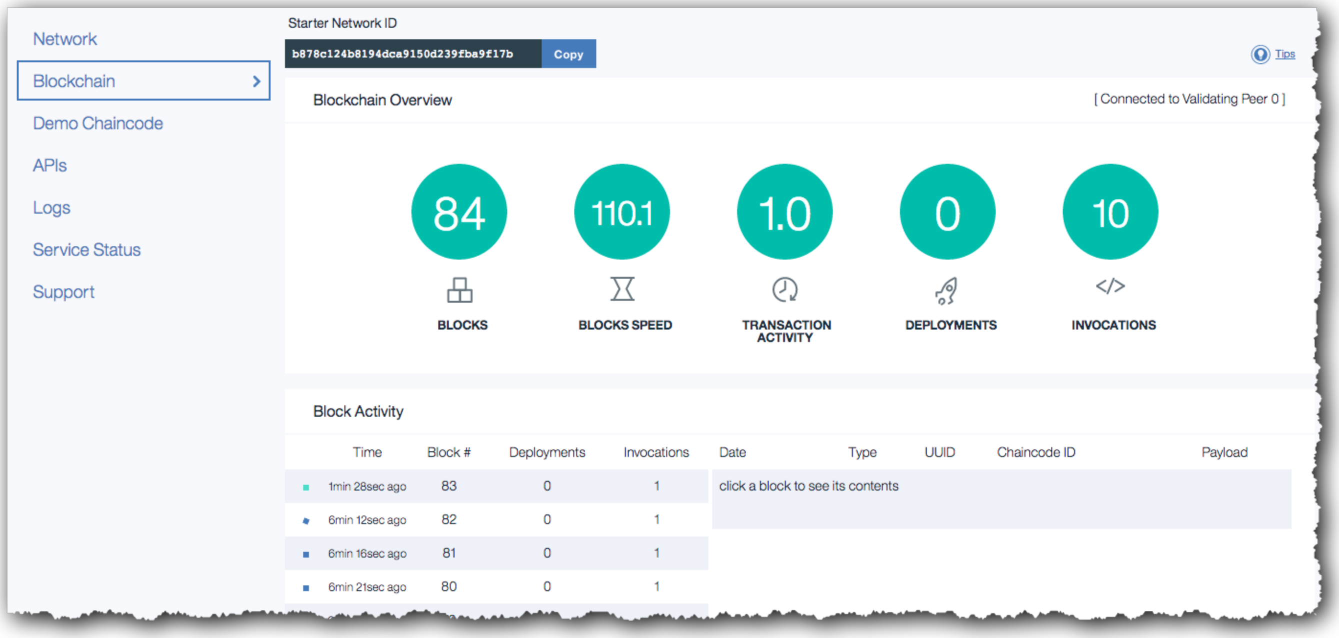
__23. Verify that the four validating peers each have the same block height.

__24. 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.)

2.2. Viewing the Blockchain

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

__25. Click the '**Blockchain**' tab at the left of the page.



Starter Network ID

`b878c124b8194dca9150d239fba9f17b`

Blockchain Overview

[Connected to Validating Peer 0]

BLOCKS	BLOCKS SPEED	TRANSACTION ACTIVITY	DEPLOYMENTS	INVOCATIONS
84	110.1	1.0	0	10

Block Activity

Time	Block #	Deployments	Invocations
1min 28sec ago	83	0	1
6min 12sec ago	82	0	1
6min 16sec ago	81	0	1
6min 21sec ago	80	0	1

Date	Type	UUID	Chaincode ID	Payload
click a block to see its contents				

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

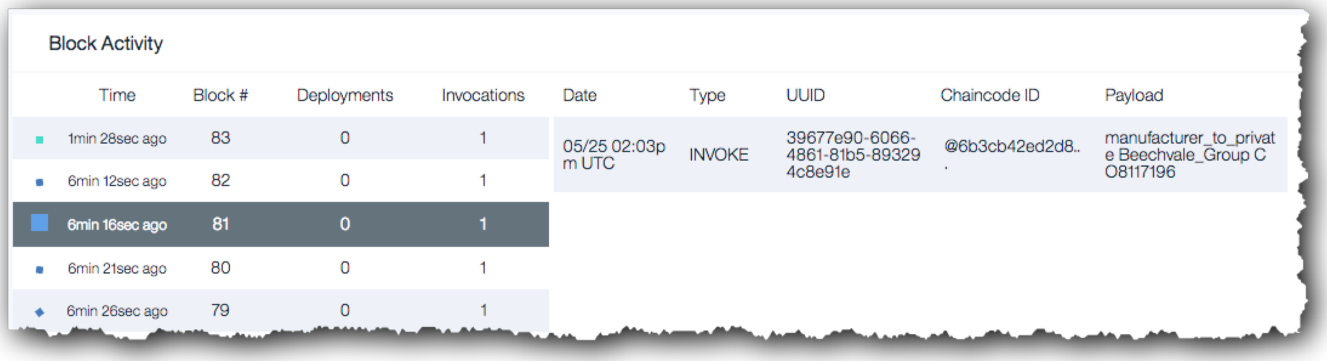
Each block contains a set of transactions. In Hyperledger Fabric V0.6, a transaction is the record of the request to interact with chaincode (a smart contract). The two most important transaction types are:

- *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.

__26. Click on a block that contains at least one invocation request.

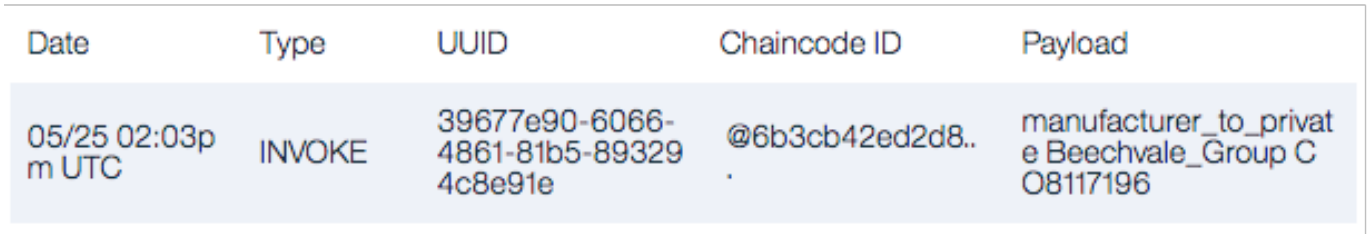


Block Activity

	Time	Block #	Deployments	Invocations
■	1min 28sec ago	83	0	1
■	6min 12sec ago	82	0	1
■	6min 16sec ago	81	0	1
■	6min 21sec ago	80	0	1
◆	6min 26sec ago	79	0	1

Date	Type	UUID	Chaincode ID	Payload
05/25 02:03pm UTC	INVOKE	39677e90-6066-4861-81b5-893294c8e91e	@6b3cb42ed2d8...	manufacturer_to_private Beechvale_Group C08117196

__27. 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 C08117196

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.

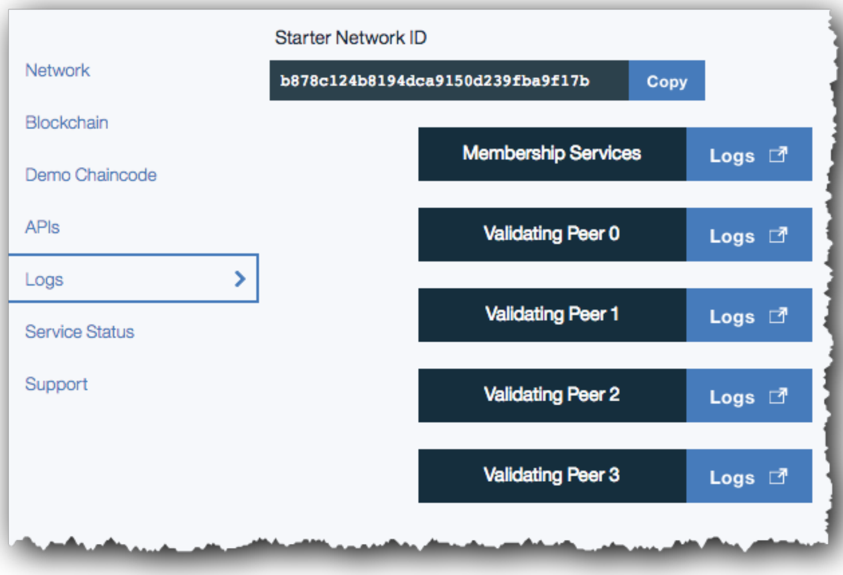
__28. Repeat this for other blocks to understand how the transactions are stored.

ℹ	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!

2.3. Understanding the Blockchain Peers

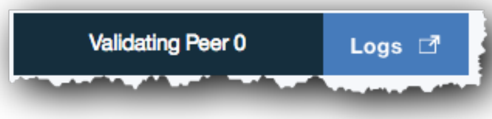
We are now going to review the logs associated with the peers. This is useful for understanding how the blockchain works, and for diagnosing problems.

__29. Click on the ‘**Logs**’ tab.

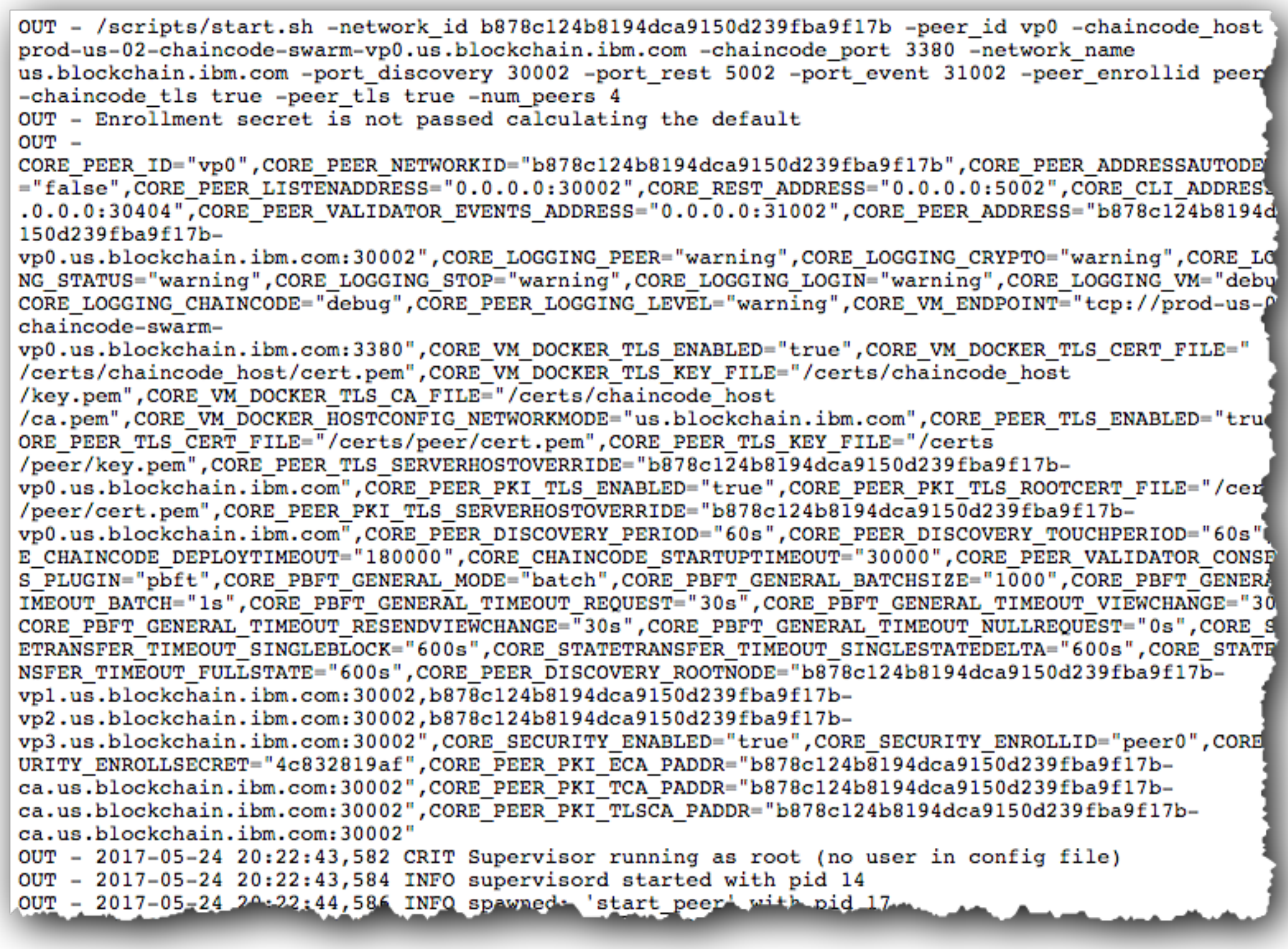


By looking at the logs for each peer you can verify that every node has executed every transaction.

__30. Click the Logs button against one of the validating peers.



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



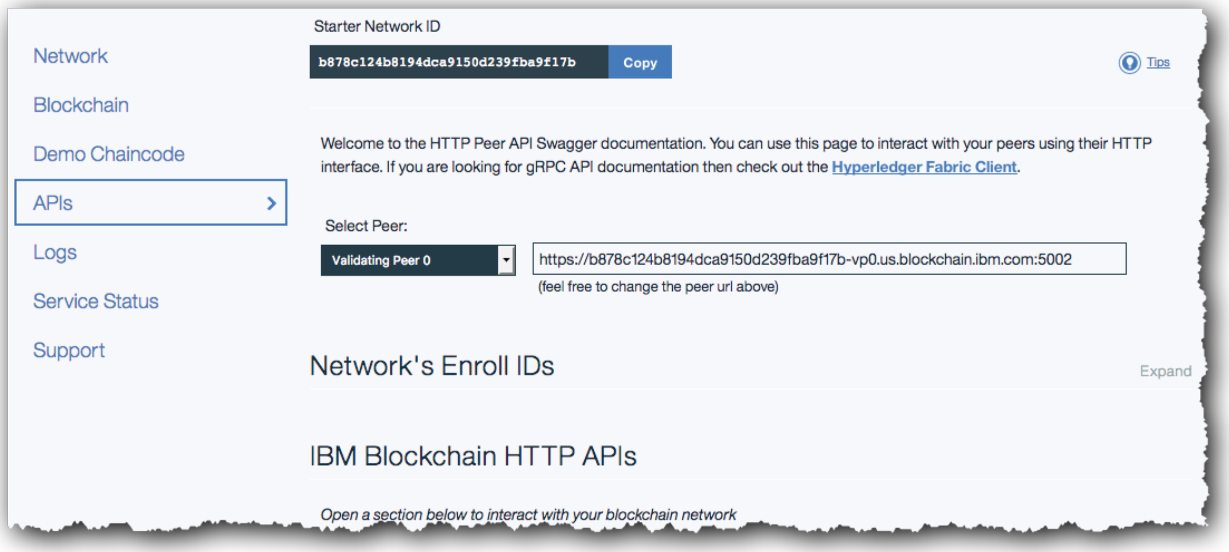
```
OUT - /scripts/start.sh -network_id b878c124b8194dca9150d239fba9f17b -peer_id vp0 -chaincode_host
prod-us-02-chaincode-swarm-vp0.us.blockchain.ibm.com -chaincode_port 3380 -network_name
us.blockchain.ibm.com -port_discovery 30002 -port_rest 5002 -port_event 31002 -peer_enrollid peer
-chaincode_tls 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-
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
```

2.4. 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!

__31. Click on the ‘**APIs**’ tab on the dashboard.



Starter Network ID

b878c124b8194dca9150d239fba9f17b

Welcome to the HTTP Peer API Swagger documentation. You can use this page to interact with your peers using their HTTP interface. If you are looking for gRPC API documentation then check out the Hyperledger Fabric Client.

Select Peer: Validating Peer 0

https://b878c124b8194dca9150d239fba9f17b-vp0.us.blockchain.ibm.com:5002

(feel free to change the peer url above)

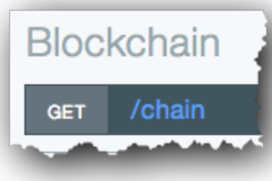
Network's Enroll IDs

IBM Blockchain HTTP APIs

Open a section below to interact with your blockchain network

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).

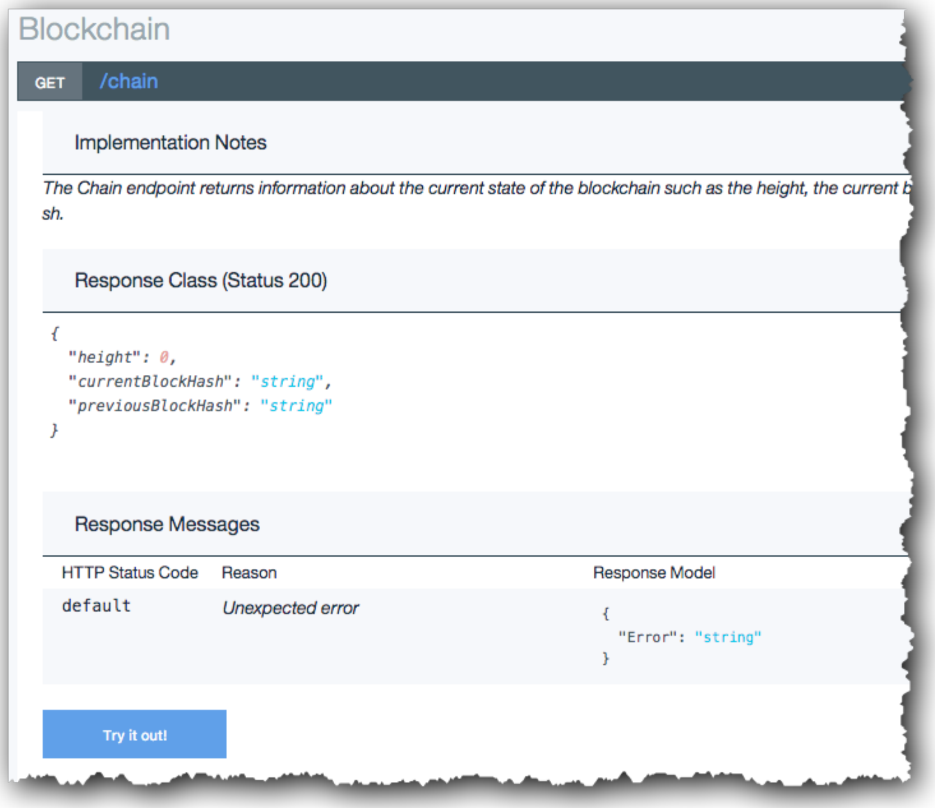
__32. Click the ‘**Blockchain**’ section.



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

__33. Select the operation to view information about it.

This reveals the input and output data formats.



Blockchain

GET /chain

Implementation Notes

The Chain endpoint returns information about the current state of the blockchain such as the height, the current b
sh.

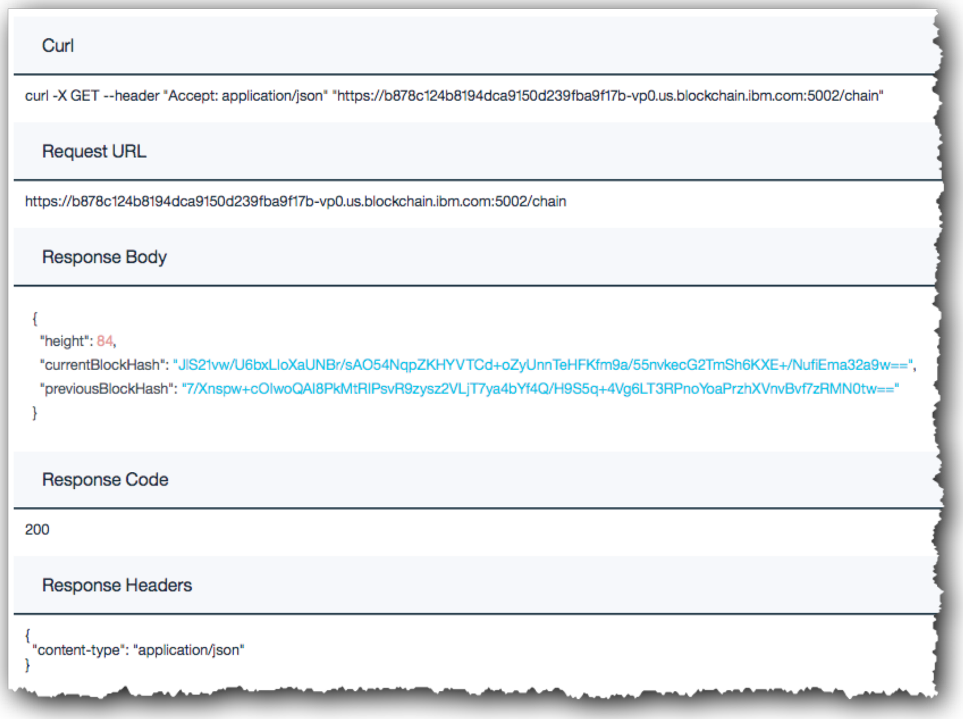
Response Class (Status 200)

```
{
  "height": 0,
  "currentBlockHash": "string",
  "previousBlockHash": "string"
}
```

Response Messages

HTTP Status Code	Reason	Response Model
default	*Unexpected error*	{ "Error": "string" }

__34. Click '**Try It Out**' to invoke the API.



Curl

```
curl -X GET --header "Accept: application/json" "https://b878c124b8194dca9150d239fba9f17b-vp0.us.blockchain.ibm.com:5002/chain"
```

Request URL

```
https://b878c124b8194dca9150d239fba9f17b-vp0.us.blockchain.ibm.com:5002/chain
```

Response Body

```
{
  "height": 84,
  "currentBlockHash": "JlS21vw/U6bxLloXaUNBr/sAO54NqpZKHYVTCd+oZyUnnTeHFKfm9a/55nvkecG2TmSh6KXE+/NufiEma32a9w==",
  "previousBlockHash": "7/Xnspw+cOlwoQAl8PkMtRIPsvR9zysz2VLjT7ya4bYf4Q/H9S5q+4Vg6LT3RPnoYoaPrzhXVnvBvf7zRMN0tw=="
}
```

Response Code

```
200
```

Response Headers

```
{
 "content-type": "application/json"
}
```

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.

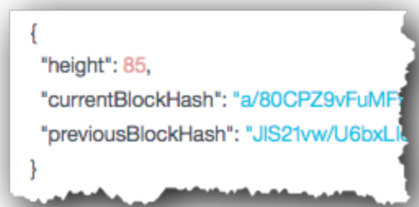
A hash function is simply a function that is applied to a data set that produces a consistent output. It is usually used to map data of an arbitrary size to data of a fixed size. Importantly for blockchain, any change to the input data set will produce a different hash output, which can be used to easily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

__35. Note the first few characters of the value of the *currentBlockHash* ("JIS21…" in the previous screenshot).

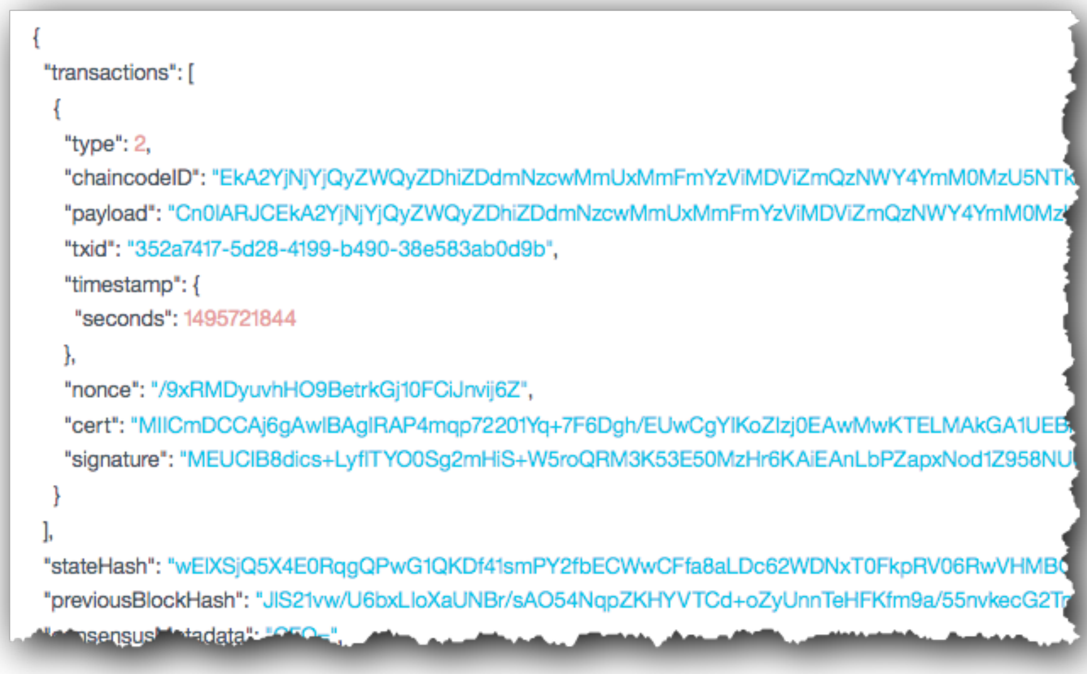
__36. Invoke another transaction in the car leasing demo to force another block to be created.

__37. 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…").



```
{
  "height": 85,
  "currentBlockHash": "a/80CPZ9vFuMF
  "previousBlockHash": "JIS21vw/U6bxLI
}
```

__38. 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+LyflTYO0Sg2mHiS+W5roQRM3K53E50MzHr6KAiEAnLbPZapxNod1Z958NU
  }
 ],
 "stateHash": "wEIXSjQ5X4E0RqgQPwG1QKDf41smPY2fbECWwCFfa8aLDc62WDNxT0FkpRV06RwVHMB
 "previousBlockHash": "JIS21vw/U6bxLloXaUNBr/sAO54NqpZKHYVTCd+oZyUnnTeHFKfm9a/55nvkecG2T
 "consensusMetadata": "CEQ=",
```

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,

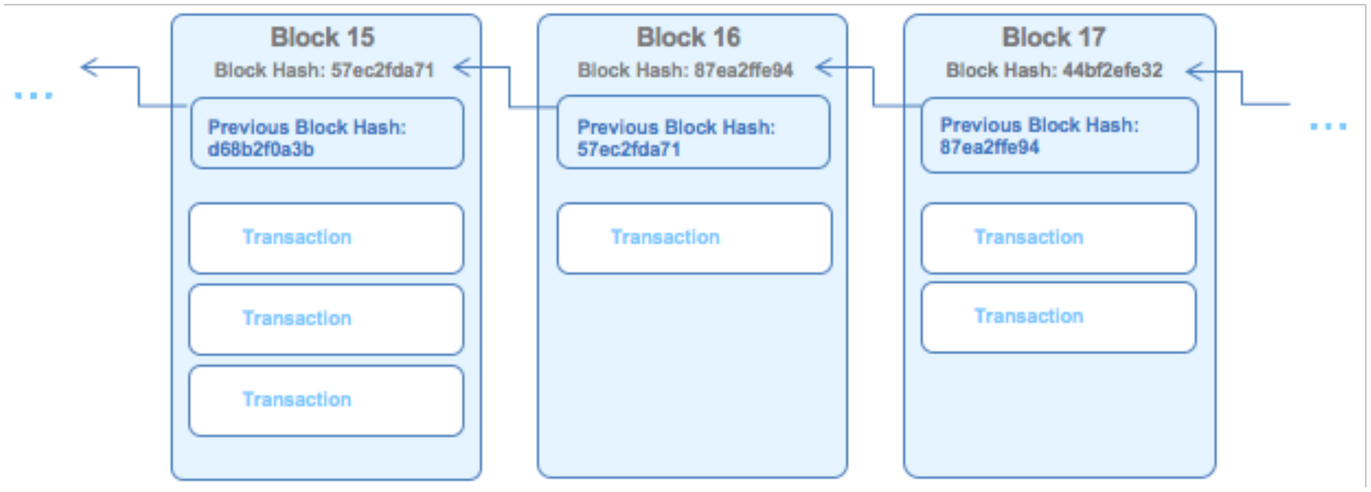
(c) a *previousBlockHash*, which is the result of running a hash function of the previous block in the chain, and

(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.

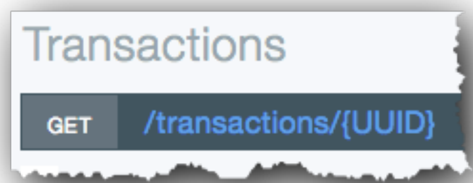
Particularly, note that the *previousBlockHash* element is *itself* used to calculate the hash of subsequent blocks. This means that even a small change to one of the first blocks in the chain can be detected in any future block.



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

__39. Copy the *txid* field of a transaction from a block; this will be a unique identifier of the form "04421f7d-652a-491d-90b0-7bc9f29b2d85".

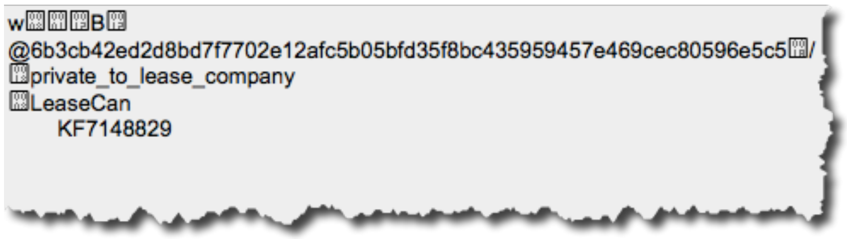
__40. Click the '**Transactions**' section.



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

__41. 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:



```
w B
@6b3cb42ed2d8bd7f7702e12afc5b05bfd35f8bc435959457e469cec80596e5c5 /
private_to_lease_company
LeaseCan
    KF7148829
```

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

2.5. Viewing the Service Status, Support Contacts and Samples

__42. 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.

__43. 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.

__44. 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 3. 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.

3.1. Transaction Signing

__45. In the blockchain dashboard, click 'APIs'.

__46. 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.

__47. Review the ‘cert’ and ‘signature’ elements of an individual transaction.

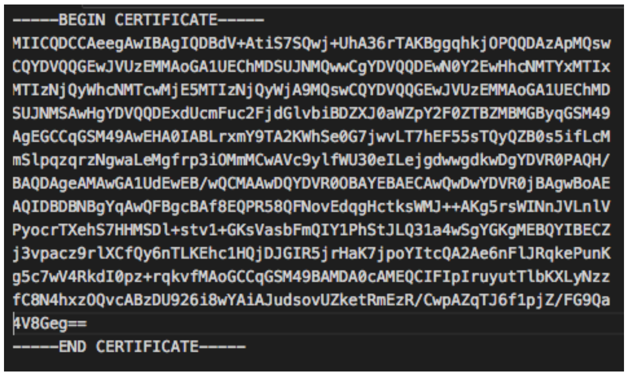
```
"cert":
"MIICQDCCAeegAwIBAgIQDBdV+AtiS7SQwj+UhA36rTAKBggqhkjOPQQDAzApMQswCQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMQwwCgYDVQQDEwN0
Y2EwHhcNMTYxMTIxMTIzNjQyWhcNMTcwMjE5MTIzNjQyWjA9MQswCQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMSAwHgYDVQQDExdUcmFuc2FjdGlvb
iBDZXJ0aWZpY2F0ZTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABLrxmY9TA2KWhSe0G7jwvLT7hEF55sTQyQZB0s5ifLcMmSlpqzqrzNgwaLeMgf
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.

__48. [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:

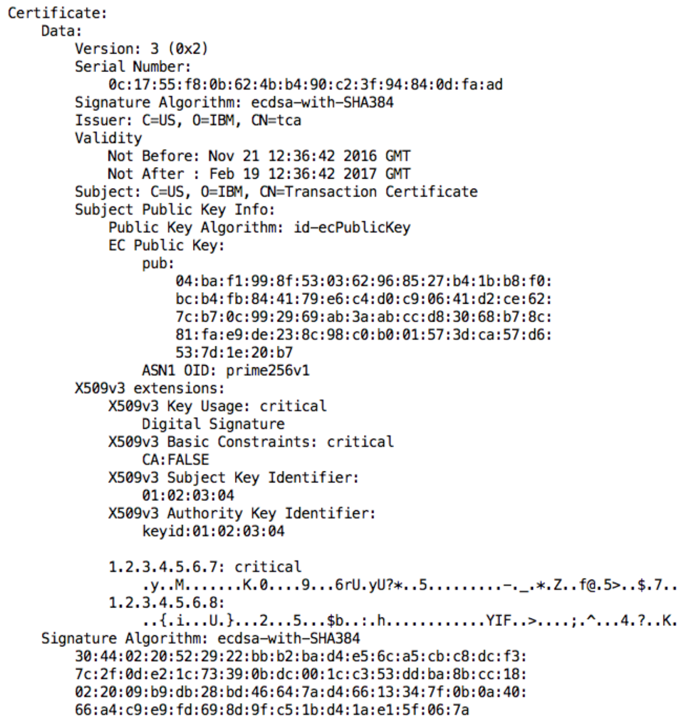


```
-----BEGIN CERTIFICATE-----
MIICQDCCAeegAwIBAgIQDBdV+AtiS7SQwj+UhA36rTAKBggqhkjOPQQDAzApMQsw
CQYDVQQGEwJVUzEMMAoGA1UEChMDSUJNMQwwCgYDVQQDEwN0Y2EwHhcNMTYxMTIx
MTIzNjQyWhcNMTcwMjE5MTIzNjQyWjA9MQswCQYDVQQGEwJVUzEMMAoGA1UEChMD
SUJNMSAwHgYDVQQDExdUcmFuc2FjdGlvbiBDZXJ0aWZpY2F0ZTBZMBMGByqGSM49
AgEGCCqGSM49AwEHA0IABLrxmY9TA2KWhSe0G7jwvLT7hEF55sTQyQZB0s5ifLcM
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

3.2. 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*.

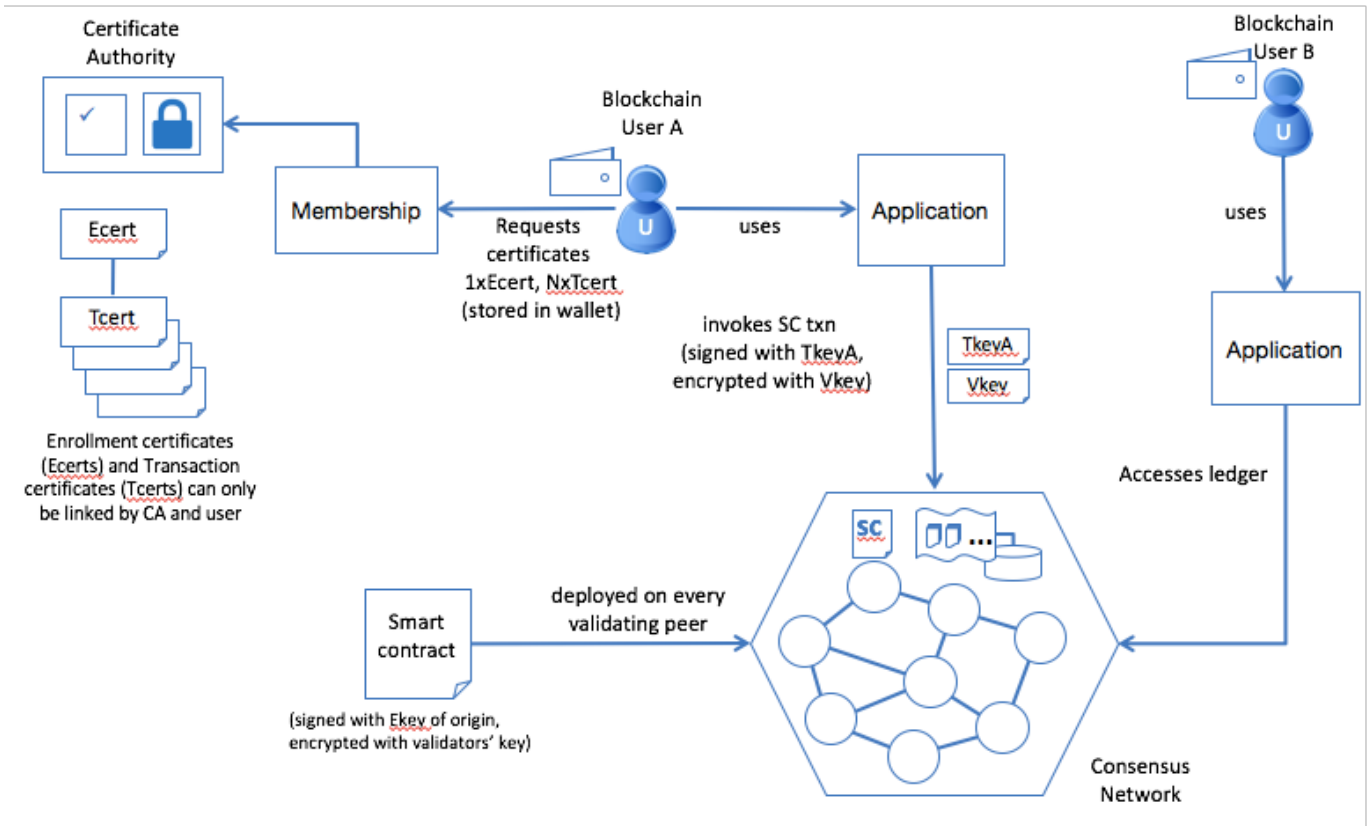
Let's assume Alice, Bob and Charlie form part of a business network and share a blockchain for transactions. Alice is dealing with Bob, and she is also dealing with Charlie. Crucially, Alice should never know that Bob and Charlie have done business together, as this knowledge could violate confidentiality 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.


Certificate Authority
Ecert
Tcert
Enrollment certificates (Ecerts) and Transaction certificates (Tcerts) can only be linked by CA and user
Membership
Blockchain User A
Requests certificates 1xEcert, NxTcert (stored in wallet)
U
uses
Application
invokes SC txn (signed with TkeyA, encrypted with Vkey)
TkeyA
Vkey
Blockchain User B
U
uses
Application
Accesses ledger
Smart contract
(signed with Ekey of origin, encrypted with validators' key)
deployed on every validating peer
SC
...
Consensus Network

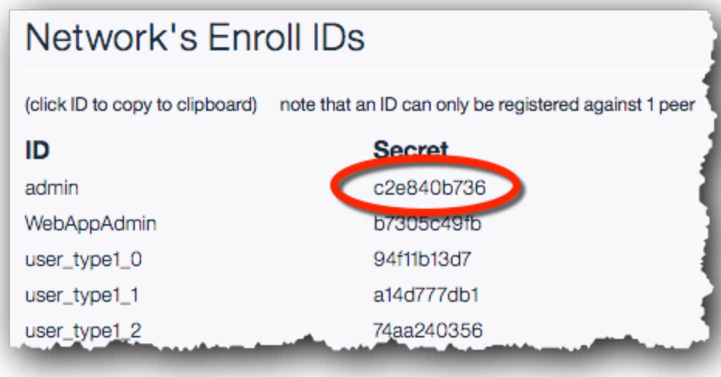
We will now use the membership services component to enroll on the blockchain network, provide us our enrollment certificate and issue some new sample transaction certificates.

3.3. Using Membership Services

__49. 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.

__50. Click the secret next to the 'admin' user to copy it to the clipboard.



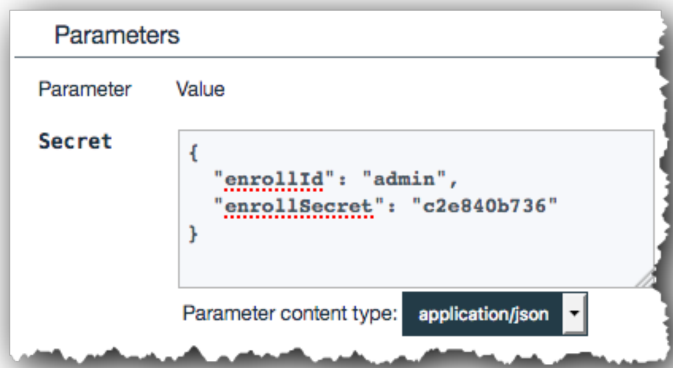
Network's Enroll IDs

(click ID to copy to clipboard) note that an ID can only be registered against 1 peer

ID	Secret
admin	c2e840b736
WebAppAdmin	b7305c49fb
user_type1_0	94f11b13d7
user_type1_1	a14d777db1
user_type1_2	74aa240356

__51. In the Registrar section, click the POST /registrar operation.

__52. Fill in the JSON structure for the ‘Secret’ parameter, as follows. The enrollSecret parameter should match the secret you just copied to the clipboard.



Parameters

Parameter Value

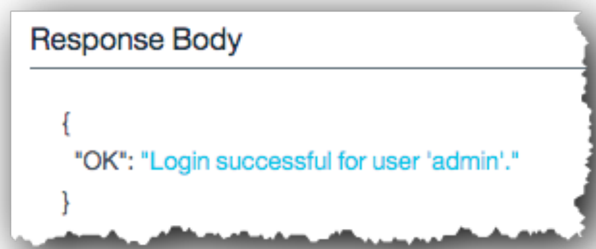
Secret

```
{
  "enrollId": "admin",
  "enrollSecret": "c2e840b736"
}
```

Parameter content type: application/json

__53. Click ‘Try it out!’.

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

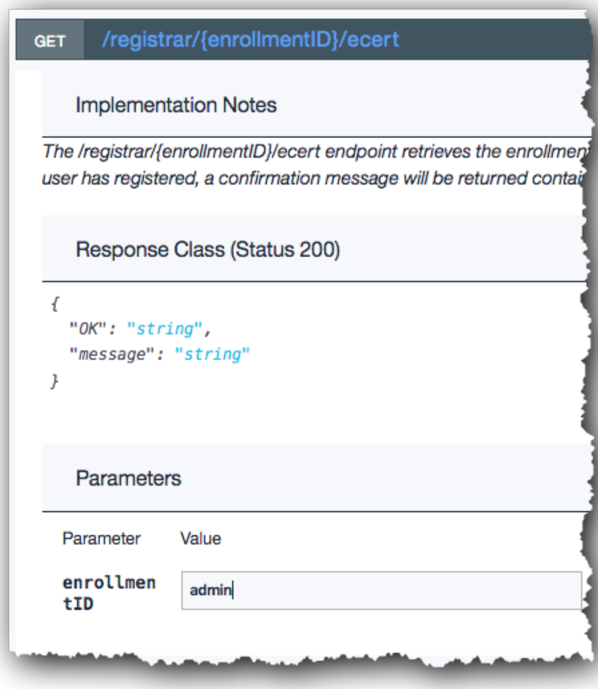


Response Body

```
{
  "OK": "Login successful for user 'admin'."
}
```

__54. Select the ‘GET /registrar/{enrollmentID}/ecert’ operation.

__55. Enter ‘admin’ as the enrollmentID.



GET /registrar/{enrollmentID}/ecert

Implementation Notes

The /registrar/{enrollmentID}/ecert endpoint retrieves the enrollmen
user has registered, a confirmation message will be returned contai

Response Class (Status 200)

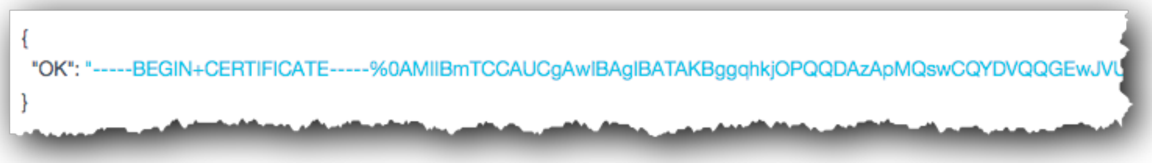
```
{
  "OK": "string",
  "message": "string"
}
```

Parameters

Parameter	Value
enrollmentID	admin

__56. Click ‘Try it out!’.

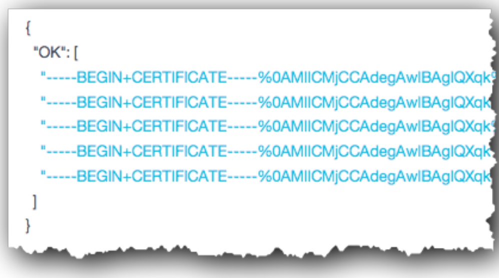
The enrollment certificate for the user will be returned.



```
{
  "OK": "-----BEGIN+CERTIFICATE-----%0AMIIBmTCCAUCgAwIBAgIBATAKBggqhkjOPQQDAzApMQswCQYDVQQGEwJVU
}
```

__57. 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.



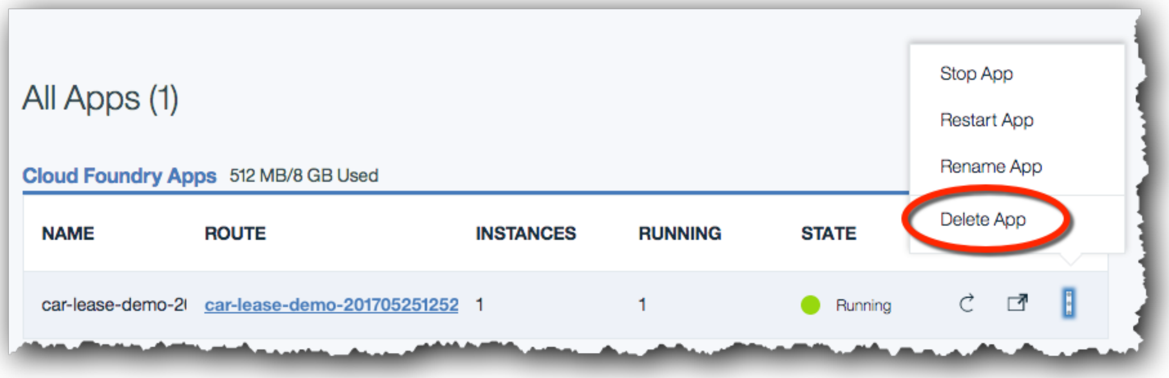
```
{
  "OK": [
    "-----BEGIN+CERTIFICATE-----%0AMIICMjCCAdegAwIBAgIQXqk
    "-----BEGIN+CERTIFICATE-----%0AMIICMjCCAdegAwIBAgIQXqk
    "-----BEGIN+CERTIFICATE-----%0AMIICMjCCAdegAwIBAgIQXqk
    "-----BEGIN+CERTIFICATE-----%0AMIICMjCCAdegAwIBAgIQXqk
    "-----BEGIN+CERTIFICATE-----%0AMIICMjCCAdegAwIBAgIQXqk
  ]
}
```

Section 4. Removing the sample application

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

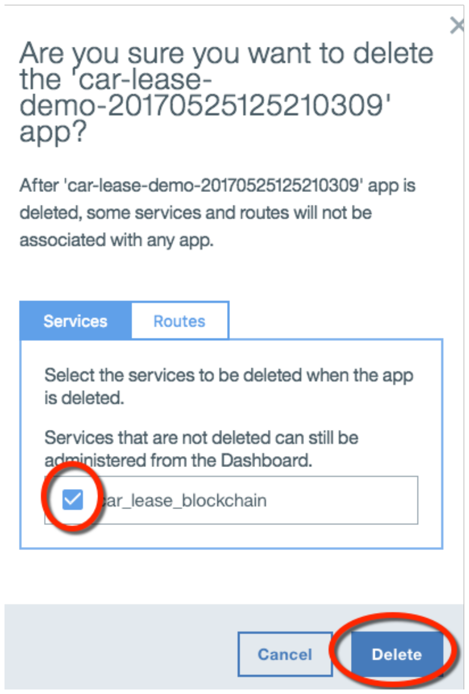
__58. Return to the Bluemix Dashboard (https://console.ng.bluemix.net/dashboard/applications).

__59. 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.



NAME	ROUTE	INSTANCES	RUNNING	STATE
car-lease-demo-2(	car-lease-demo-201705251252	1	1	Running

__60. Ensure that the 'car_lease_blockchain' service is selected for deletion and click '**Delete**'.



Are you sure you want to delete the 'car-lease-demo-20170525125210309' app?

After 'car-lease-demo-20170525125210309' app is deleted, some services and routes will not be associated with any app.

Select the services to be deleted when the app is deleted.

Services that are not deleted can still be administered from the Dashboard.

[✓] car_lease_blockchain

__61. 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.

<u>Congratulations on completing Lab two – “Blockchain Explored”!</u>

Appendix A. Notices

Appendix B. Trademarks and copyrights

NOTES

NOTES

IBM Software