



Lab: Explore the IBM Blockchain service on IBM Bluemix

Overview

In this lab, you use the IBM® Blockchain service on IBM® Bluemix to build on the car leasing demo that was introduced in the previous lab, “Transfer assets in a business network.”

If you completed the previous lab, you have already deployed the car leasing application to your account, which means you can skip Step 1 and reuse your existing application.

Tip: This lab uses the IBM Bluemix interface in the Classic Experience view. If you log in to Bluemix and want to work in the Classic Experience UI, click **Go to Classic Experience** at the top of the page.

Prerequisites

It's recommended that you use Firefox or Chrome web browsers.


You will need a [Bluemix account](#) to create the sample application.

Step 1. Deploy and configure the sample application

To deploy the sample application:

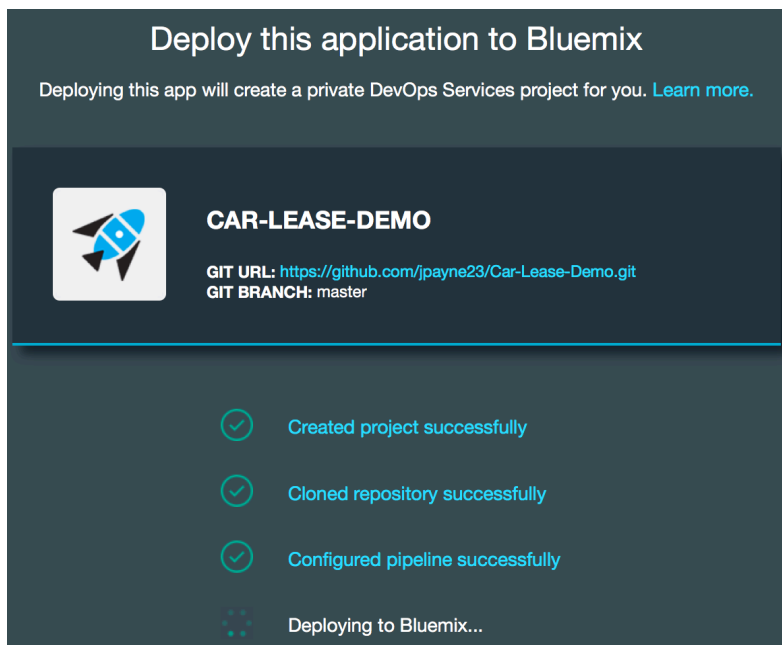
1. Open your browser and go to <http://www.ibm.com>.
2. Click **Sign up** or **Log in** to create a new Bluemix account or log into your existing account.
3. After you have successfully signed up and logged into Bluemix, click **CATALOG** from the top bar.
4. Scroll down to the **Network** section and click **Blockchain**. Review the service description and information about the service.
5. Click **View Docs** to learn about the process of creating a blockchain environment.
6. Expand **Sample Apps and Tutorials** on the right side of the page to view the available apps.
7. Select the **Using Car Lease Demo** item from the list of apps.
8. Click **Deploy to Bluemix** after the Car-Lease-Demo overview paragraph. You might need to log into Bluemix again.

If this is your first time using Bluemix DevOps services, you will be prompted to create an alias for the DevOps Services Git repository that will link to your IBM ID. This could be the first part of your email address (add a number afterward if needed to make it unique). Click **Create** after providing the alias.



The screenshot shows a dialog box titled "IBM Bluemix™ DevOps Services" with the heading "Pick an alias". Below the heading, it says: "To set up your Git repository, we need to associate your IBM id with an alias." and "An alias is a unique, publicly visible short name used in Git repository paths, Track & Plan, and desktop and command line clients." There is a text input field with a speech bubble icon and the placeholder text "Pick an alias". Below the input field is a checkbox labeled "I accept the DevOps Services [Terms of Use](#).". At the bottom is a blue button labeled "Create".

You can leave the App Name, Region, Organization, and Space attributes in the default state and click **Deploy**. It will take a few seconds for the default field values to be populated. This action will cause the Car-Lease-Demo to be deployed to your Bluemix environment and might take a couple of minutes to complete.



When you see the “Success!” message, click **DASHBOARD** to see the new car leasing application and associated blockchain service that you created.



9. Click the application icon in the dashboard.

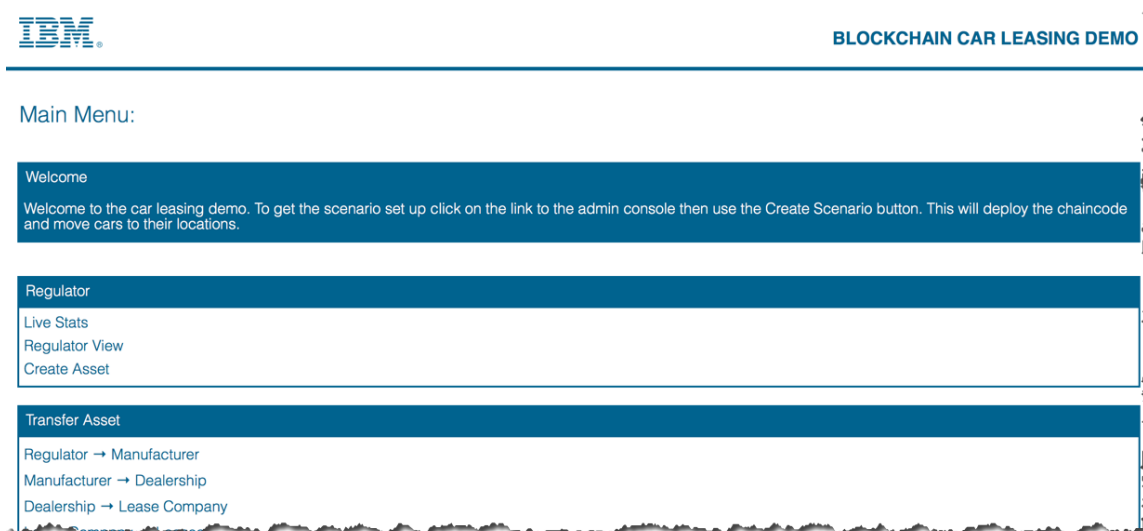


Your application icon might be different from the sample in the demo. Clicking the icon will show you information about the application, including memory consumption and the activity log.

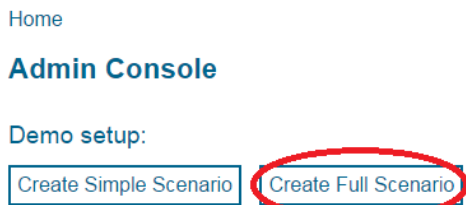
10. To run the scenario, click the link displayed on the route for the application, for example:

Routes: [Car-Lease-Demo-mqmatt-147.mybluemix...](#)

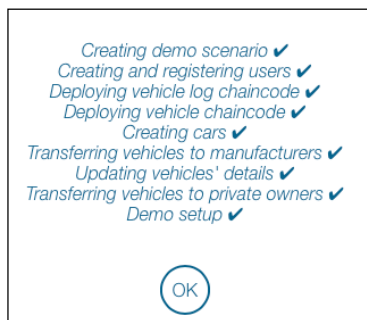
This will load a webpage that is served from the application.



11. Click **Admin Console > Create Full Scenario** to load the initial set of assets into the blockchain. This will take several minutes to complete.



The scenario setup is complete when "Demo setup" is displayed.




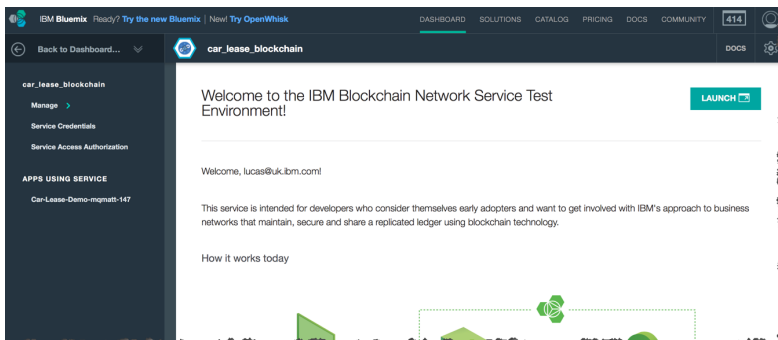
If an error occurs when creating the scenario, read "Remove the sample application" at the end of this document for instructions about how to delete the service.

Step 2. Manage the sample application

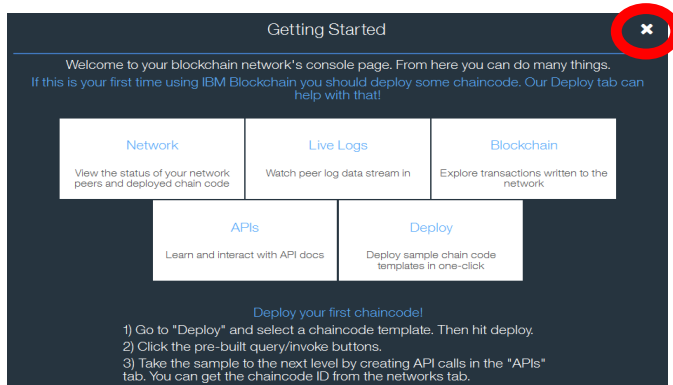
In this section, use the tools available in the IBM Bluemix environment to view and manage the blockchain.

2.1 View the components of the IBM Blockchain service

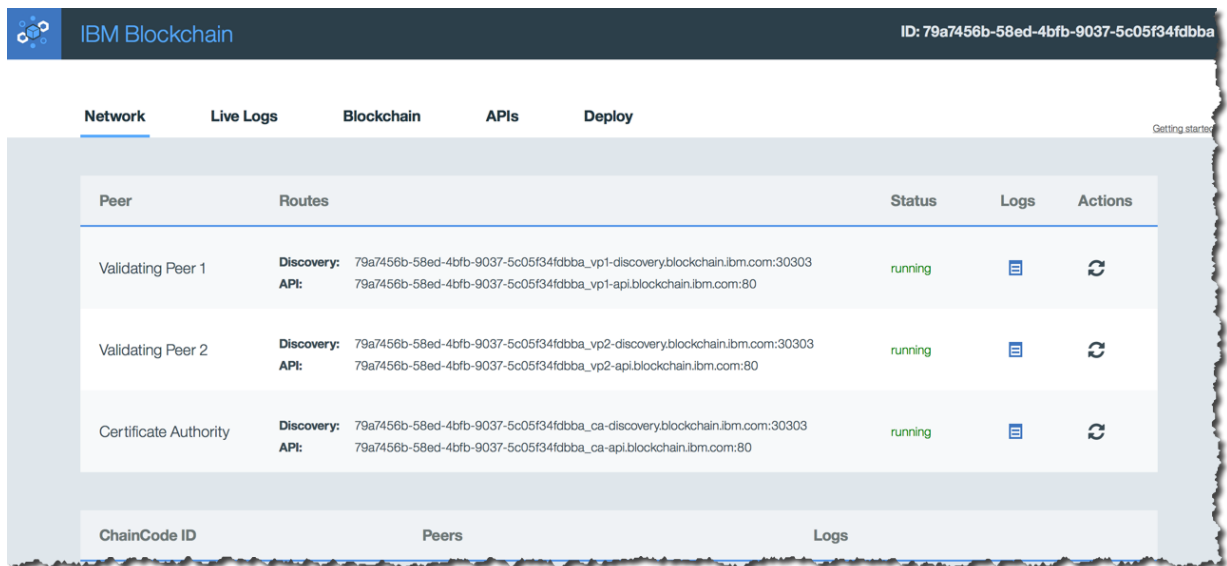
1. In Bluemix, click **DASHBOARD** to view the Car Leasing application.
2. Click the service icon for your new IBM Blockchain service in the Dashboard . This will take you to the service welcome page.



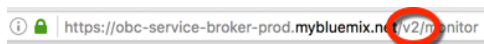
3. Review the details and click **LAUNCH** to start the service console.
4. Close the window that shows information about the sections. You will be able to look at these in more detail throughout this lab.



After closing the window, you will see the monitor page with the Network tab selected.

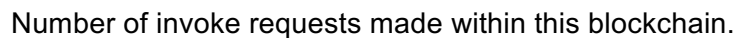
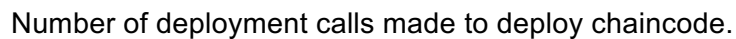
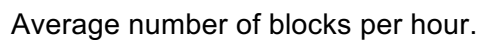
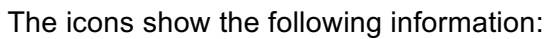


- If you see fewer than five tabs on this page, change the version in the URL to “v2” and reload the page.



This view confirms that two validating peers and a certificate authority are running under the service that you created.

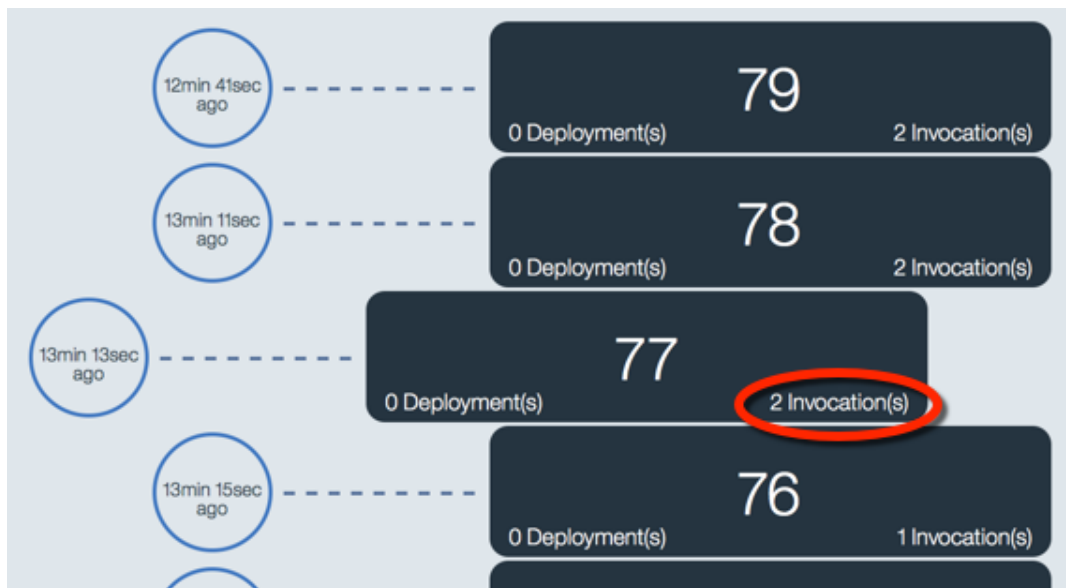
1. Click the **Blockchain** tab at the top of the page.



Each block contains a set of transactions. In Hyperledger Fabric, a transaction is the record of the request to interact with chaincode, a smart contract. Two important transaction types are:

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

Other request types exist, such as QUERY, UPDATE, and TERMINATE. Not all request types are recorded on the blockchain.



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

2. Click a block that contains at least one invocation request.
3. Look through the list of transactions that are contained in the block.

DATE	TYPE	UUID	CHAINCODE ID	PAYLOAD
05/26 01:49pm UTC	INVOKE	47897ad9-4356-4b3c-9405-38856383a5ee	9c48cd2b3...	create_vehicle_logTransferELease(an & Joe Payne&{720965981630055}1 oyota Yaris, Red, QD65 YKR DP88811 3826/5/2016 13:49:04LeaseCan Joe i ayne

Each line of information is a transaction stored in the block. A block can contain multiple transactions, but in this demo, there will often be only one transaction per block because of the low frequency of transactions being made. The information being provided is:

- **Date:** The date the transaction was submitted.
- **Type:** The type of transaction taking place, such as 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 of the chaincode.

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

When the blockchain is initialized for the car leasing application, the first two blocks in the chain usually contain DEPLOY transactions by which the chaincode is deployed to the validating peers.

You can view these blocks if you're willing to scroll down the blockchain explorer that far!

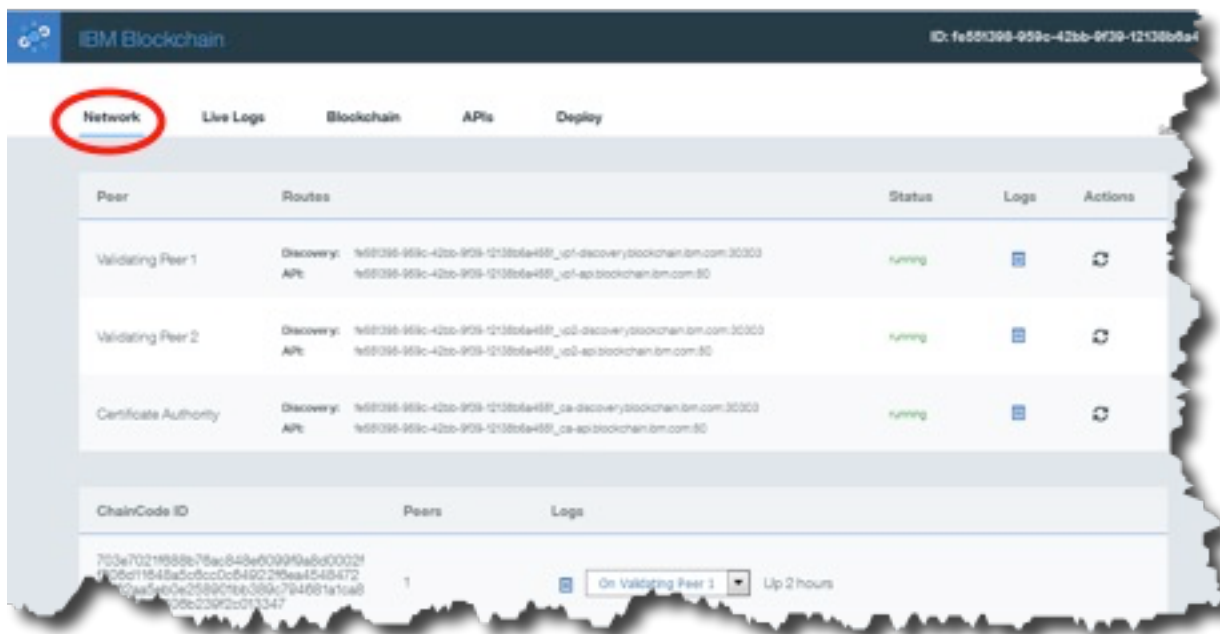
2.3 Understand the blockchain peers

Now, you will review the logs associated with the peers. This is useful for understanding how the blockchain works and for diagnosing problems.

There are two ways of accessing the logs of the peers:

- The **Logs** button under the **Network** tab is useful for downloading log files from the peers for offline analysis.
- The **Live Logs** tab shows you what the peers are doing currently.

1. Click the **Network** tab on the service page.

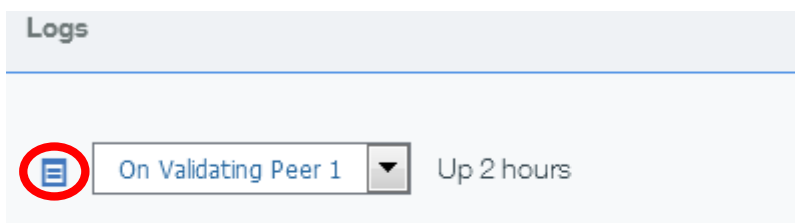


Here you can see that this blockchain network contains two validating peers and a Certificate Authority. The table underneath shows that there are two chaincode applications deployed to this network.

Requests to invoke chaincode, including the method name and any input parameters, are replicated onto every validating node and when a block is created, every validating node will execute the chaincode independently. The validating peers will then attempt to achieve consensus over any changes proposed to the world state as a result of running this chaincode and as a consequence, will persist or discard the changes.

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

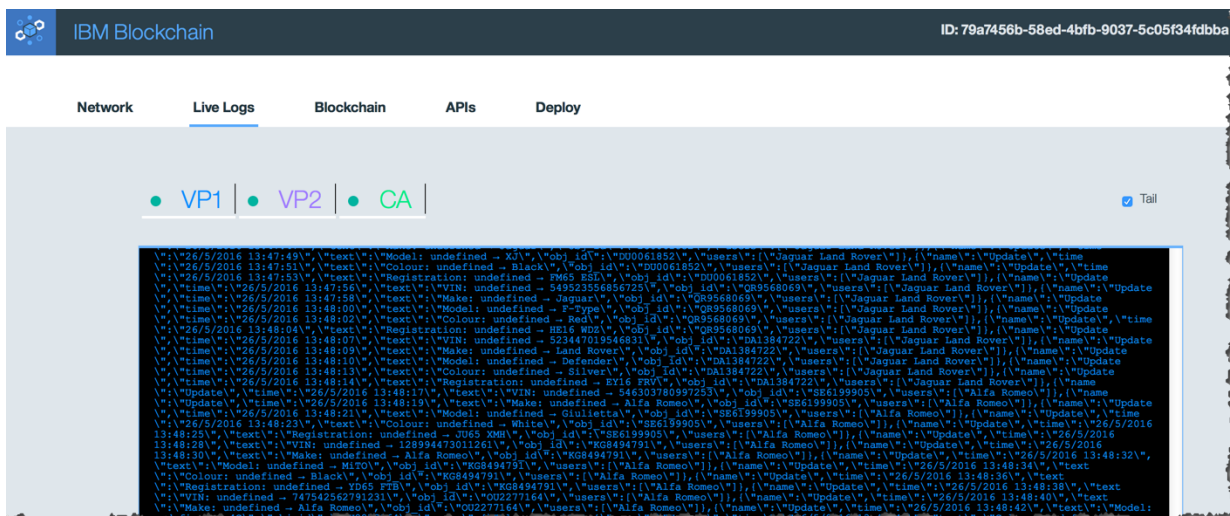
- Choose a chaincode ID from the table and click the log icon for the peer whose logs you want to examine.



A new tab opens with the logs for the selected peer.

- After you view the messages, close the browser tab.

```
ERR - 2016/05/26 13:46:28 Peer address: 79a7456b-58ed-4bfb-9037-5c05f34fdbba_vpl-discovery.blockchain.ibm
ERR - 2016/05/26 13:46:28 Yes, TLS is enabled
ERR - 2016/05/26 13:46:28 os.Args returns: [/go/bin/c6elf96377aac59bb0985b26ad78a8958dcdca7db5702ff4882fa
ERR - 2016/05/26 13:46:28 Registering.. sending REGISTER
ERR - 2016/05/26 13:46:28 Chaincode Keepalive Time is
ERR - 2016/05/26 13:46:28 []Received message REGISTERED from shim
ERR - 2016/05/26 13:46:28 []Handling ChaincodeMessage of type: REGISTERED(state:created)
ERR - 2016/05/26 13:46:28 Received REGISTERED, ready for invocations
ERR - 2016/05/26 13:46:29 [c6elf963]Received message INIT from shim
ERR - 2016/05/26 13:46:29 [c6elf963]Handling ChaincodeMessage of type: INIT(state:established)
ERR - 2016/05/26 13:46:29 Entered state init
ERR - 2016/05/26 13:46:29 [c6elf963]Received INIT, initializing chaincode
ERR - 2016/05/26 13:46:29 [c6elf963]Inside putstate, isTransaction = true
ERR - 2016/05/26 13:46:29 [c6elf963]Sending PUT_STATE
ERR - 2016/05/26 13:46:29 [c6elf963]Received message RESPONSE from shim
ERR - 2016/05/26 13:46:29 [c6elf963]Handling ChaincodeMessage of type: RESPONSE(state:init)
ERR - 2016/05/26 13:46:29 [c6elf963]before send
ERR - 2016/05/26 13:46:29 [c6elf963]after send
ERR - 2016/05/26 13:46:29 [c6elf963]Received RESPONSE, communicated (state:init)
ERR - 2016/05/26 13:46:29 [c6elf963]Received RESPONSE. Successfully updated state
ERR - 2016/05/26 13:46:29 [c6elf963]Init succeeded. Sending COMPLETED
ERR - 2016/05/26 13:46:29 [c6elf963]Move state message COMPLETED
ERR - 2016/05/26 13:46:29 [c6elf963]Handling ChaincodeMessage of type: COMPLETED(state:init)
ERR - 2016/05/26 13:46:29 [c6elf963]send state message COMPLETED
ERR - 2016/05/26 13:46:29 [5852a903]Received message QUERY from shim
ERR - 2016/05/26 13:46:29 [5852a903]Handling ChaincodeMessage of type: QUERY(state:ready)
ERR - 2016/05/26 13:46:29 [5852a903]Sending GET_STATE
ERR - 2016/05/26 13:46:29 [5852a903]Received message RESPONSE from shim
ERR - 2016/05/26 13:46:29 [5852a903]Handling ChaincodeMessage of type: RESPONSE(state:ready)
```

4. Click the **Live Logs** tab on the service page.

This page shows the same logs that were shown from the **Network** tab except that these update live if you selected **Tail**. You can also see a combined view of multiple validating peers and the Certificate Authority.

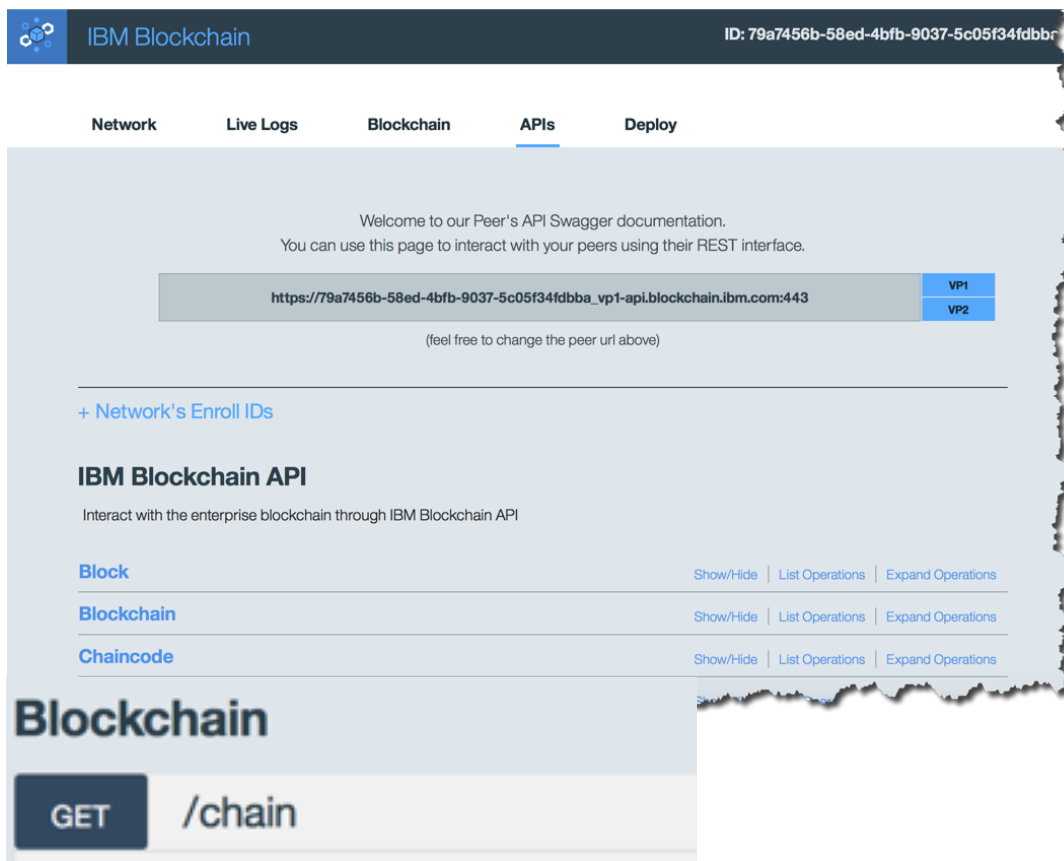
5. Click **VP1**, **VP2**, and **CA** to toggle the live logs view for each peer.

2.4 Interact with the peers

It is possible to invoke the management APIs that interact directly with the peers. In this lab, you will be trying out these APIs directly from the IBM Blockchain service environment.

These APIs are used to operationally manage the blockchain. This is not the same as adding and invoking transactions through chaincode.

1. Click the **APIs** tab on the Service page.

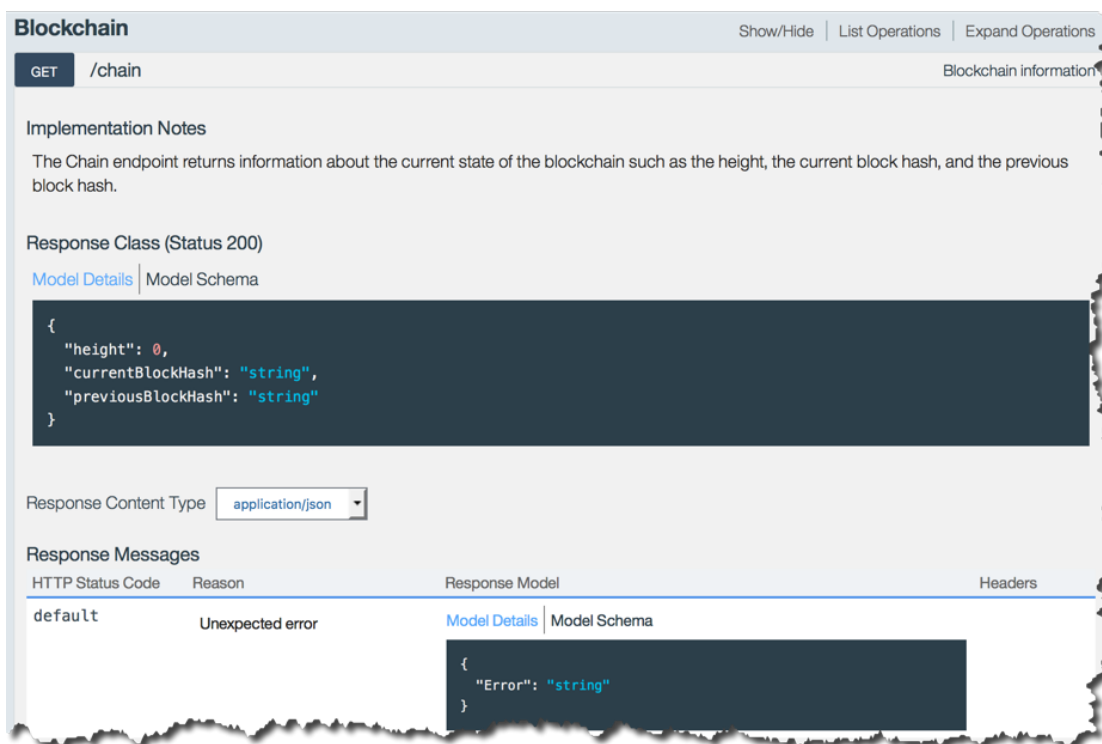


This page allows you to invoke APIs that will directly interrogate and manage the blockchain. First, you will use the API endpoint to query the height of the blockchain (the number of blocks).

2. Click the **Blockchain** section.

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

3. Click **Expand Operations** to view information about this API. This displays the input and output data formats.



The screenshot displays the IBM Blockchain API Explorer interface for the `/chain` endpoint. The interface includes tabs for `GET` and `/chain`, and a `Blockchain information` tab. The `Implementation Notes` section states: "The Chain endpoint returns information about the current state of the blockchain such as the height, the current block hash, and the previous block hash." The `Response Class (Status 200)` section shows the `Model Details` tab with a JSON schema:

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

 Below this, the `Response Content Type` is set to `application/json`. The `Response Messages` section contains a table with one row:

HTTP Status Code	Reason	Response Model	Headers
default	Unexpected error	Model Details Model Schema	

 The `Model Details` tab for the error message shows a JSON schema:

```
{  "Error": "string"}
```

4. Click **Try It Out!** to invoke the API.

Curl

Request URL

```
https://79a7456b-58ed-4bfb-9037-5c05f34fdbba_vp1-api.blockchain.ibm.com:443/chain
```

Response Body

```
{
  "height": 80,
  "currentBlockHash": "mgbwJnFsZQEPCkK5t3YVIF1z7Q6rUMdLxLfede6AX7AbidS5tc81dZsuuWO7sN2zODBneawVfCKKIlzOfc7Q==",
  "previousBlockHash": "vBr27Y6mkqHhRCLVWQbe+o26TmEQTYymSbJuk0CPAAvTUs2vsnPff1XBS47q95qV1CJtewCCk3tIT+FZC5utQ=="
}
```

Response Code

```
200
```

Response Headers

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

Review the displayed fields:

- **Request URL:** shows the URL that was invoked, including the endpoint information of the peer (hostname:port) and the method call (`/chain`).
 - **Response Body:** shows the information that was returned including, importantly, the height of the blockchain.
 - **Response Code:** 200 shows that the request was successful.
 - **Response Headers:** a field that confirms that the response body data was returned in a JSON data structure.
5. Expand the **Block** section and review the information about how to interrogate an individual block in the blockchain.

Block Show/Hide List Operations Expand Operations

GET `/chain/blocks/{Block}` Individual block information

Implementation Notes

The {Block} endpoint returns information about a specific block within the Blockchain. Note that the genesis block is block zero.

Response Class (Status 200)

[Model Details](#) | [Model Schema](#)

```
{
  "proposerID": "string",
  "timestamp": {
    "seconds": 0,
    "nanos": 0
  },
  "transactions": [

```


6. Enter the Block parameter to be a number less than the height of the chain and click **Try it out!**

Parameters				
Parameter	Value	Description	Parameter Type	Data Type
Block	<input type="text" value="40"/>	Block number to retrieve	path	integer
Response Messages				
HTTP Status Code	Reason	Response Model		Headers
default	Unexpected error	Model Details	Model Schema	
		<pre>{ "Error": "string" }</pre>		
<div>Try it out!</div>				

7. Review the information returned in the **Response Body**.

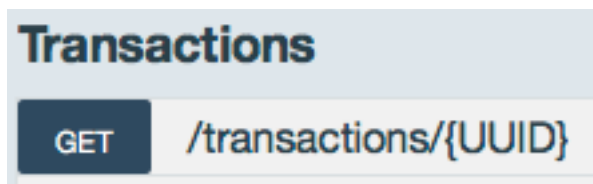
Request URL

```
https://79a7456b-58ed-4bf6-9037-5c05f34fdbba_vp1-apl.blockchain.ibm.com:443/chain/blocks/40
```

Response Body

```
{
  "transactions": [
    {
      "type": 2,
      "chaincodeID": "EoABYzZIMWY5NjM3N2FhYzU5YmUwOTgYIjY2YWQ3OGE4OTU4ZGNkY2E3ZGh1bzAyZmY0ODgyZmEmZOTRiZA2MDU4YmQ3OTE5MmxkYXY1ZmY4MmNkZGZYIWhiMDA3MGEzNA2OGQ0NDEx",
      "payload": "CrlBCAESgwESgAFJNmUxZjk2Mzc3YWFnNTlYYA50DVmJZhZDc4YTg5NThkY2RlY2dkYUJ3MDJmZjQ0ODJmYTM5NGUjMDMwNThiZDc5MTkyYjFlNjVnZjgyY2RkZmNhOGIwMDcwYTM2MDY4ZDQ0",
      "uid": "a338564e-ceef-4df6-9efd-95b65fa43efc",
      "timestamp": {
        "seconds": 1464270464,
        "nanos": 266423527
      },
      "nonce": "s+mSVUX6XeUBLf4brf4YPp4sz56sjXo",
      "cert": "MIICQTCCAJOgAwIBAgIRAOt1VhHEMIITZW6Aq+AUwCgYIKoZIzj0EAwwKTELMAkGA1UEBhMCVVmkDDAKBgNVBAoTA0CTTEMMAAoGA1UEAxMDdGNhMB4XDTE2MDUyNjEzNDUyInloXDTE2MDQyMEQCIDAB48rgMyGZY52Bwpz2p2WshRGUYKhvPwMgoCc3zB24AIBDatFcqdYOMBHhuOexQBAAaDNF+6ZBuDWqrpt1RYlgw==",
      "signature": "MEQCIB48rgMyGZY52Bwpz2p2WshRGUYKhvPwMgoCc3zB24AIBDatFcqdYOMBHhuOexQBAAaDNF+6ZBuDWqrpt1RYlgw=="
    }
  ],
  "stateHash": "02UFHgocVkoNafxxRAdqMXNOETIZRoy4aDP1gxox2WBHENy+DPKZeBskSePwiFYNIHzYINsEXiZLXmcL9g29elg==",
  "prevBlockHash": "cxCELk0Tf5Xm0RJ6R2fnRyo/B2EIL3NKs4xZJ6TKPGqcqGcsKnripsMxuKZJ7o2e0C4tkFr2r3hBokcRw=:"
```

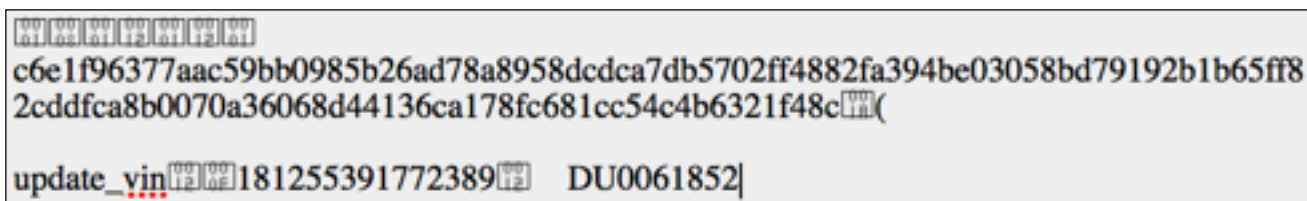
8. Copy the UUID field of a transaction from a block; this will be of the form a338564e-ceef-4df6-9efd-95b65fa43efc.
9. Click the **Transactions** section.



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

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

The **payload** field is base64 encoded. You can use a web tool such as <http://www.base64decode.org> to decode this information. When the information is decoded, you'll see that the payload includes the chaincode ID of the smart contract being called together with its input parameters as shown in the image.



This application does not encrypt the transactions, so the payloads are visible to all although they are encoded in base64.

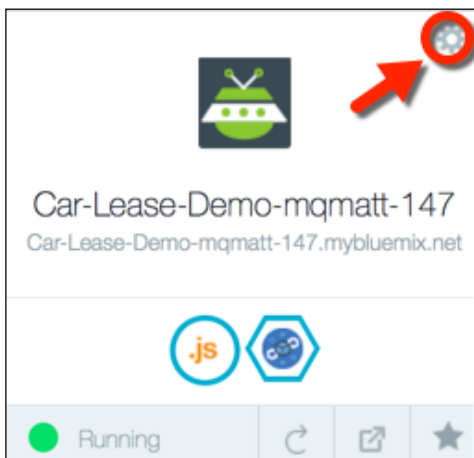
To be able to copy the full payload data, use the browser debug mode by pressing F12 or copy the Request URL shown and use cURL or Postman to make the request.

11. Interact with the other APIs that are available to you.

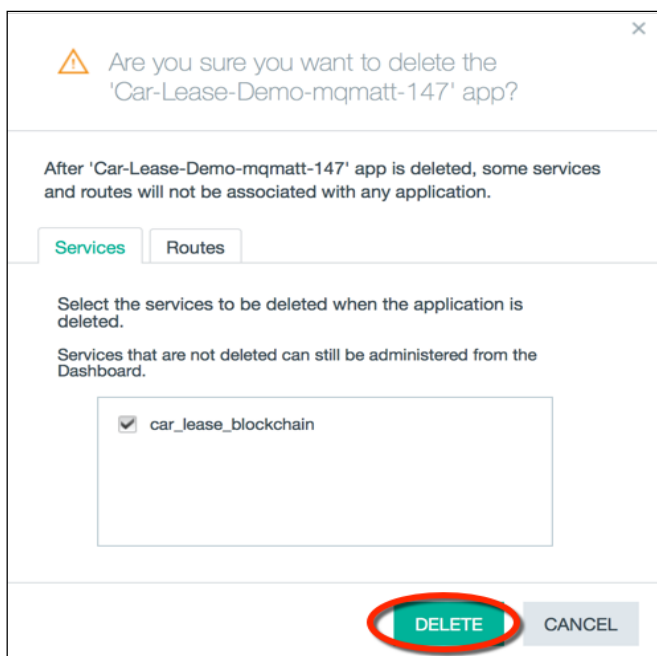
Supplemental: Remove the sample application

If the blockchain service that you created crashes, here's how you can stop and remove it.

1. Click **Dashboard** to return to the Bluemix dashboard.
2. Click the settings icon in the upper-right corner of the car lease demo application.



3. Select **Delete App** from the menu.
4. Ensure that the `car_lease_blockchain` service is also selected for deletion and click **Delete**.



Wait for the items to stop and be deleted. After this is done, both the application and the associated service will no longer be visible in the Bluemix dashboard.