Deploy a sample application to the IBM Blockchain Platform

Get up and running with the Enterprise Membership Plan fast

Yogendra K. Srivastava
Matthew Golby-Kirk

September 20, 2017

In this tutorial, see how to create a network on the IBM Blockchain Platform (Enterprise Membership Plan), install a smart contract (chaincode) in a channel on the network, and deploy a sample application to the network. You'll then see how to populate, query, and update the shared ledger.

In this tutorial, see how to create a network on the IBM Blockchain Platform, install a smart contract (chaincode) in a channel on the network, and deploy a sample application to it. You'll see how to populate the shared ledger, and communicate with it by making calls from a local client application to query and update the ledger.

The IBM Blockchain Platform, previously known as the High Security Business Network (HSBN), is a fully integrated and enterprise-ready service running on the IBM Cloud. The platform is designed to accelerate the development, governance, and operation of a multi-institution business network.

The sample application we'll use in this tutorial is a Hyperledger Fabric sample called "fabcar" and, while you can run it locally in a Docker container, in this tutorial you'll see how to install that application and run it on the IBM Blockchain Platform on the IBM Cloud. Hyperledger is a global, open source, collaborative effort, hosted by The Linux Foundation, to advance cross-industry blockchain technologies. Hyperledger Fabric is a Hyperledger project and framework implementation whose modular architecture powers the IBM Blockchain Platform.
1. Create a network on the IBM Blockchain Platform

Not using the Enterprise Membership Plan?

No problem! You can prototype blockchain apps for free, starting with our online samples. Develop and test locally in a complete dev environment. Later you can share your blockchain app for collaborative development on the IBM Cloud.

Learn more ways to get started with Blockchain today.

To complete this tutorial, you need an IBM Cloud account and a paid subscription to the IBM Blockchain Platform's Enterprise Membership Plan, which will enable you to activate a blockchain network in production. Please note that the Platform is updated often, and the screen shots in this tutorial may not exactly match the ones you see.

1. Starting on the Rapidly build with IBM Blockchain Platform page, select Service Plans. Then click Upgrade (if you already have a trial account) or Create (if you don't have a trial account). You can now see the welcome screen. Click Create Network to see the dashboard for your network.
2. You can now see the Create Network wizard. Fill in the wizard.

Let's Get Started
Start by giving your network a name

Fabcar

Location

Frankfurt

☐ Allow Member Orgs to add peers from other listed locations (coming soon).

Institution Name

IBM

3. On the next page, you can invite members by adding their institution name and email address. However, for this tutorial, you don't need to invite any members; you can just click Next. On the next page, you can accept the default governance rules and policies by clicking Next. On the last page, you can review the summary of the details you provided and click Done.
4. When you're done, you'll see a summary page; click **Enter Network** to get an overview of your blockchain network and the resources you just created.
5. Click **Add Peers** on the Overview page and add the number of peers. For this tutorial, 2 peers is sufficient.
6. Peers will now appear on the Overview page.

![Overview](image)

**2. Create a channel**

**Glossary**

Check the [Glossary for Hyperledger Fabric](#) for any unfamiliar terms.

The next step is to create a channel called “fabcar.” A **channel** is a logical grouping of selected peers in the blockchain network. A channel allows peers associated with that channel to see each other’s transactions, but peers outside the channel cannot see these transactions. It provides a level of transaction privacy and security among the peers. Each channel could have same or different chaincode instantiated, based on business needs.
1. In the blockchain network, click the **Channels** tab on the left, and select **Create Channel**. Enter the channel name `fabcar` and then click **Next**.

**Create a new channel request**

Channels are where your network members transact.

**01. Setup a channel name**

**Channel Name**

`fabcar`

**Description**

*Optional*

*Enter the description of your new channel*

[Cancel] [Next]
2. Assign each member a role and click **Next**. Make sure all options are selected for this step.

**Create a new channel request**

Follow the steps below to create a channel. Once submitted, the invited members will be notified and either approve or decline the request.

**02. Select members and set roles**

Please select the appropriate network members to participate in the channel. At least one operator must be selected in order to proceed.

**Member Name:**

Member Name: [Select a Member of the Network]

<table>
<thead>
<tr>
<th>Member</th>
<th>Permissions</th>
<th>Remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM <a href="mailto:ysrivastava@in.ibm.com">ysrivastava@in.ibm.com</a></td>
<td>Operator, Writer, Reader</td>
<td></td>
</tr>
</tbody>
</table>

[Reset] [Cancel] [Back] [Next]
3. Review the channel update policy and click **Submit Request**.

### Create a new channel request

Channels are where your network members transact.

**03. Review channel update policy**

Set this channel’s update policy.

**Policy**

- of 1 admin from the following organizations need to accept to update this channel.

<table>
<thead>
<tr>
<th>Organization</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td><a href="mailto:ysrivastava@in.ibm.com">ysrivastava@in.ibm.com</a></td>
</tr>
</tbody>
</table>

[Submit Request]

4. You will find a notification in the **Notifications** tab. Select the Channel Request checkbox and click **Review Request**.

### Notifications

These are the action items waiting for your response.

- Unread (1)
- Read (0)
- Completed (0)

[Search in notifications]

[Delete Requests]

<table>
<thead>
<tr>
<th>NAME</th>
<th>UPDATED</th>
<th>STATUS</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Request</td>
<td>Join &quot;fabcar&quot;</td>
<td>By: IBM</td>
<td>0/1 voted</td>
</tr>
<tr>
<td></td>
<td>01 August, 2017 - 2:42:30 PM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5. Review the channel policy and members. Click **Accept** to accept the channel invitation.

**Channel Invitation**

*Join "fabcar"*

IBM has invited you to join **fabcar**, the channel will be created once **1 member has accepted and submitted the policy**. Invited members will be emailed when the channel is created.

- Channel Policy
- Channel Members

![Later, Decline, Accept buttons]

6. On the **Notifications** tab, select the channel and click **Submit Request**.

**Notifications**

These are the action items waiting for your response.

![Search notifications, Delete Requests buttons]

<table>
<thead>
<tr>
<th>NAME</th>
<th>DATE UPDATED</th>
<th>STATUS</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Request</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Join &quot;fabric&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

By: IBM
15 September, 2017 - 11:03:14 AM

- Vote Accepted

Submit Request
7. Select the **Channels** tab and click **Submit**.

```
Channel Invitation
Submit request to create fabcar

The channel will be activated once the channel policy requirements are met

**Current Policy**
1 member must approve to create the channel

**Current Members**

<table>
<thead>
<tr>
<th>Member</th>
<th>Role</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>Operator, Writer, Reader</td>
<td>Accepted</td>
</tr>
</tbody>
</table>
```

8. Click **Join Peers** on the **Channels** tab.
9. Select the peers who should join this channel. Click Add Selected.

The channel 'fabcar' has been created

Join your peers to the channel

<table>
<thead>
<tr>
<th>PEER NAME</th>
<th>PEER STATUS</th>
<th>MSP ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>fabric-peer-org1-21824c</td>
<td></td>
<td>PeerOrg1</td>
</tr>
<tr>
<td>fabric-peer-org1-3071c</td>
<td></td>
<td>PeerOrg1</td>
</tr>
</tbody>
</table>

10. Now you can see that the channel has been created.

3. Install chaincode on the channel

After the channel is created, the next step is to install the chaincode. We'll use the chaincode from the “fabcar” sample application.

1. To obtain the chaincode, follow the instructions for "Getting a Test Network" or git clone this repo:

   git clone https://github.com/hyperledger/fabric-samples.git

2. To install the chaincode, select the Chaincode tab, choose a peer, and click Install chaincode. Enter "fabcar" for the chaincode name, enter "v3" for the version of the chaincode, and click Choose Files to select the fabcar.go chaincode file from the
Deploy a sample application to the IBM Blockchain Platform

fabric-samples/chaincode/fabcar folder you downloaded from GitHub. Click Submit.

3. The chaincode is now installed on the peer.

4. The next step is to instantiate the chaincode on a channel by calling the init method of the chaincode. For this step, no arguments are needed, so you can just choose the fabcar channel and leave the arguments field empty.
5. The instantiated chaincode will look like this:

```
<table>
<thead>
<tr>
<th>CHAINCODE ID</th>
<th>VERSION</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>fabcar</td>
<td>v2</td>
<td>:</td>
</tr>
</tbody>
</table>
```

6. When the chaincode has been instantiated on the channel, select the Channels side tab and choose the **Chaincode** option. You can then see three buttons under fabcar:
   - **JSON**: This JSON file holds the credentials and peer information for the blockchain network.
   - **Logs**: All the chaincode logs get printed here.
   - **Delete**: This deletes/stops the chaincode instance.

In the fabric-samples/fabcar folder, create a new folder called config. Click the **JSON** button, and a new tab will open. Copy all the data shown in this tab into a new file called blockchain_creds.json and save this file inside the config folder.

### 4. Configure your application to run on the IBM Blockchain Platform

To run your application on the IBM Blockchain Platform, you need to make few changes to the fabcar sample.

Start by copying the `extractCerts.js` and `enrollUser.js` files into the root of the fabcar folder `fabric-samples/fabcar`.

**Run npm install**

Run `npm install` from within the fabcar folder:

```
cd fabric-samples/fabcar
npm install
```
On Windows, you may need to install the Windows Build Tools if you get errors:

```
npm install --global windows-build-tools
```

### Extract certificates

In the IBM Blockchain Platform, Transport Layer Security (TLS) is enabled, and therefore you need to extract the TLS certs provided by the service so that you can establish a connection by following these steps.

Make a directory called network in the fabcar directory, and then run extractCerts.js as shown below. This will create a CA certificate and a peer certificate in the given path, or by default in the ./network/tls folder inside the fabcar directory. By default, it will take input from /config/blockchain_creds.json inside the fabcar directory.

```
node extractCerts.js
```

or

```
node extractCerts.js --input= /config/blockchain_creds.json --cert_path=./network/tls
```

In addition to creating the certificate files, this command outputs an EnrollId, an EnrollSecret, and a CA_URL to be used in the next step. It also outputs three URLs to be used next to enroll users (peer-url, peer-event-url, and orderer-url).

### Enroll users

Using the EnrollId, EnrollSecret, and CA_URL from the previous section, run the following command. It will generate a certificate in the given wallet path for the user provided.

```
node enrollUser.js --wallet_path=./network/creds --user_id=admin --user_secret=secret --ca_url=https://fft-zbcxxx.4.secure.blockchain.ibm.com:15185
```

### Initialize the chaincode

Before you can query and update the ledger, you need to run a setup transaction to populate the ledger with some sample data. Make a copy of the invoke.js file that is in the fabric-samples/fabcar folder and call the copy invokeNetwork.js. Make sure the copy is in the same folder as the original file. Then make the following changes to the `invokeNetwork.js` command.

1. Add the following `require` parameter just below the `util` variable.

   ```javascript
   var fs = require("fs")
   ```

2. Modify the `options` variable as shown below. Make sure you update the `peer_url`, `event-url`, and `orderer-url` fields to the values that were returned from running `extractCerts.js` above.
var options = {
    wallet_path: path.join(__dirname, './network/creds'),
    user_id: 'admin',
    channel_id: 'fabcar',
    chaincode_id: 'fabcar',
    peer_url: '<peer_url>',
    event_url: '<peer-event-url>',
    orderer_url: '<orderer-url>',
    tls_cert: {
        pem: fs.readFileSync(path.join(__dirname, './network/tls') + '/peer.cert').toString(),
    }
};

3. To enable TLS, replace this code:

```javascript
var peerObj = client.newPeer(options.peer_url);
```

with:

```javascript
var peerObj = client.newPeer(options.peer_url, {
    pem: options.tls_cert.pem
});
```

4. Replace this code:

```javascript
channel.addOrderer(client.newOrderer(options.orderer_url));
```

with:

```javascript
channel.addOrderer(client.newOrderer(options.orderer_url, {
    pem: options.tls_cert.pem
}));
```

5. Replace this code:

```javascript
eh.setPeerAddr(options.event_url);
```

with:

```javascript
eh.setPeerAddr(options.event_url, {
    pem: options.tls_cert.pem
});
```

6. Finally, change the value of the `fcn` field in the `request` variable from `createCar` to `initLedger`, and make `args` an empty array. The changed `request` variable should now look like this:

```javascript
var request = {
    targets: targets,
    chaincodeId: options.chaincode_id,
    fcn: 'initLedger',
    args: [''],
    chainId: options.channel_id,
    txId: tx_id
};
```

7. Now save the file and run the new command using `node invokeNetwork.js`. The expected output is:

```javascript
Successfully sent transaction to the orderer.
```

**Query the ledger**

The `invokeNetwork.js` command has now populated the ledger with some sample data, so let's query the ledger to see the data. Make a copy of the `query.js` file that is in the `fabric-samples/fabcar` folder and call the copy `queryNetwork.js`. Make sure the copy is in the same folder as the original file. Then make the following changes to the `queryNetwork.js` command.
1. Add the following `require` parameter just below the `path` variable.

   ```javascript
   var fs = require("fs")
   ```

2. Modify the `options` variable as shown below. Make sure you update the value of the `network_url` field to the value of the `peer_url` that was returned from running `extractCerts.js` above.

   ```javascript
   var options = {
     wallet_path: path.join(__dirname, './network/creds'),
     user_id: 'admin',
     channel_id: 'fabcar',
     chaincode_id: 'fabcar',
     network_url: '<peer_url>',
     tls_cert: {
       pem: fs.readFileSync(path.join(__dirname, './network/tls') + '/peer.cert').toString(),
     }
   };
   ```

3. Replace this code:

   ```javascript
   channel.addPeer(client.newPeer(options.network_url));
   ```

   with:

   ```javascript
   channel.addPeer(client.newPeer(options.network_url, {
     pem: options.tls_cert.pem
   }));
   ```

Now save the file and run the new command using `node queryNetwork.js`. The expected output is:

```
Query result count = 1
Response is [{"Key":"CAR0","Record":{"make":"Toyota","model":"Prius","colour":"blue","owner":"Tomoko"}},{"Key":"CAR1","Record":{"make":"Ford","model":"Mustang","colour":"red","owner":"Brad"}},{"Key":"CAR10","Record":{"make":"Honda","model":"Accord","colour":"Black","owner":"yog"}},{"Key":"CAR11","Record":{"make":"Honda","model":"Accord","colour":"Black","owner":"Tom"}},{"Key":"CAR12","Record":{"make":"Honda","model":"Accord","colour":"Black","owner":"Tom"}},{"Key":"CAR2","Record":{"make":"Hyundai","model":"Tucson","colour":"green","owner":"JinSoo"}},{"Key":"CAR3","Record":{"make":"Volkswagen","model":"Passat","colour":"yellow","owner":"Max"}},{"Key":"CAR4","Record":{"make":"Tesla","model":"S","colour":"black","owner":"Adriana"}},{"Key":"CAR5","Record":{"make":"Peugeot","model":"205","colour":"purple","owner":"Michel"}},{"Key":"CAR6","Record":{"make":"Chery","model":"S2ZL","colour":"white","owner":"Aarav"}},{"Key":"CAR7","Record":{"make":" Fiat","model":"Punto","colour":"violet","owner":"Pari"}},{"Key":"CAR8","Record":{"make":"Tata","model":"Nano","colour":"indigo","owner":"Valeria"}},{"Key":"CAR9","Record":{"make":"Holden","model":"Barina","colour":"brown","owner":"Shotaro"}]}
```

**Update the ledger**

Finally, let's make an update to the ledger. To do this, you can make another simple change to the new `invokeNetwork.js` command that you created above.

1. Edit the `request` variable as shown below so that it will invoke the `createCar` chaincode with a set of arguments that describe the car to be created. The changed `request` variable should look like this:

   ```javascript
   var request = {
     targets: targets,
     chaincodeId: options.chaincode_id,
     fcn: 'createCar',
     args: ['CAR11', 'Honda', 'Accord', 'Black', 'Tom'],
     chainId: options.channel_id,
     txId: tx_id
   };
   ```
2. Save the file and run the command using `node invokeNetwork.js`. The expected output is:

```
Successfully sent transaction to the orderer.
```

This has created a new vehicle with the owner "Tom" and stored it on the ledger. You can see the new car on the ledger by running the `queryNetwork.js` command again. You can now experiment with creating new cars on the ledger with different names and owners.

3. Finally, you may want to experiment with the `changeOwner` chaincode to change the owner of a vehicle. To do this, change the `request` variable in `invokeNetwork.js` again to look like this:

```javascript
var request = {
  targets: targets,
  chaincodeId: options.chaincode_id,
  fcn: 'changeCarOwner',
  args: ['CAR11', 'MGK'],
  chainId: options.channel_id,
  txId: tx_id
};
```

4. Now save the file and run the command again using `node invokeNetwork.js`. The expected output is:

```
Successfully sent transaction to the orderer.
```

5. You can see the updated owner on the ledger by running the `queryNetwork.js` command again. You can see the owner has changed from "Tom" to "MGK."

6. If you want to query for a single car rather than for all cars, make this change to the `request` variable in the `queryNetwork.js` command and rerun it:

```javascript
const request = {
  chaincodeId: options.chaincode_id,
  txId: transaction_id,
  fcn: 'queryCar',
  args: ['CAR11']
};
```

You should now see the information for a single car:

```
Query result count = 1
Response is {"make":"Honda","model":"Accord","colour":"Black","owner":"MGK"}
```

**Summary**

You now have a running network on the IBM Blockchain Platform, with sample chaincode in a channel on the network and a running application that you can easily work with. You've populated the ledger with sample data, and your application can now communicate with (query and update) the blockchain on the IBM Blockchain Platform. Happy blockchaining!

**Acknowledgments**

The authors thank Anthony O'Dowd and David Gorman of the IBM Blockchain Labs Global Engagement team for their expert guidance and support throughout the development of this tutorial.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>enrollUser.js and extractCerts.js</td>
<td>2-javascript-files.zip</td>
<td>3KB</td>
</tr>
</tbody>
</table>
Related topics

- Blockchain newsletter for developers
- Blockchain Developer Center
- Blockchain developer sandbox
- Blockchain essentials (free course for developers)
- Blockchain solutions and services from IBM
- Hyperledger Composer
- Hyperledger Fabric
- Hyperledger community

© Copyright IBM Corporation 2017

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)