Deploy a sample application to the IBM Blockchain Platform Starter Plan

Use the Hyperledger Fabric "fabcar" sample to get up and running with the Starter Plan fast

Matthew Golby-Kirk
David Gorman
Yogendra K. Srivastava

July 14, 2018

Step by step, see how to create a network on the IBM Blockchain Platform, install a smart contract (chaincode), and run a sample application to invoke it. You'll also 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.

Developers, start your (blockchain) engines!

Need an easy and economical way to kick-start your business network on the IBM Cloud? Try the new IBM Blockchain Platform Starter Plan! It’s by developers, for developers.

In this tutorial, you'll see how to create a network on the IBM Blockchain Platform Starter Plan, install a smart contract (chaincode) onto a peer in the network, and run a sample application to invoke the smart contract. You'll also 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 Starter Plan 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 Starter Plan is specifically aimed at test and development scenarios rather than production ones. For production scenarios, you should use the Enterprise Plan.

About Hyperledger

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

Please note that the IBM Blockchain Platform is updated often, and the screen shots in this tutorial may not exactly match the ones you see.

Learning objectives

- Get familiar with the IBM Blockchain Platform Starter Plan.
- Create your first network on the Starter Plan.
- Install chaincode onto a peer in the network.
- Run a sample application to invoke the chaincode.
- Populate, query, and update the shared ledger.

Time to complete this tutorial

- Approximately 45 to 60 minutes.

Prerequisites

- You'll need to create an IBM Cloud account, if you don't have one already.
- You'll need git installed on your local machine, so that you can get sample chaincode from GitHub to deploy.
- You'll need Node.js and npm installed on your local machine, so that you can configure your application. Currently Node.js v9.x is not supported; you need to use v8.9.x.

1. Create a network on the IBM Blockchain Platform Starter Plan

1. Log in to your IBM Cloud account (or create an account, if you don't yet have one).
2. Starting on the Rapidly build with IBM Blockchain Platform page, select Service Plans. Make sure you are logged in to your IBM Cloud account.
   Enter Blockchain-demo as the Service name, and then select Starter Plan Membership and click Create.
3. You can now see the "Network created!" panel. Click **Launch** on this panel to see the dashboard for your network.

4. Next you see the "Let's get started!" welcome screen. Click **Got it** when you have finished reading.

5. Notice at the top left that your network has been given a generated name. Click the name and change it to **fabcar**.

6. Once you have changed the name, you will see a pop-up in the top right for a few seconds informing you that the change was successful:
7. Out of the box, you can see that the Starter Plan creates a working simple network for you. Click the **Overview** tab, and you can see exactly what was automatically created for you:

### Overview

View and manage network resources for your organization. You can stop or start your resources and view the log file of the resource by selecting View Logs under “Actions”. [Learn more](#)

<table>
<thead>
<tr>
<th>TYPE</th>
<th>NAME</th>
<th>STATUS</th>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orderer</td>
<td>orderer</td>
<td>Running</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>org1-ca</td>
<td>Running</td>
<td></td>
</tr>
<tr>
<td>Peer</td>
<td>org1-peer1</td>
<td>Running</td>
<td></td>
</tr>
</tbody>
</table>

The network actually consists of two organizations: **Company A (Org1)** and **Company B (Org2)**. You are logged in as Org1 by default, and so you can only see the **Orderer** service, which is shared by both orgs, along with a **Certificate Authority (CA)** and a **Peer** for Org1.

8. Click the **Members** tab on the left to see these network members in a little more detail:

### Members

View and manage the members (organizations) of the network on the Members tab. You can invite other organizations to the network or add more organizations in addition to the two organizations that you own by default. You can also view and manage admin certificates that are associated to your organizations on the Certificates tab. [Learn more](#)

<table>
<thead>
<tr>
<th>Company A</th>
<th>MSP ID</th>
<th>REQUESTER</th>
<th>STATUS</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:mgh@uk.ibm.com">mgh@uk.ibm.com</a></td>
<td>org1</td>
<td>--</td>
<td>Joined</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company B</th>
<th>MSP ID</th>
<th>REQUESTER</th>
<th>STATUS</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:mgh@uk.ibm.com">mgh@uk.ibm.com</a></td>
<td>org2</td>
<td>--</td>
<td>Joined</td>
<td></td>
</tr>
</tbody>
</table>
9. Click the **Channels** tab on the left, and you will see that a channel called **defaultchannel** has been created as well:

<table>
<thead>
<tr>
<th>ID</th>
<th>TIME CREATED</th>
<th>BLOCK HEIGHT</th>
<th>PEERS</th>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultchannel</td>
<td>05/24/18 14:02 GMT</td>
<td>3</td>
<td>org1-peer1</td>
<td></td>
</tr>
</tbody>
</table>

*Channels created with the SDK cannot be edited here (yet)*

10. For this tutorial, we will use this defaultchannel for simplicity. Click the **defaultchannel** row in the table to see more details on the channel:

11. Notice that here are already 3 total blocks on the chain held by the channel. These blocks store the initial configuration information. You can select each row to see more details if you wish.

### 2. Install chaincode on the channel

The next step is to deploy the chaincode to the channel. But first you need to get the chaincode from a GitHub repository using `git clone`.

1. Make sure you have `git` installed on your local machine.
2. Open a command-line or terminal window, navigate to a suitable directory, and create a new directory called `fabcar`. Navigate to the new `fabcar` directory and run the following command to clone the Fabric samples source:

```bash
git clone https://github.com/hyperledger/fabric-samples.git
```
This command copies the code for all the samples to your local machine. Using the Starter Plan UI, select the **Install Code** tab from the left sidebar to see the "Install code" panel:

### Install code

Chaincode must be installed on a peer. Select a peer and then install a chaincode on it. After you install the chaincode, you can request to instantiate it on a channel by clicking the Instantiate button under "Actions" for that chaincode. Clicking elsewhere on the chaincode will show you what channels the chaincode is instantiated on. [Learn more](https://www.ibm.com/support/knowledgecenter/SSSHTN_65.0.0/com.ibm.bsc.doc/topics/chaincodeinst.html). Do you already have a .cgo file to deploy? See our [deployment guide](https://www.ibm.com/support/knowledgecenter/SSSHTN_65.0.0/com.ibm.bsc.doc/topics/chaincodemgmt.html).

3. Click the drop-down called **Choose-peer...** and select **org1-peer1**. Now click the **Install Chaincode** button.

4. On the "Install chaincode on org1-peer1" dialog, enter **fabcar** for the "Chaincode ID," and enter **v1** for the "Chaincode Version."
   
   You have a choice to deploy either the Node.js or Golang version of the fabcar chaincode. Both provide the same functionality, so the choice is yours. Choose **Node** or **Golang** for the "Chaincode Type."

   Finally, on this panel, click **Choose files** to select the chaincode source to install:
• To use the Go version, select the `fabcar.go` chaincode single file from the `fabric-samples/chaincode/fabcar/go` folder that you downloaded from GitHub.

```
Install chaincode on org1-peer1
```

- `fabcar`
- `v1`
- Chaincode Type: Golang

```
Select local chaincode files to install them on this peer:
Select a zip or multiple chaincode files
```

- Choose files
- `fabcar.go`

• Or, to use the Node.js version, select both the `fabcar.js` file and the `package.json` files from the `fabric-samples/chaincode/`
Deploy a sample application to the IBM Blockchain Platform
Starter Plan

5. Next you need to instantiate the chaincode. Click the three dots in the Actions column to see the menu, and choose Instantiate.

6. On the Instantiate chaincode dialog, there are no arguments to provide for fabcar as it does not need any. However, you do need to select defaultchannel from the Channel drop-down and match the "Chaincode Type" to the language
Deploy a sample application to the IBM Blockchain Platform Starter Plan

(\textit{Node} or \textit{Golang}) of the code you uploaded above. Then click \textbf{Submit}.

### Instantiate Chaincode

**Instantiate chaincode**

Enter the arguments for the 'init()' function as a comma separated list.

**Arguments:**
\[ ex: \ a, \ 200, \ b, \ 250 \]

**Channel:**
\[ \text{defaultchannel} \]

**Chaincode Type:**
\[ \text{Golang} \]

7. Once this completes, you can select the \textbf{fabcar} row to see that the chaincode is now instantiated on the defaultchannel:

**fabcar v1**

**Instantiated on channel:**

- \text{defaultchannel}
8. Now the chaincode has been instantiated on the channel, select the **Channels** side tab and select the **defaultchannel** row to see its details:

![Channels screenshot]

You can now see there is one extra block on the channel, which is the record of the instantiate operation you performed above.

9. Now select the **Chaincode** tab on this panel and select the **fabcar** row to expand it. You can now see two buttons under fabcar:
   - **JSON** is the file that holds the credentials and peer information for the blockchain network.
   - **Delete** stops and deletes the chaincode instance.

![Chaincode screenshot]

10. Finally, click the **JSON** button shown above, and a new tab will open. Copy all the data shown in this tab into a new file called **network-profile.json** and save this file somewhere on your machine; you will need to copy it later.

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

The fabcar applications you downloaded from GitHub are hard-coded to use a local instance of Hyperledger Fabric. Because we're using Hyperledger Fabric located remotely on IBM Blockchain Platform, you'll need to get new versions of the fabcar apps that will connect securely to the remote platform using the **network-profile.json** configuration file you just downloaded and saved above.

The **original** 4 fabcar applications are called:
Get the new versions of the fabcar applications

The new versions of these files are in a GitHub repository called fabcar-network. These new files all end with the word Network to indicate these are the new versions that will access fabcar remotely:

- enrollAdminNetwork.js
- registerUserNetwork.js
- invokeNetwork.js
- queryNetwork.js

One additional file in the GitHub repo is a configuration file, which contains some extra configuration needed to access the remote fabcar chaincode. This file is called:

- client-profile.json

To get these files, run this command from inside the fabric-samples folder:

```
git clone https://github.com/m-g-k/fabcar-network.git
```

This command creates a folder called fabcar-network at the same level as the existing fabcar folder.

Your next step is to copy the network-profile.json file you downloaded and saved earlier into the fabric-samples/fabcar-network/config folder alongside the existing client-profile.json file.

Run npm install

The network version of the fabcar client application is written in Node.js. To complete this next step, you need to have Node.js and npm installed on your local machine.

2. Return to the command line or terminal window and run the `npm install` command from within the fabric-samples/fabcar-network folder:

```
npm install
```

On Windows, you may need to install the Windows Build Tools if you get build errors:

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

You now have all the prerequisite node packages installed in order to run the fabcar client application.

Enroll an Admin

When your network was created in the Starter Plan, each organization had an Administrator user called admin automatically registered with the Certificate Authority (CA). You now need to send an enrolment request to the CA to retrieve that user’s enrollment certificate (eCert).
3. From within the `fabric-samples/fabcar-network` folder, run:

   ```shell
   node enrollAdminNetwork.js
   ```

   You should see output like this:
   ```
   > node enrollAdminNetwork.js
   Found organization: org1 and ca name: org1-ca
   Enrolling using enrollmentId: admin and EnrollmentSecret: 2d87ae1b59
   Successfully enrolled admin user "admin" with msp: "org1"
   Assigned the admin user to the fabric client ::{…}
   ```

   The `enrollAdminNetwork.js` application creates a local public/private key pair in a folder it creates called `hfc-key-store` and sends a Certificate Signing Request (CSR) to the remote CA for org1 to issue the eCert. The eCert, along with some metadata, will also be stored in the `hfc-key-store` folder. The connection details of where the CA is located and the TLS certificate needed to connect to it are all obtained by the application from the `network-profile.json` you downloaded earlier.

### Enroll a new user

Using the eCert for the admin user, you can now register and enroll a new user called `user1` with the CA. This user will be the one whose identity you will use to query and update the ledger.

4. From within the `fabric-samples/fabcar-network` folder, run:

   ```shell
   node registerUserNetwork.js
   ```

   You should see output like this:
   ```
   > node registerUserNetwork.js
   Successfully loaded admin from persistence
   Successfully registered "user1" - with secret:gTYshgoNxoLH
   Successfully enrolled member user "user1" with msp: "org1"
   "user1" was successfully registered and enrolled and is ready to interact with the fabric network
   ```

   Like the previous command, this application has created a new public/private key pair and sent a CSR request to the CA to issue the eCert for `user1`. If you look in the `hlf-key-store` folder, you should see 6 files, 3 for each identity.

### Initialize the ledger

The original fabcar sample has a script that automatically calls an `initledger` transaction to populate the ledger with 10 initial vehicles to get you started quickly. Because we are not using this script, we need to call this transaction ourselves. The `invokeNetwork.js` command has been written to call this directly.

5. From within the `fabcar` folder, run:

   ```shell
   node invokeNetwork.js
   ```

   You should see output that ends like this:
   ```
   Successfully committed the change to the ledger by the peer
   ```
Because the chaincode runs in a separate chaincode docker container, it can take some time to start this container on first use. If you get a timeout or a "premature execution" error, just try running the command again. If you take a look at the Channel Overview after the command has completed successfully, you should see there is now an extra block on the ledger:

```
<table>
<thead>
<tr>
<th>TIME</th>
<th>BLOCK NUMBER</th>
<th>INSTANTIATIONS</th>
<th>INVOCATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>05/25/18 13:22 GMT</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

### Query the ledger

The `invokeNetwork.js` command has now populated the ledger with sample data for 10 cars, so let's query the ledger to see the data. To do this, you'll run the `queryNetwork.js` command, which is set up to query all cars that exist on the ledger.

6. From within the `fabcar` folder, run:

```
node queryNetwork.js
```

You should see output like this:

```
> node queryNetwork.js
Successfully loaded user1 from persistence
Query has completed, checking results
Response is
[
{
"Key": "CAR0", "Record":{
"colour": "blue", "make": "Toyota", "model": "Prius", "owner": "Tomoko"}}],
{
"Key": "CAR1", "Record":{
"colour": "red", "make": "Ford", "model": "Mustang", "owner": "Brad"}}],
{
"Key": "CAR2", "Record":{
"colour": "green", "make": "Hyundai", "model": "Tucson", "owner": "Jin Soo"}}],
{
"Key": "CAR3", "Record":{
"colour": "yellow", "make": "Volkswagen", "model": "Passat", "owner": "Max"}}],
{
"Key": "CAR4", "Record":{
"colour": "black", "make": "Tesla", "model": "S", "owner": "Adriana"}}],
{
"Key": "CAR5", "Record":{
"colour": "purple", "make": "Peugeot", "model": "205", "owner": "Michel"}}],
{
"Key": "CAR6", "Record":{
"colour": "white", "make": "Chery", "model": "S22L", "owner": "Aarav"}}],
{
"Key": "CAR7", "Record":{
"colour": "violet", "make": "Fiat", "model": "Punto", "owner": "Pari"}}],
{
"Key": "CAR8", "Record":{
"colour": "indigo", "make": "Tata", "model": "Nano", "owner": "Valeria"}}],
{
"Key": "CAR9", "Record":{
"colour": "brown", "make": "Holden", "model": "Barina", "owner": "Shotaro"}}]
```

### Update the ledger

Finally, you can now make an update to the ledger. To do this, you need to make a simple change to the `invokeNetwork.js` command.

7. Open the `invokeNetwork.js` file in an editor of your choice, such as atom or VSCode. Then find and edit the request variable, as shown below, so that it will invoke the `createCar` function.
chaincode with a set of arguments that describe the car to be created. The changed request variable should look like this:

```javascript
var request = {
    chaincodeId: 'fabcar',
    fcn: 'createCar',
    args: ['CAR10', 'Honda', 'Accord', 'Black', 'Dave'],
    txId: tx_id
};
```

8. Save the file and run the edited command again using `node invokeNetwork.js`. The expected output is:

```
Successfully committed the change to the ledger by the peer.
```

You can now experiment with creating new cars on the ledger with different names and owners. If you look at the **Channel Overview**, you should see new blocks added as you create new cars.

9. To change the owner of a vehicle, use the `changeCarOwner` transaction to change the request variable in `invokeNetwork.js`. For example:

```javascript
var request = {
    chaincodeId: 'fabcar',
    fcn: 'changeCarOwner',
    args: ['CAR10', 'MGK'],
    txId: tx_id
};
```

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

```
Successfully committed the change to the ledger by the peer.
```

You can see the updated owner on the ledger by running the `queryNetwork.js` command again. You can see the owner of CAR10 has changed from "Dave" to "MGK".

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: 'fabcar',
    fcn: 'queryCar',
    args: ['CAR10']
};
```

You should now see the information for a single car:

```
> node queryNetwork.js
Successfully loaded user1 from persistence
Query has completed, checking results
Response is
{"colour":"Black","make":"Honda","model":"Accord","owner":"MGK"}
```

## Conclusion

You now have a running network on the IBM Blockchain Platform Starter Plan, with sample chaincode deployed to a peer and instantiated on a channel. You also have a running application that you can easily work with locally. 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 of the IBM Blockchain Labs Global Engagement team for his expert guidance and support during the development of this tutorial.

Next steps

- Now that you've deployed the Hyperledger Fabric "fabcar" sample, learn how to create business networks from scratch using Hyperledger Composer in our popular series, Hyperledger Composer basics, by Steve Perry.
- Stay in the know with the monthly Blockchain Newsletter for developers. Check out the current issue and subscribe.
- Check out the many code patterns, which provide roadmaps for solving complex problems with blockchain technology, and include architecture diagrams, code repos, and additional reading.
- Stop by the Blockchain Developer Center. It's your source for free tools and tutorials, along with code and community support, for developing and deploying blockchain solutions for business.

© Copyright IBM Corporation 2018
Trademark
(www.ibm.com/developerworks/ibm/trademarks/)