Deploy Blockchain with Hyperledger Fabric

Due: March 1, 2019

1 Overview

Hyperledger Fabric is an open source enterprise-grade permissioned distributed ledger technology (DLT) platform, designed for use in enterprise contexts. It has many features which can help students get better understanding of some key concepts of blockchain. Fabric has a highly modular and configurable architecture, enabling innovation, versatility and optimization for a broad range of industry use cases including banking, finance and so on. Besides, Fabric is the first distributed ledger platform to support smart contracts authored in general-purpose programming languages such as Java, Go and Node.js, rather than constrained domain-specific languages. It also supports pluggable consensus protocols that enable the platform to be more effectively customized to fit particular use cases and trust models.

The objective of this lab is to familiarize students with permissioned blockchain by building up their own blockchain based supply-chain solution using Hyperledger Fabric. Student will setup the network, write smart contracts to enforce business logic, and define endorsement policies to achieve access control. At the end of this lab, students should be able to build their own blockchain from scratch.

2 Lab Setup

For this lab we prepared a virtualbox image which contains Ubuntu 16.04 and Hyperledger Fabric v1.1. You can download it from the course website.

The username is 'cs590user' and the password is 'cs590pass'. Your working directory will be '/home/cs590user/go'.

You can list all docker images using the following command:

```
docker image ls
```

All images are already downloaded and if you need to download them again, make sure the tag of image is consistent with hyperledger 1.1.

NOTE: The default RAM of virtual image is 8 GB, feel free to change it based on your computer hardware. While it is recommended that RAM should be at least 4GB.

NOTE: We use Hyperledger Fabric v1.1 in this lab which is NOT the latest version of Fabric. However, it is good enough since the goal of this lab is to get a better understanding of permissioned blockchain.

NOTE: Using virtual machine is one option. You can also install Hyperledger Fabric 1.1 on your computer to get better performance.

3 Lab Tasks

3.1 Task 1: Building the network (30 points + 15 bonus points)

In this task you will learn how to build up a blockchain network using Hyperledger Fabric. And you will build a network with three organizations, each with one peer node, and a solo ordering service.

Task 1.1: Generate Network Artifacts (10 points) The first step is using Hyperledger tools to generate network artifacts including genesis block, channel config transaction, etc. You don't need to start it from scratch, instead, there is an example in "$HOME/go/fabric-samples/first-network". You can follow this example or directly modify this example to finish the lab.

The first step is to write configuration files. A sample of such configuration file could be found at '$HOME/go/fabric-samples/first-network/crypto-config.yaml'. More details could be found at Hyperledger official documents:

https://hyperledger-fabric.readthedocs.io/en/release-1.1/build_network.html#crypto-generator

Your task is to modify this file so that the network structure matches the structure mentioned above.

After that, you will use the binary file "cryptogen" to generate artifacts. To do that, run the following command:

```
../bin/cryptogen generate --config=./crypto-config.yaml
```

With similar logic, you will generate configuration transactions using "configtxgen" in the same path, and the corresponding config file that you need to modify is "configtx.yaml". Here you need to make sure the names of organizations match the names in "crypto-config.yaml" and correct path is set for "MSPDir". The ouppút of this step should include: genesis block, anchor peer transaction for each organization, channel creation transaction.

NOTE: More details about how to use these tools is available in the following link:

https://hyperledger-fabric.readthedocs.io/en/release-1.1/build_network.html#manually-generate-the-artifacts

Task 1.2: Edit Docker Compose Files (10 points) Now you get all artifacts including crypto config files of all peers, genesis block , anchor peer transactions and channel creation transactions. The next step is to generate corresponding docker-compose file. Docker-compose file is used to start multiplec docker containers which represent different roles such as peers, ordering nodes and so on.

There are many docker-compose files in "first network" example and the main files are "docker-compose-cli.yaml" and "docker-compose-base.yaml" in "base" directory. Your task is to follow its format and write your own docker compose file which starts a network as described above.

There are several things you need to take care of here. The first one is to mount all certificate files into correct containers. To achieve that, you need to edit the "volumes" section of each peer to send certificate files into corresponding containers. Besides, make sure you allocate different ports to different containers by editing "ports" section.

If everything is correct, you should be able to launch the whole network by the following command:

```
docker-compose -f Your-compose-file-name.yaml up
```

Task 1.3: Run Sample Chaincodes (10 points) Now you are able to start the whole network. Next step is to run sample chaincode which "first-network" runs to show the success of task 1. You should be able to invoke and query the same chaincode. Here is the instruction of how to setup chaincode and interact with it:

https://hyperledger-fabric.readthedocs.io/en/latest/build_network.html#start-the-network

Task 1.4(Bonus): Replace solo ordering service with Kafka-zookeeper cluster (15 points) A solo ordering service becomes single point of failure in this network. Your bonus goal is to replace it with Kafka-zookeeper which provides crash fault tolerance.

You should follow this instruction carefully to finish this task:

`https://hyperledger-fabric.readthedocs.io/en/release-1.1/kafka.html`

3.2 Task 2: Writing Smart Contracts (50 points)

In this task you will learn how to write smart contacts with Go. You need to write smart contracts to enforce the business logic in a car supply chain management example.

Task 2.1: Network Construction For this task you can directly use the "balance transfer" example in fabric samples so that all the networks including CAs, orderding nodes and peers are already there and you can focus on only smart contracts. To run the balance transfer example, first open a terminal, go to "balance-transfer' folder, then type:

```
./runApp.sh
```

This command will start up the network. Then open another terminal, go to "balance-transfer' folder and run the following command:

```
./testAPIs.sh
```

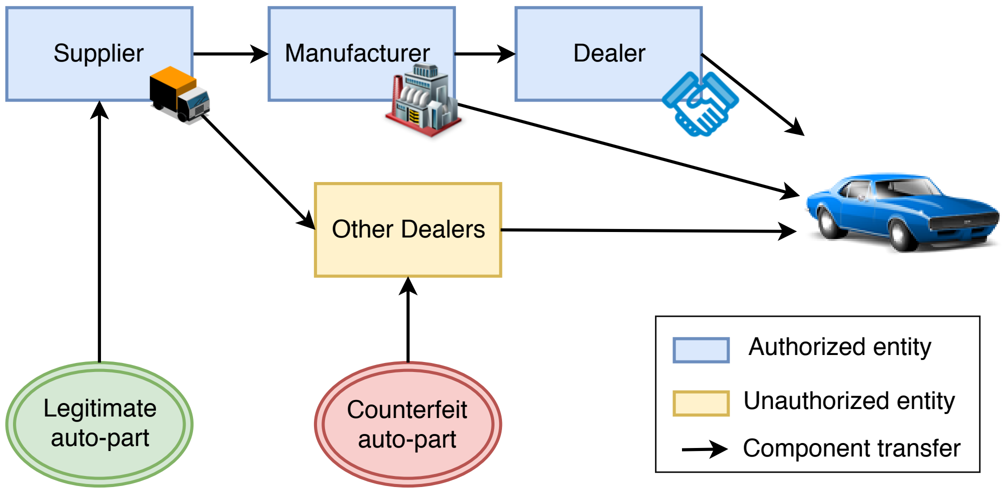
This command tests all the APIs that this example provides. You can use them later for testing.

The smart contract of balance-transfer example is located in

```
"\$HOME/go/fabric-samples/balance-transfer/artifacts/src/github.com/example_cc/go"
```

You can replace it with your chaincode.

Task 2.2: Write Chaincode for Supply Chain Management Now we describe the supply chain scenario of this task. Figure 1 provides a general view of the players in a car supply chain. We have four kind of players in this example. The roles and what they can do are shown in the following table:



Role	Operation
Supplier	(AddComponent, TransferComponent)
Manufacturer	(MountComponent, ReplaceComponent, RecallComponent, TransferComponent)
Authorized dealer	(TransferComponent, ReplaceComponent)
Cars	(CheckComponent)

NOTE: In this example, we will describe entities with the form "ROLE_TYPE.ROLE_NAME" such as "Supplier.supplier1". Please follow this form in your chaincode design.

In this example, suppliers are responsible to produce car components like tires and airbags and then add them to the blockchain. The components should be transferred from suppliers to Manufacture where these components are mounted into new cars. Manufacture could transfer components again to dealers for selling. And cars should also be able to query the blockchain to check if the components mounted is recalled or not.

So you need to write the following functions to enforce business logic mentioned above:

- AddComponent(Role, ComponentID)
- TransferComponent(Role, New Owner, ComponentID)
- MountComponent(Role, ComponentID, CarID)
- ReplaceComponent(Role, ComponentID, CarID)
- RecallComponent(Role, ComponentID)
- CheckComponent(Role)

You need to design access control in chaincode level so that only the correct roles can do corresponding operation. The restrictions are as follows:

- All CompomentID should be 9-digit unique strings.
- The "Role" of AddComponent() has to be a supplier.
- Only Manufactory can do MountComponent() and ReplaceComponent(). You can only mount a component to a new car with a new component. And when you run ReplaceComponent(), the car has to be already with a component mounted and the old component should be retired permanently which means this ID should never be introduced into system again(e.g. through AddComponent()).

- Only the owner of the component could run TransferComponent().
- You cannot recall a retired component().
- CheckComponent() can only be called by car and if the component mounted in this car is a valid one (not recalled or retired), the system should return "True", otherwise return "False".

If you haven't touched chaincode or Go language before, it is recommended that you start with this tutorial:

https://hyperledger-fabric.readthedocs.io/en/release-1.1/chaincode4ade.html

This tutorial also provides links to the list of all APIs which are helpful.

NOTE: There is an example called "chaincode-docker-devmode" in "fabric-samples" folder. It is recommanded to be used for developing chaincodes since it just starts a very basic network and runs the chaincode so that you do not need to worry about the problems caused by any issues not relevant to chaincode. So you can first test your code in "chaincode-docker-devmode" example, then put the code to your project if code works.

3.3 Task 3: Setting up Certificates (20 points)

In previous task, we decide the identity of client by sending their identity through "Role" parameter. But this is very weak since anyone could fake their identities by putting whatever they want into "Role" parameter.

In this task we will build up stronger identity recognition for clients by using certificates. In balance transfer example, Certificate Authorities are already built for each organization. Besides, there is also a node.js server provided which you can use to interact with ledger and certificate authorities.

Task 3.1: Send Enrollment Requests (10 points) Clients will send enrollment request through node API, an example of how to use node APIs is located at

```
~/go/fabric-samples/balance-transfer/testAPI.sh
```

The http request regarding enrollment is in line 55-75. You can follow the format and change the "username" field to be the role name in supply chain example such as "Supplier.supplier1". The output of this request is a token which you can use to authenticate other transactions.

Task 3.2: Design Access Control on Chaincode Level (10 points) You can get the certificates of client through chaincode API "GetCreator()" in chaincode. Once you get the real identity of clients, you can design your own access control logic using chaincode(e.g. Only Supplier can AddComponent()).

3.4 Task 4 (bonus): Setting up Endorsement Policies (25 points)

In this task, you will learn how to set up endorsement policies to enforce business logic. Every chaincode has an endorsement policy which specifies the set of peers on a channel that must execute chaincode and endorse the execution results in order for the transaction to be considered valid. These endorsement policies define the organizations (through their peers) who must endorse (i.e., approve of) the execution of a proposal.

Endorsement policies can be used to enforce business logic. For example, in the supply chain use case where we have three endorsing peers Supplier, Manufacturer and Dealer, if a client send a AddComponet() transaction, only supplier endorsing peers has enough background info to tell if such a transaction is valid or not. It is because only the supplier knows the fact if the physical component is produced and added to blockchain, manufacturer and dealer have no knowledge about it and therefore they should not have privilege to endorse AddComponent() transactions.

So you will setup correct endorsement policies to enforce these business logic. In this task we assume there are 2 endorsing peers: Supplier, Manufacturer(Since in balance transfer example we only have 2 organizations, we can use one organization as Supplier, the other Manufacturer. To make this task easier you don't need to deal with Dealer and operations regarding Dealer). And some sample business logic are as follows:

- Only Supplier peer can endorse AddComponent().
- Only the sender and receiver of an transfer process can endorse TransferComponent().
- Only Manufacturer peer can endorse MountComponent(), ReplaceComponent(), RecallComponent().

Task 4.1: Split Chaincodes As you can see now we need more than 1 endorsement policy to fulfill these requirements, but one chaincode could only be assigned one endorsement policy. So you need to split the chaincode into multiple chaincodes based on the endorsement policies.(e.g. There should be one chaincode which only contains AddComponent() with endorsement policy "ONLY Supplier").

It is your decision how to put different operations into different chaincodes as long as the previous business logic is satisfied.

Task 4.2: Setup Chaincode Endorsement policy is setted when chaincode is instantiated. There are two ways to achieve that:

- Through node API, the node.js server in balance transfer does NOT support setting endorsement policies, so you need to change the code of node.js server (mainly /app/instantiate-chaincode.js) to add endorsement policy as a parameter. Then you can send http request to node server to instantiate chaincode with endorsement policy.
- Through "peer chaincode instantiate" command. With this method you should be careful about all the flags that this command requires. Endorsement policy is also one of the flag. For more knowledge please refer to Hyperledger Official Document.

NOTE: This is the link describing the structure of chaincode instantiate request. You can change the code in instantiate-chaincode.js based on this structure.

`https://fabric-sdk-node.github.io/global.html#ChaincodeInstantiateUpgradeRequest`