BLOCKCHAIN AS A SERVICE DEPLOYING AN END-TO-END APPLICATION WITH HYPERLEDGER FABRIC CLI

GUILLAUME Lasmayous
guillaume.lasmayous@fr.ibm.com

Table of Contents

At the request of INET and their clients, this document explores an alternative way to administer a Hyperledger Fabric network provisioned with the Blockchain-as-a-Service asset. This alternate method is using the command line interface provided by the Hyperledger Fabric community.

Using an IBM-written sample application called Marbles, this document will take the reader through all the steps required to deploy and configure an Hyperledger Fabric network backend, install and configure the frontend and exercise the application.

This document is by no means a replacement for the user and administration guide of the Blockchain-as-a-Service software. It is intended to document an alternative administration method to perform certain tasks without the User Interface.

Please note: in the following sections, the terms smart contract and chaincode will be used interchangeably to refer to the program that is deployed on the Fabric infrastructure to interact with the ledger.

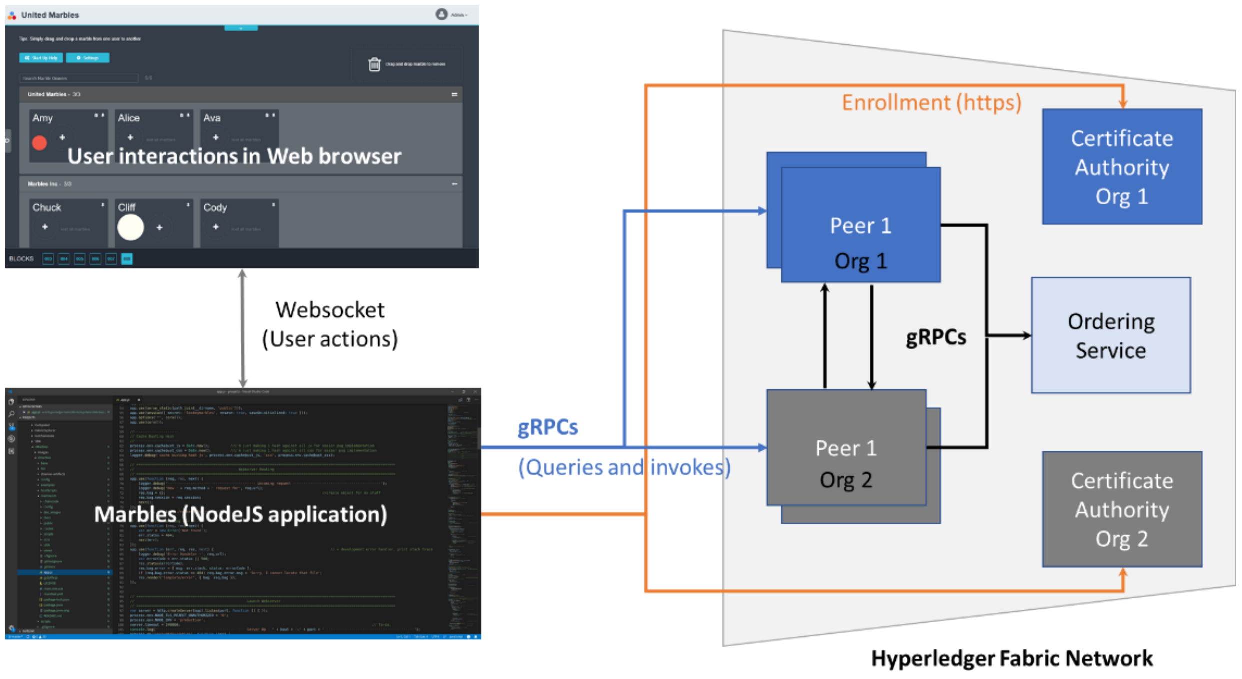
1. Introducing Marbles

The present document will take the reader through the necessary steps to deploy a sample IBM-written application using a Hyperledger Fabric blockchain network. This application showcases the transfer of marbles between different companies, keeping track of the ownership changes in a Fabric blockchain.

The application frontend is written in Javascript, deployed in a NodeJS server. The backend is a Hyperledger Fabric network running a smartcontract written in Golang.

1.1. Application architecture

The diagram below shows the high-level architecture of the application and the flow of actions.



User interactions in Web browser
Websocket
(User actions)
Marbles (NodeJS application)
gRPCs
(Queries and invokes)
Enrollment (https)
Certificate Authority Org 1
Peer 1
Org 1
gRPCs
Ordering Service
Peer 1
Org 2
Certificate Authority Org 2
Hyperledger Fabric Network

1. The admin will interact with Marbles, our Node.js application, in their browser. See section Use the application for more details.
2. This client-side JS code will open a WebSocket to the backend Node.js application. The client JS will send messages to the backend when the admin interacts with the site. See section Install the NodeJS runtime to install and configure the frontend application.
3. Reading or writing the ledger is known as a proposal. This proposal is built by Marbles (via the SDK) and then sent to a blockchain peer.

4. The peer will communicate to its Marbles chaincode container. The chaincode will run/simulate the transaction. If there are no issues it will endorse the transaction and send it back to our Marbles application. Refer to section Install chaincode and following for more information about chaincode installation and instantiation mechanisms.
5. Marbles (via the SDK) will then send the endorsed proposal to the ordering service. The orderer will package many proposals from the whole network into a block. Then it will broadcast the new block to peers in the network.
6. Finally, the peer will validate the block and write it to its ledger. The transaction has now taken effect and any subsequent reads will reflect this change

1.2. Get the Marbles sample application source code

On the LinuxONE platform, the Marbles application is maintained by Barry Siliman, a member of the IBM Washington Systems Center in the US, in a fork of the original IBM-Blockchain/marbles repository found at this address: https://github.com/silliman/2019FastStart .

In the rest of this document, it is assumed the zmarbles.tar.gz archive has been retrieved from this repository and extracted into ${PWD}/src.

2. Prerequisites

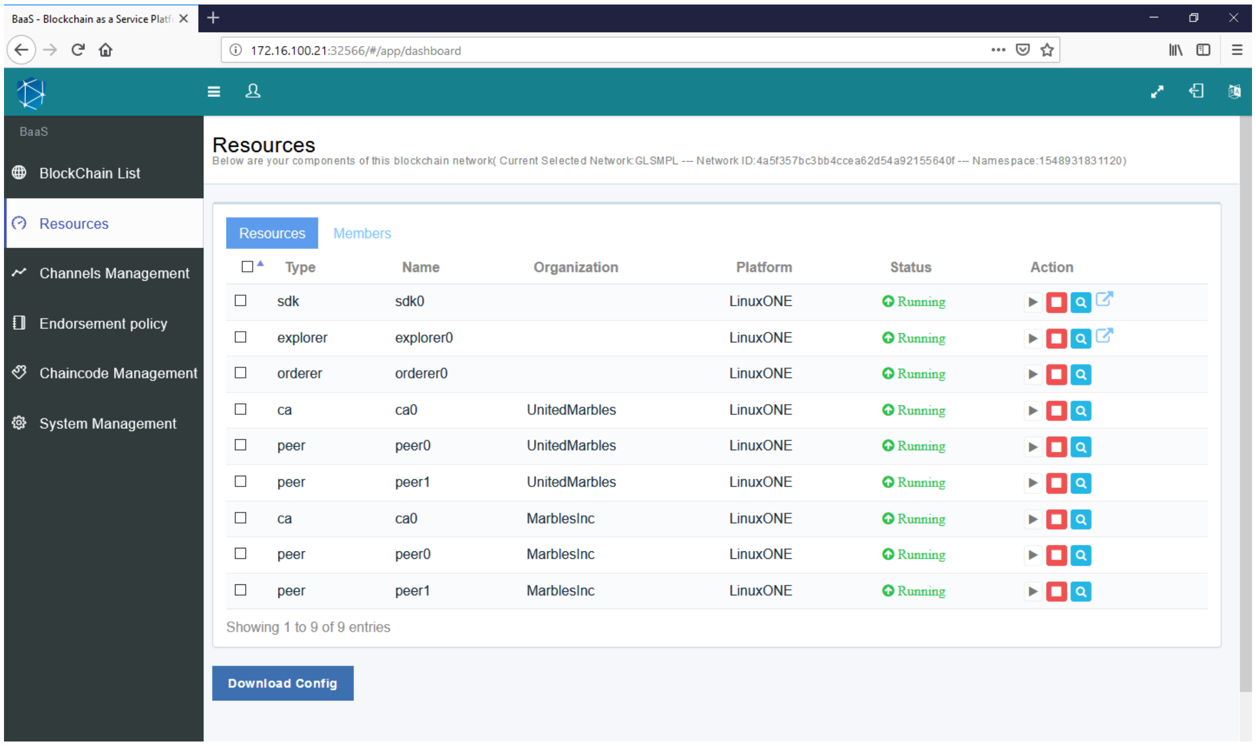
This section describes the prerequisites needed for the rest of this document.

2.1. Provision a Hyperledger Fabric blockchain network using the BCaaS graphical user interface

Because it is far easier than doing it manually, the Fabric network is deployed using the BCaaS user interface. This tool allows to deploy a fully configured, functional Hyperledger Fabric network.

However, detailing the deployment of a Hyperledger Fabric network using the BCaaS User Interface is behond the scope of this document. Please refer to the BCaaS User's Guide for details.

In the following sections, a simple, 2 organizations, Hyperledger Fabric network will be used. The provisioned network resources are displayed below in the Resources panel, as depicted below.



Resources

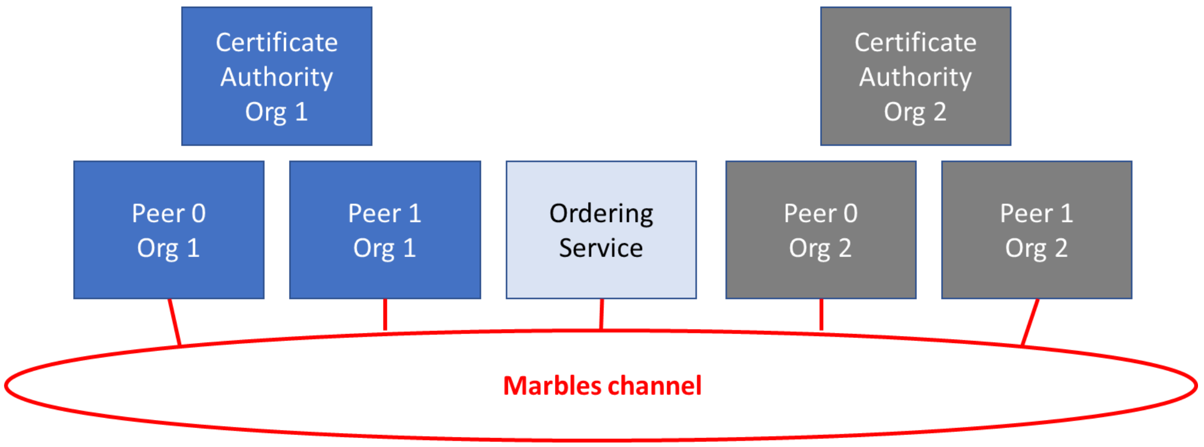
Below are your components of this blockchain network(Current Selected Network:GLSMPL --- Network ID:4a5f357bc3bb4ccea62d54a92155640f --- Namespace:1548931831120)

Type	Name	Organization	Platform	Status	Action
sdk	sdk0		LinuxONE	Running	
explorer	explorer0		LinuxONE	Running	
orderer	orderer0		LinuxONE	Running	
ca	ca0	UnitedMarbles	LinuxONE	Running	
peer	peer0	UnitedMarbles	LinuxONE	Running	
peer	peer1	UnitedMarbles	LinuxONE	Running	
ca	ca0	MarblesInc	LinuxONE	Running	
peer	peer0	MarblesInc	LinuxONE	Running	
peer	peer1	MarblesInc	LinuxONE	Running	

Showing 1 to 9 of 9 entries

The organizations have been named, for the sake of the example, UnitedMarbles and MarblesInc. However, internally, the organizations are named org1 and org2. This topic will be discussed in more details later in this document.

Once the network deployment is completed, a channel named 'Marbles' also has been created using the user interface, and all 4 peers have joined this channel, like so:



2.2. Get and configure the tools

The Hyperledger Fabric project provides a set of command-line administration tools to interact with an existing Fabric network, regardless of the mean used to deploy this network. These command-line tools can be used to administer Hyperledger Fabric networks created using the BCaaS tool on your premises for instance, but also networks running in the IBM Cloud or simple testing environments on laptops.

It is important to remember the command-line tools version need to match the Fabric version they will be used with. That is, in the case of the BCaaS, the command-line tools need to be at version 1.1, since the Fabric network deployed in previous section is in version 1.1.0. Failure to do so can lead to unexpected results and inconsistent behaviors.

The command-line utility programs are provided in binary format by the Hyperledger project. They only need to be fetched once and can be reused with as many networks as needed. The tools are described in the next section.

2.2.1. Get the tools

The Hyperledger project regularly provides a packaged set of binaries that can be grabbed using a shell script as explained here in Fabric documentation (https://hyperledger-fabric.readthedocs.io/en/release-1.1/getting_started.html#install-binaries-and-docker-images).

We would recommend getting the script first and executing it after inspection.

```
curl -sSL https://goo.gl/6wtTN5 -o bootstrap.sh
```

The goo.gl address is in fact a shortened URL directing to the Hyperledger Fabric Github repo, where the bootstrap.sh script can be found at: https://github.com/hyperledger/fabric/blob/master/scripts/bootstrap.sh

The bootstrap.sh script accepts several parameters, as shown below:

```
$ bash bootstrap.sh -h
Usage: bootstrap.sh [version [ca_version [thirdparty_version]]] [options]

options:
-h : this help
-d : bypass docker image download
-s : bypass fabric-samples repo clone
-b : bypass download of platform-specific binaries

e.g. bootstrap.sh 1.4.0 -s
would download docker images and binaries for version 1.4.0
```

Only the Fabric binaries are required, so the script should be used as follows:

```
$ bash bootstrap.sh 1.1.1 1.1.0 0.4.6 -d -s
```

This will download the Fabric binaries in version 1.1.1, and Fabric-CA binaries in version 1.1.0, to match the version of Hyperledger Fabric deployed previously. The last parameter refers to the version of the 3rd-party Docker images (Zookeeper, Kafka, CouchDB) but is unused in this case.

The result of the execution of the script is shown below:

```
$ tree .
.
├── bin
│   ├── configtxgen
│   ├── configtxlator
│   ├── cryptogen
│   ├── fabric-ca-client
│   ├── get-docker-images.sh
│   ├── orderer
│   └── peer
├── bootstrap.sh
└── config
    ├── configtx.yaml
    ├── core.yaml
    └── orderer.yaml

2 directories, 11 files
```

There are 5 binaries and a script part of the Hyperledger 1.1 binaries release:

- configtxgen: the tool used to generate the network's configuration artifacts – genesis block, channel configuration transaction and anchor peers' configuration transactions. This is done automatically as part of the Blockchain network creation process.
  Once deployed, the artifacts and configuration files can be downloaded from the BCaaS UI for reuse.
- configtxlator: a tool used to dynamically modify the configuration of the blockchain network.
- cryptogen: the tool used to generate the required cryptographic material required for Hyperledger Fabric to start. This is also done automatically as part of the provisioning process. The generated digital certificates will be retrieved in the same tarball file as the artifacts and configuration files, from the BCaaS UI.
- fabric-ca-client: the administration tool for Fabric-CA server.

- get-docker-images.sh: shell script to retrieve the necessary Docker images from the Docker Hub. It will not be used; the Docker images are already available as part of the BCaaS.
- orderer: the command used in the orderer pods. It will not be used for Hyperledger Fabric administration.
- peer: the command, which is launched in the peer pods in the BCaaS, which can also be used as a client for Hyperledger Fabric. That is how the tool will be used in the upcoming sections.

The config folder holds sample configuration files for:

- configtxgen (in configtx.yaml)
- the peer (in core.yaml)
- the ordering service (in orderer.yaml)

2.2.2. Update PATH

For the Fabric binaries to be found, it is required to update the PATH variable to add the folder just created by the bootstrap.sh script:

```
admin1@inetlx2:~/gls$ export PATH=${PWD}/bin:$PATH
admin1@inetlx2:~/gls$ echo $PATH
/home/admin1/gls/bin:/home/admin1/bin:/home/admin1/.local/bin:/usr/local/sbin:/usr/loc
al/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

Make sure the PATH value has been updated correctly by trying to execute the peer command:

```
admin1@inetlx2:~/gls$ peer
2019-01-17 22:00:20.178 +07 [main] main -> ERRO 001 Fatal error when initializing core
config : error when reading core config file: Unsupported Config Type ""
```

At this stage, the ‘peer’ command is found but needs some configuration.

2.3. Gather blockchain network configuration information

For the peer client to operate properly, it requires an understanding of the blockchain network topology and configuration, which includes:

- the peer(s) to address and the port(s) to connect to (PEER_ADDRESS)
- the Membership Service Provider they belong to and its ID (LOCALMSPID), and where the cryptographic material is located (MSPCONFIGPATH)
- the channel the peer(s) is(are) connected to

The following sections will describe how to get this information.

2.3.1. Identify the peers to address.

To interact with the deployed Hyperledger Fabric network, the peer client or the NodeJS SDK (as well as any other SDKs, for that matter) need the hostnames of the components to reach, and the port to connect to.

In the BaaS environment, the components' names are built around the Kubernetes namespace, appended to standardized hostnames. The ports are automatically assigned by the underlying Kubernetes infrastructure.

2.3.1.1. Standardized hostnames

The deployed Hyperledger Fabric network uses pre-defined, standardized component names using the following convention:

- organizations are named orgX, with X starting at 1 and increasing
- orderers are named ordererX, with X starting at 0
- certificate authorities are named caX, with X starting at 0
- peers are named peerX, with X starting at 0

To identify which components belongs to which organization, in configuration files, the components are suffixed with the organization they belong to, like so: peer0-org1, peer1-org2, ca0-org1, ca0-org2…

2.3.1.2. Kubernetes namespace

To distinguish between different Hyperledger Fabric networks managed by the BCaaS tool, each network is deployed in its own, private, Kubernetes namespace. This namespace is identified by a 13-digit number which can be found in the upper-right end of the Resources panel, as shown below. In this example, the Kubernetes namespace ID is 1548931831120.

The fully qualified domain name of the component can be built using the standardized hostnames suffixed with the domain name in the format <Kubernetes namespace ID>.svc.cluster.local. These are the names shown in the configuration files.

2.3.1.3. Kubernetes services

Kubernetes will automatically map the ports used by the technical components (peers, orderers, CAs) to available ports on each of the nodes used to deploy the Kubernetes Pods. The Kubernetes command '*kubectl -n <namespace ID> get svc*' can be used to retrieve the mapped ports.

```
root@inetlx2:~# kubectl -n 1548931831120 get svc
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                                          AGE
ca0-org1      NodePort    10.99.101.218    <none>        7054:32355/TCP                                   5h
ca0-org2      NodePort    10.109.177.164   <none>        7054:31063/TCP                                   5h
explorer0     NodePort    10.107.111.121   <none>        8080:32093/TCP                                   5h
orderer0      NodePort    10.106.212.170   <none>        7050:32039/TCP                                   5h
peer0-org1    NodePort    10.104.116.205   <none>        7051:31355/TCP,7052:32560/TCP,7053:31952/TCP     5h
peer0-org2    NodePort    10.101.53.191    <none>        7051:32409/TCP,7052:30258/TCP,7053:31915/TCP     5h
peer1-org1    NodePort    10.104.245.21    <none>        7051:30218/TCP,7052:31356/TCP,7053:31479/TCP     5h
peer1-org2    NodePort    10.97.9.151      <none>        7051:30667/TCP,7052:30886/TCP,7053:30557/TCP     5h
sdk0          NodePort    10.111.75.33     <none>        3000:32681/TCP                                   5h
```

Note: the *kubectl -n <namespace ID> get svc* command needs to be run as root.

The ports assigned to the peers have been collected in the table below:

Component	Standart port	Port assigned
ca0-org1	7054	32355
ca0-org2	7054	31063
peer0-org1	7051 (GRPC)	31355
	7053 (Events)	31952
peer1-org1	7051 (GRPC)	30218
	7053 (Events)	31479
peer0-org2	7051 (GRPC)	32409
	7053 (Events)	31915
peer1-org2	7051 (GRPC)	30667
peer1-org2	7053 (Events)	30557
orderer0	7050	32038

2.3.2. Membership Service Provider

Strong identity management is the cornerstone of Hyperledger Fabric. Identity in Fabric is provided through an abstraction called an MSP, Membership Service Provider. This MSP is used to provide a set of x509.v3 digital certificates, which are used by all Fabric components to:

- authenticate users, client applications, peers or orderers
- sign proposals, transactions and blocks
- verify signatures

Each component manages its own local MSP (a set of digital certificates), and Fabric maintains a channel MSP to allow for cross organization signatures verification.

In Fabric, the local MSPs are identified by a Membership Service Provider Identifier, or MSPID. In BCaaS, the MSP Identifiers always have the form orgXMSP, with X starting at 1, regardless of the name given to the organization in the UI.

Once the network is deployed, the local MSPs and configuration files are packaged in a tarball which can be retrieved from the Resource List panel by clicking on the "Download Config" button. Once extracted, the cryptographic material can be found in the crypto-confiig subfolder.

Example below shows the content of the local MSP maintained by the peers in organization 1.

```
admin1@inetlx2:~/gls/GLSMPL/crypto-config$ tree peerOrganizations/
peerOrganizations/

This is the peer organization referred to as org1, which MSPID is org1MSP.


|-- org1.1548931831120.svc.cluster.local
|   |-- ca
|   |   |-- ca-org1.1548931831120.svc.cluster.local-cert.pem
|   |   `-- e4b161b5e84a50e516afd149bfb3725b5f94e054e316a317144389fb8122ca2c_sk
```

The CA folder holds the private (e4b161b5e84a50e516afd149bfb3725b5f94e054e316a317144389fb8122ca2c_sk) and public key of the root certificate of the CA. The private key will be used to sign all subsequent certificates requests. The public key is shared across the organization to check for the signatures of the certificates.

```
|   |-- msp
|   |   |-- admincerts
|   |   |   `-- Admin@org1.1548931831120.svc.cluster.local-cert.pem
|   |   |-- cacerts
|   |   |   `-- ca-org1.1548931831120.svc.cluster.local-cert.pem
|   |   `-- tlscacerts
|   |       `-- tlsca-org1.1548931831120.svc.cluster.local-cert.pem
```

This is the MSP of the organization. Admincerts holds a signed certificate of a user who is the administrator of the organization. Cacerts holds the public root certificate of the organization’s CA (as found in the above folder), tlscacerts has the public root certificate of the CA used to issue TLS certificates.

```
|   |-- peers
|   |   |-- peer0-org1.1548931831120.svc.cluster.local
|   |   |   |-- msp
|   |   |   |   |-- admincerts
|   |   |   |   |   `-- Admin@org1.1548931831120.svc.cluster.local-cert.pem
|   |   |   |   |-- cacerts
|   |   |   |   |   `-- ca-org1.1548931831120.svc.cluster.local-cert.pem
|   |   |   |   |-- keystore
|   |   |   |   |   `--
3638cb58cbfb8c76994214bd60b2052b56f6db849cc8dfb2cd476bfade1d797f_sk
|   |   |   |   |-- signcerts
|   |   |   |   |   `-- peer0-org1.1548931831120.svc.cluster.local-cert.pem
|   |   |   |   `-- tlscacerts
|   |   |   |       `-- tlsca-org1.1548931831120.svc.cluster.local-cert.pem
```

This is the local MSP for organization 1’s peer0. Admincerts forlder holds the signed certificate of an admin for the peer. Cacerts holds the public root certificate of organization 1’s CA. keystore has the private key of the peer. Signcerts holds the public key of the peer signed by org1’s CA. tlscacerts has the public root certificate of the CA used to issue TLS certificates.

```
|   |   |   `-- tls
|   |   |       |-- ca.crt
|   |   |       |-- server.crt
|   |   |       `-- server.key
```

This folder holds the certificates used by organization 1’s peer0 to establish secure, TLS, connections.
ca.crt is the public certificate of the CA which issues the TLS certificates, server.crt is the public key of the peer to use for TLS, server.key is the private key of the peer used for TLS.

```
[…] peer1 local MSP goes here
```

```
|   |-- tlsca
|   |   |-- e40a24a7e6382c7c0a3cb42d4d6b7f007d9b9bbaf0b09be9914394b54e2cff4f_sk
|   |   `-- tlsca-org1.1548931831120.svc.cluster.local-cert.pem

This is the private and public key of the CA used to issue TLS certificates.
Below is the MSP for the users defined in organization org1, following the same
structure.


|   `-- users
|       |-- Admin@org1.1548931831120.svc.cluster.local
|       |   |-- msp
|       |   |   |-- admincerts
|       |   |   |   `-- Admin@org1.1548931831120.svc.cluster.local-cert.pem
|       |   |   |-- cacerts
|       |   |   |   `-- ca-org1.1548931831120.svc.cluster.local-cert.pem
|       |   |   |-- keystore
|       |   |   |   `--
e7d64967d613669f2d251a34e3295961df37a0c484b383098604b374eff75f37_sk
|       |   |   |-- signcerts
|       |   |   |   `-- Admin@org1.1548931831120.svc.cluster.local-cert.pem
|       |   |   `-- tlscacerts
|       |   |       `-- tlsca-org1.1548931831120.svc.cluster.local-cert.pem
|       |   `-- tls
|       |       |-- ca.crt
|       |       |-- client.crt
|       |       `-- client.key
|       `-- User1@org1.1548931831120.svc.cluster.local
|           |-- msp
|           |   |-- admincerts
|           |   |   `-- User1@org1.1548931831120.svc.cluster.local-cert.pem
|           |   |-- cacerts
|           |   |   `-- ca-org1.1548931831120.svc.cluster.local-cert.pem
|           |   |-- keystore
|           |   |   `--
54534cb19c5afd4312509452c2bbe9f2a771f9d1259a8eaad2ed2b59542a1b51_sk
|           |   |-- signcerts
|           |   |   `-- User1@org1.1548931831120.svc.cluster.local-cert.pem
|           |   `-- tlscacerts
|           |       `-- tlsca-org1.1548931831120.svc.cluster.local-cert.pem
|           `-- tls
|               |-- ca.crt
|               |-- client.crt
|               `-- client.key
```

2.3.3. Channel

As mentioned in section 2.1, a channel named marbles has been created using the UI, and all 4 peers have joined this channel.

3. Deploy Marbles smartcontract using Hyperledger Fabric command-line.

The next chapter will describe the necessary required configuration steps for the peer tool, as well as operation of the Fabric network.

3.1. Configure the peer client

The section describes the steps required to configure the peer client to connect to any of the Hyperledger Fabric networks deployed using the BCaaS interface.

3.1.1. Configuration file location

The peer Fabric client looks for a configuration file *core.yaml* in the following places by default:

- the same folder as the peer binary
- in a folder in Hyperledger Fabric source code repository in the GOPATH folder (${GOPATH}src/github.com/hyperledger/fabric/sampleconfig)
- in any folder specified using the FABRIC_CFG_PATH environment variable

To use the configuration file installed as part of the bootstrap process, we exported the FABRIC_CFG_PATH environment variable and made it point to the '*/home/admin1/gls/config*' as shown below:

```
admin1@inetlx2:~/gls$ export FABRIC_CFG_PATH=/home/admin1/gls/config/

admin1@inetlx2:~/gls$ peer

2019-01-17 22:34:31.416 +07 [main] main -> ERRO 001 Cannot run peer because cannot
init crypto, missing /home/admin1/gls/config/msp folder
```

The error message is expected though, as peer needs some cryptographic material to operate properly. The MSP and TLS communications will be configured in the next section.

3.1.2. Configure TLS and MSP

3.1.2.1. Default configuration override

The configuration file includes directives instructing the client about the local MSP ID to use to connect, the MSP configuration path where to look for digital credentials, which peer to connect to….

One can update the file and use the peer command without any further configuration. However, to allow quickly switching between peers and organizations, we find it easier to override the *core.yaml* default values with environment variables rather than managing different sets of configuration files.

A utility script, ‘setpeer’ has been written to simplify the switch between environments

3.1.2.2. setpeer utility script

The setpeer utility script is a very simple that takes 3 parameters, in that order:

- the Kubernetes namespace ID allocated by the BCaaS when creating the network. See section 2.3.1 for details about how to retrieve it
- the organization ID to work with (value between 1 and 5). See section 2.3.2.
- the peer to target (0 or 1). See section 2.3.1 for details.

Before use, the content of the NETWORK_NAME variable needs to be updated to refer to the actual network name, as specified in the ‘Blockchain List’ panel.

Called with the right arguments, the setpeer script will export all required TLS and MSP related variables in a single command.

```
admin1@inetlx2:~/gls$ cat setpeer

#!/bin/bash

NETWORK_NAME=GLS
NS=$1
ORG=$2
PEER=$3

export PATH=/home/admin1/gls/bin:$PATH
export FABRIC_CFG_PATH=/home/admin1/gls/config/
export CORE_PEER_TLS_ROOTCERT_FILE=/home/admin1/gls/${NETWORK_NAME}/crypto-
config/peerOrganizations/org${ORG}.${NS}.svc.cluster.local/peers/peer${PEER}-
org${ORG}.${NS}.svc.cluster.local/tls/ca.crt
export CORE_PEER_TLS_CERT_FILE=/home/admin1/gls/${NETWORK_NAME}/crypto-
config/peerOrganizations/org${ORG}.${NS}.svc.cluster.local/peers/peer${PEER}-
org${ORG}.${NS}.svc.cluster.local/tls/server.crt
export CORE_PEER_TLS_KEY_FILE=/home/admin1/gls/${NETWORK_NAME}/crypto-
config/peerOrganizations/org${ORG}.${NS}.svc.cluster.local/peers/peer${PEER}-
org${ORG}.${NS}.svc.cluster.local/tls/server.key
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID=org${ORG}MSP
export CORE_PEER_MSPCONFIGPATH=/home/admin1/gls/${NETWORK_NAME}/crypto-
config/peerOrganizations/org${ORG}.${NS}.svc.cluster.local/users/Admin@org${ORG}.${NS}
.svc.cluster.local/msp
#export CORE_PEER_ADDRESS=peer${PEER}:32762
export FABRIC_TLS="--tls --cafile /home/admin1/gls/${NETWORK_NAME}/crypto-
config/ordererOrganizations/${NS}.svc.cluster.local/orderers/orderer0.${NS}.svc.cluste
r.local/msp/tlscacerts/tlsca.${NS}.svc.cluster.local-cert.pem"
env |grep CORE
```

Please note the setpeer script is provided as-is. IBM holds no responsibility for its use whatsoever.

3.1.2.3. Transport Layer Security configuration

The networks deployed using the BCaaS interface implement Transport Layer Security to secure the communication between the different technical components.
It is therefore necessary to override the defaults found in *core.yaml* in the tls section by populating the following configuration variables:

- CORE_PEER_TLS_ENABLED: needs to be set to true. Be aware that ‘yes’ is NOT an accepted value.
- CORE_PEER_TLS_ROOTCERT_FILE: needs to point to the root certificate of the CA used to sign the certificates used to establish the TLS connection.
- CORE_PEER_TLS_CERT_FILE: points to the location of the x509.3 certificate used by the server for TLS communications.
- CORE_PEER_TLS_KEY_FILE: point to the private key of the server for TLS communications.
- FABRIC_TLS is a convenience variable that instructs the peer command to use TLS for communication and identifies the root CA certificate used to generate the TLS certificate for the orderers.

```
CORE_PEER_TLS_ROOTCERT_FILE=/home/admin1/gls/GLSMPL/crypto-
config/peerOrganizations/org1.1548931831120.svc.cluster.local/peers/peer0-org1.
1548931831120.svc.cluster.local/tls/ca.crt

CORE_PEER_TLS_KEY_FILE=/home/admin1/gls/GLSMPL/crypto-
config/peerOrganizations/org1.1548931831120.svc.cluster.local/peers/peer0-
org1.1548931831120.svc.cluster.local/tls/server.key

CORE_PEER_TLS_CERT_FILE=/home/admin1/gls/GLSMPL/crypto-
config/peerOrganizations/org1.1548931831120.svc.cluster.local/peers/peer0-
org1.1548931831120.svc.cluster.local/tls/server.crt

CORE_PEER_TLS_ENABLED=true
```

3.1.2.4. Membership Services Provider configuration

In addition to those TLS-related statements, it is also necessary to set the MSP information:

- CORE_PEER_LOCALMSPID: sets the MSP Identifier to use when interacting with the Fabric
- CORE_PEER_MSPCONFIGPATH: is where the peer will find all required credentials to authenticate the participants, sign the requests and verify those signatures.

```
CORE_PEER_LOCALMSPID=org1MSP

CORE_PEER_MSPCONFIGPATH=/home/admin1/gls/GLSMPL/crypto-
config/peerOrganizations/org1.1548931831120.svc.cluster.local/users/Admin@org1.1548931
831120.svc.cluster.local/msp/
```

3.1.2.5. Command-line configuration

When using TLS to connect to the ordering service, it is also required to set the peer client options --tls and –cafile on the command line, which can be facilitated by, for instance, the following FABRIC_TLS environment variable:

```
FABRIC_TLS=--tls --cafile /home/admin1/gls/GLSMPL/crypto-
config/ordererOrganizations/1548931831120.svc.cluster.local/orderers/orderer0.15489318
31120.svc.cluster.local/msp/tlscacerts/tlsca.1548931831120.svc.cluster.local-cert.pem
```

3.1.2.6. Fabric Network configuration

The CORE_PEER_ADDRESS environment variable will tell the peer client which peer to target. Please note it is needed to specify the port to connect to, even if it is using the standard port.

```
CORE_PEER_ADDRESS=peer0:31355
```

When installing Golang chaincodes, the peer client looks for source code to copy to the peer in a folder relative to $GOPATH/src. Therefore, if the Golang development environment is not configured on the Hyperledger Fabric administration machine, it is required to set the GOPATH variable

```
GOPATH=/home/admin1/gls
```

3.2. Install chaincode

3.2.1. “peer chaincode install” command

The peer client provides a dedicated subcommand to install chaincodes onto peers. The chaincode needs to be installed on each peer belonging to a given channel.

The peer chaincode install takes 3 parameters:

- -n is the name of the smartcontract
- -v is the version of the smartcontract being deployed
- -p is the path to the smartcontract. In the case of Golang smartcontract, the path is relative to $GOPATH.

Below is an example of the complete command:

```
admin1@inetlx2:~/gls$ peer chaincode install -n marbles -v 1.0 -p
zmarbles/examples/chaincode/go/marbles
```

An interesting thing to note is that for the *peer chaincode install* command does not need to specify the $FABRIC_TLS environment variable. This is because this operation does not cause the peer to communicate with the orderer. Installing chaincode on a peer is a necessary step, but not the only step needed, to execute chaincode on that peer. The chaincode must also be instantiated on a channel that the peer participates in. You will do that in section 3.3.

3.2.2. Install the Marbles chaincode

Make sure the environment is set correctly, by looking at the values of PATH, FABRIC_CFG_PATH and GOPATH:

```
admin1@inetlx2:~/gls$ echo $PATH
/home/admin1/gls/bin:/home/admin1/bin:/home/admin1/.local/bin:/usr/local/sbin:/usr/loc
al/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

admin1@inetlx2:~/gls$ echo $FABRIC_CFG_PATH
/home/admin1/gls/config
```

```
admin1@inetlx2:~/gls$ echo $GOPATH
/home/admin1/gls

admin1@inetlx2:~/gls$ peer
2019-01-31 20:41:58.556 +07 [main] main -> ERRO 001 Cannot run peer because cannot
init crypto, missing /home/admin1/gls/config/msp folder
```

Update the CORE_PEER_ADDRESS environment to target the peer onto which installation should happen, using the information collected in section 2.3.1

```
admin1@inetlx2:~/gls$ export CORE_PEER_ADDRESS=peer0:31355
```

Use the setpeer to set the environment, after updating the path in which the configuration tarball has been extracted. Then proceed with the installation itself.

```
admin1@inetlx2:~/gls$ sed -i 's/GLS/GLSMPL/g' setpeer
```

For example, to install on the first peer (peer0) of organization UnitedMarbles (org1):

```
admin1@inetlx2:~/gls$ source ./setpeer 1548931831120 0 1
CORE_PEER_TLS_ROOTCERT_FILE=/home/admin1/gls/GLSMPL/crypto-
config/peerOrganizations/org1.1548931831120.svc.cluster.local/peers/peer0-
org1.1548931831120.svc.cluster.local/tls/ca.crt
CORE_PEER_TLS_KEY_FILE=/home/admin1/gls/GLSMPL/crypto-
config/peerOrganizations/org1.1548931831120.svc.cluster.local/peers/peer0-
org1.1548931831120.svc.cluster.local/tls/server.key
CORE_PEER_LOCALMSPID=org1MSP
CORE_PEER_TLS_CERT_FILE=/home/admin1/gls/GLSMPL/crypto-
config/peerOrganizations/org1.1548931831120.svc.cluster.local/peers/peer0-
org1.1548931831120.svc.cluster.local/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/home/admin1/gls/GLSMPL/crypto-
config/peerOrganizations/org1.1548931831120.svc.cluster.local/users/Admin@org1.1548931
831120.svc.cluster.local/msp
admin1@inetlx2:~/gls$ export CORE_PEER_ADDRESS=peer0:31355
admin1@inetlx2:~/gls$ peer chaincode install -n marbles -v1.0 -p
zmarbles/examples/chaincode/go/marbles
2019-01-31 21:14:13.658 +07 [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2019-01-31 21:14:13.658 +07 [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2019-01-31 21:14:13.883 +07 [main] main -> INFO 003 Exiting.....

admin1@inetlx2:~/gls$
```

The operation needs to be repeated on all the peers connect to the same channel which need access to this smartcontract.

The operation can be checked using the peer chaincode list subcommand, like so:

```
admin1@inetlx2:~/gls$ peer chaincode list --installed

Get installed chaincodes on peer:
Name: marbles, Version: 1.0, Path: zmarbles/examples/chaincode/go/marbles, Id:
c281c66f2b5012b6d5522b5390aca4db1bf35322db7bb0d20be6ccb8af62f08d
2019-02-01 18:10:02.384 +07 [main] main -> INFO 001 Exiting.....
```

3.3. Instantiate chaincode on channels

3.3.1. “peer chaincode instantiate” command

Chaincode installation is a peer-level operation. Chaincode instantiation, however, is a channel-level operation. It only needs to be performed once on the channel, no matter how many peers have joined the channel. Chaincode instantiation causes a transaction to occur on the channel, so even if a peer on the channel does not have the chaincode installed, it will be made aware of the instantiate transaction, and thus be aware that the chaincode exists and be able to commit transactions from the chaincode to the ledger- it just would not be able to endorse a transaction on the chaincode.

The peer chaincode instantiate requires 6 parameters:

- -o: specifies the ordering service to connect to.
- -n: is the name of the chaincode to instantiate
- -v: version of the smartcontract to instantiate
- -c: parameters to pass to the smartcontract for initialization. In this example, upon initialization, the ‘init’ method will be called with 1 as an argument.
- -P: specifies the endorsement policy for that smartcontract.
- -C: channel on to which instantiate the smartcontract.

Note in this case it is mandatory to use the $FABRIC_TLS variable, to ensure the communication with the ordering service endpoint is properly secured.

3.3.2. Instantiate the Marbles chaincode

In the case of the Marbles sample, the chaincode initialization method is called ‘init’, and takes just one argument (‘1’). The endorsement policy specifies that either a member of UnitedMarbles (organization 1), or a member of MarblesInc (organization 2) need to sign the transaction for it to be validated.

```
admin1@inetlx2:~/gls$ peer chaincode instantiate -o orderer0:32039 -n marbles -v1.0 -C
marbles -c '{"Args":["init","1"]}' -P 'OR("org1MSP.member","org2MSP.member")'
$FABRIC_TLS

2019-02-01 19:25:52.803 +07 [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using
default escc

2019-02-01 19:25:52.803 +07 [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using
default vscc

2019-02-01 19:26:35.236 +07 [main] main -> INFO 003 Exiting.....
```

Please note the instantiate may take a couple of minutes to complete. The reason behind this delay is that as part of the instantiate, a Docker image for the chaincode is created and then a Docker container is started from the image.

The logs of the peer during instantiation show what exactly happened:

```
2019-02-01 07:25:52.830 EST [dockercontroller] Start -> DEBU 583 Start container dev-
peer0-org1.1548931831120.svc.cluster.local-marbles-1.0

2019-02-01 07:25:52.831 EST [dockercontroller] getDockerHostConfig -> DEBU 584 docker
container hostconfig NetworkMode: bridge

2019-02-01 07:25:52.833 EST [dockercontroller] createContainer -> DEBU 585 Create
container: dev-peer0-org1.1548931831120.svc.cluster.local-marbles-1.0

2019-02-01 07:25:52.837 EST [dockercontroller] Start -> DEBU 586 start-could not find
image <dev-peer0-org1.1548931831120.svc.cluster.local-marbles-1.0-
77082ac68b368f49df97ba06ba45de7f243ab453c18fd050859578290dab908a> (container id <dev-
peer0-org1.1548931831120.svc.cluster.local-marbles-1.0>), because of <no such
image>...attempt to recreate image

2019-02-01 07:25:52.837 EST [chaincode-platform] generateDockerfile -> DEBU 587
FROM hyperledger/fabric-baseos:s390x-0.4.6
ADD binpackage.tar /usr/local/bin
LABEL org.hyperledger.fabric.chaincode.id.name="marbles" \
      org.hyperledger.fabric.chaincode.id.version="1.0" \
      org.hyperledger.fabric.chaincode.type="GOLANG" \
      org.hyperledger.fabric.version="1.1.0" \
      org.hyperledger.fabric.base.version="0.4.6"
ENV CORE_CHAINCODE_BUILDLEVEL=1.1.0

2019-02-01 07:25:52.840 EST [golang-platform] GenerateDockerBuild -> INFO 588 building
chaincode with ldflagsOpt: '-ldflags "-linkmode external -extldflags '-static'"'

2019-02-01 07:25:52.841 EST [golang-platform] GenerateDockerBuild -> INFO 589 building
chaincode with tags:

2019-02-01 07:25:52.841 EST [util] DockerBuild -> DEBU 58a Attempting build with image
hyperledger/fabric-ccenv:s390x-1.1.0

2019-02-01 07:26:33.780 EST [dockercontroller] deployImage -> DEBU 58b Created image:
dev-peer0-org1.1548931831120.svc.cluster.local-marbles-1.0-
77082ac68b368f49df97ba06ba45de7f243ab453c18fd050859578290dab908a

2019-02-01 07:26:33.780 EST [dockercontroller] Start -> DEBU 58c start-recreated image
successfully

2019-02-01 07:26:33.780 EST [dockercontroller] createContainer -> DEBU 58d Create
container: dev-peer0-org1.1548931831120.svc.cluster.local-marbles-1.0

2019-02-01 07:26:33.895 EST [dockercontroller] createContainer -> DEBU 58e Created
container: dev-peer0-org1.1548931831120.svc.cluster.local-marbles-1.0-
77082ac68b368f49df97ba06ba45de7f243ab453c18fd050859578290dab908a

2019-02-01 07:26:35.147 EST [dockercontroller] Start -> DEBU 58f Started container
dev-peer0-org1.1548931831120.svc.cluster.local-marbles-1.0

2019-02-01 07:26:35.149 EST [container] unlockContainer -> DEBU 590 container lock
deleted(dev-peer0-org1.1548931831120.svc.cluster.local-marbles-1.0)

2019-02-01 07:26:35.182 EST [dev-peer0-org1.1548931831120.svc.cluster.local-marbles-
1.0] func2 -> INFO 591 2019-02-01 12:26:35.178 UTC [shim] SetupChaincodeLogging ->
INFO 001 Chaincode (build level: 1.1.0) starting up ...
```

```
2019-02-01 07:26:35.183 EST [dev-peer0-org1.1548931831120.svc.cluster.local-marbles-
1.0] func2 -> INFO 592 2019-02-01 12:26:35.178 UTC [bccsp] initBCCSP -> DEBU 002
Initialize BCCSP [SW]

2019-02-01 07:26:35.183 EST [dev-peer0-org1.1548931831120.svc.cluster.local-marbles-
1.0] func2 -> INFO 593 2019-02-01 12:26:35.179 UTC [shim] userChaincodeStreamGetter ->
DEBU 003 Peer address: peer0-org1.1548931831120.svc.cluster.local:7052

2019-02-01 07:26:35.199 EST [dev-peer0-org1.1548931831120.svc.cluster.local-marbles-
1.0] func2 -> INFO 594 2019-02-01 12:26:35.197 UTC [shim] userChaincodeStreamGetter ->
DEBU 004 os.Args returns: [chaincode -peer.address=peer0-
org1.1548931831120.svc.cluster.local:7052]

2019-02-01 07:26:35.199 EST [dev-peer0-org1.1548931831120.svc.cluster.local-marbles-
1.0] func2 -> INFO 595 2019-02-01 12:26:35.197 UTC [shim] chatWithPeer -> DEBU 005
Registering.. sending REGISTER

2019-02-01 07:26:35.200 EST [accessControl] authenticate -> DEBU 596 Chaincode
marbles:1.0 's authentication is authorized
```

Once instantiated, the compiled smartcontract code is isolated in a dedicated Docker container (dev-peer0-org1-1548931831120.svc.cluster.local-marbles-1.0) with a point-to-point connection to the peer.

Instantiation status can be checked by running the peer chaincode list subcommand with –instantiated switch:

```
admin1@inetlx2:~/gls$ peer chaincode list --instantiated -C marbles
Get instantiated chaincodes on channel marbles:
Name: marbles, Version: 1.0, Path: zmarbles/examples/chaincode/go/marbles, Escc: escc,
Vscc: vscc
2019-02-01 19:41:13.517 +07 [main] main -> INFO 001 Exiting.....
```

3.4. Interact with chaincode using the peer command

The peer client also provides the invoke subcommand to call specific functions in the smartcontract from the command line. The functions and their arguments will vary from smartcontract to smartcontract. The example below shows how to invoke a function to create a new Marbles owner and assign a new Marbles to him / her.

```
admin1@inetlx2:~/gls$ peer chaincode invoke -o orderer0:32039 -n marbles -C marbles -c
'{"Args":["init_owner","o0000000000001","John","UnitedMarbles"]}' $FABRIC_TLS
2019-02-01 20:35:30.146 +07 [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using
default escc
2019-02-01 20:35:30.147 +07 [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using
default vscc
2019-02-01 20:35:30.188 +07 [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 003
Chaincode invoke successful. result: status:200
2019-02-01 20:35:30.188 +07 [main] main -> INFO 004 Exiting.....
```

The Marbles chaincode function called init_owner will create a new owner object in the ledger, with the following attributes:

- ID o0000000000001
- name John
- organization United Marbles.

Note the return status 200 in the output of the command, which is a success.

Creating a Marbles is as simple as calling the init_marble function with the right set of arguments, as shown below:

```
admin1@inetlx2:~/gls$ peer chaincode invoke -n marbles -c
'{"Args":["init_marble","m0000000000001","blue","35","o0000000000001","UnitedMarbles"]
}' $FABRIC_TLS -C marbles -o orderer0:32039
2019-02-01 20:47:35.387 +07 [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using
default escc
2019-02-01 20:47:35.387 +07 [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using
default vscc
2019-02-01 20:47:35.402 +07 [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 003
Chaincode invoke successful. result: status:200
2019-02-01 20:47:35.403 +07 [main] main -> INFO 004 Exiting.....
```

At this stage, the Marbles chaincode is ready to use. The last step is the deployment and configuration of the frontend application, written in Javascript with the Hyperledger Fabric Node SDK.

4. Frontend application deployment

Frontend applications usually interact with an Hyperledger Fabric network using a Software Development Kit, or SDK. The Hyperledger project maintains different SDKs for different programming languages to choose from:

- NodeJS (https://github.com/hyperledger/fabric-sdk-node)
- Java (https://github.com/hyperledger/fabric-sdk-java)
- Go (https://github.com/hyperledger/fabric-sdk-go)
- Python (https://github.com/hyperledger/fabric-sdk-py)
- REST (https://github.com/hyperledger/fabric-sdk-rest)

The Marbles sample application is written in Javascript, making use of the Hyperledger Fabric NodeJS SDK to interact with the blockchain network. As with many other applications, the deployment is a four-steps process:

- Install the NodeJS runtime
- Install the application and the required Node modules
- Configure the application
- Use!

4.1. Install the NodeJS runtime

The Marbles application has a prerequisite on Node version 8. Later versions are not supported by the application yet. The NodeJS package as found in Ubuntu is outdated, it is necessary to install the s390x binary provided by the NodeJS project from here: https://nodejs.org/dist/latest-v8.x/

At the time of writing, the latest version is 8.15.0.

Once downloaded, the package is extracted, and the bin subfolder added to the PATH:

```
admin1@inetlx2:~/gls$ tar -zxvf node-v8.15.0-linux-s390x.tar.gz
admin1@inetlx2:~/gls$ ls
GLSDEMO  bin  bootstrap.sh  config  connection_profile1.json   node-v8.15.0-linux-
s390x  node-v8.15.0-linux-s390x.tar.gz  setpeer  src  zmarbles.tar.gz

admin1@inetlx2:~/gls$ mv node-v8.15.0-linux-s390x node
admin1@inetlx2:~/gls$ export PATH=${PWD}/node/bin
```

4.2. Install the application

As a typical NodeJS application, Marbles includes a package.json file listing all the dependencies required for the application to behave as intended.

To install all the dependencies required by the application, run the '*npm install*' command from the zmarbles/marblesUI folder. This will fetch all required packages from '*registry.npmjs.org*' repository and install them locally. If a specific package is not available for the architecture where the application is installed, npm will build the package locally before installing it.

```
admin1@inetlx2:~/gls/src/zmarbles/marblesUI$ npm install
[…]
make: Leaving directory '/home/admin1/gls/src/zmarbles/marblesUI/node_modules/node-sass/build'
gyp info ok
Installed to /home/admin1/gls/src/zmarbles/marblesUI/node_modules/node-sass/vendor/linux-s390x-57/binding.node
added 723 packages from 414 contributors and audited 2975 packages in 256.328s
found 21 vulnerabilities (8 low, 4 moderate, 9 high)
  run `npm audit fix` to fix them, or `npm audit` for details


   ╭────────────────────────────────────────────────────────────────╮
   │                                                                │
   │      New minor version of npm available! 6.4.1 -> 6.7.0        │
   │   Changelog: https://github.com/npm/cli/releases/tag/v6.7.0    │
   │               Run npm install -g npm to update!                │
   │                                                                │
   ╰────────────────────────────────────────────────────────────────╯

admin1@inetlx2:~/gls/src/zmarbles/marblesUI$
```

NOTE: npm install command assumes a working Internet connection, as it will fetch some packages from the npmjs registry.

Marbles uses gulp as an automation tool to ensure all components and prerequisites are available and in the expected state. If not already, the gulp tool needs to be installed as well.

```
admin1@inetlx2:~/gls/src/zmarbles/marblesUI$ npm install gulp
npm WARN deprecated gulp-util@3.0.8: gulp-util is deprecated - replace it, following the guidelines at https://medium.com/gulpjs/gulp-util-ca3b1f9f9ac5
npm WARN deprecated graceful-fs@3.0.11: please upgrade to graceful-fs 4 for compatibility with current and future versions of Node.js
npm WARN deprecated minimatch@2.0.10: Please update to minimatch 3.0.2 or higher to avoid a RegExp DoS issue
npm WARN deprecated minimatch@0.2.14: Please update to minimatch 3.0.2 or higher to avoid a RegExp DoS issue
npm WARN deprecated graceful-fs@1.2.3: please upgrade to graceful-fs 4 for compatibility with current and future versions of Node.js
+ gulp@3.9.1
updated 1 package and audited 2975 packages in 153.604s
found 21 vulnerabilities (8 low, 4 moderate, 9 high)
  run `npm audit fix` to fix them, or `npm audit` for details
```

The ‘gulp’ command is a symbolic link to node_modules/gulp/bin/gulp.js.

4.3. Configure the application

The Marbles configuration files are in the marblesUI/config folder. There are 2 configuration files per organization:

- marblesX.json

- connection_profileX.json

(with X the organization number).

The marblesX.json file is primarily the configuration file for the frontend application itself, containing default information used to initialize the web application (port to bind, users to create upon startup). It also points to the connection profile to look up the connection details:

```
admin1@inetlx2:~/gls/src/zmarbles/marblesUI$ cat config/marbles1.json
{
    "cred_filename": "connection_profile1.json",
    "use_events": false,
    "keep_alive_secs": 120,
    "company": "United Marbles",
    "usernames": [
        "amy",
        "alice",
        "ava"
    ],
    "port": 3001,
    "last_startup_hash": ""
}
admin1@inetlx2:~/gls/src/zmarbles/marblesUI$
```

The second file, connection_profileX.json, is a standard Hyperledger Fabric connection profile. The connection profile definition is shared with Composer, which reference can be found here: https://hyperledger.github.io/composer/latest/reference/connectionprofile.

These connection profiles can be written in YAML (https://fabric-sdk-node.github.io/tutorial-network-config.html) or use the JSON format as the Marbles application does.

The connection_profileX.json includes the parameters needed to connect the application to the Hyperledger Fabric network. The file needs to be updated to match the characteristics of the network deployed in the BCaaS (bold attributes need to be adjusted):

```
{
    "name": "Docker Compose Network",
    "x-networkId": "not-important",
    "x-type": "hlfv1",
    "description": "Connection Profile for an IBM Blockchain Network",
    "version": "1.0.0",
    "client": {
        "organization": "org1MSP"
    },
    "channels": {
        "marbles": {
            "orderers": [
                "fabric-orderer"
            ],
            "peers": {
                "fabric-peer-org1": {
                    "x-chaincode": {}
                }
            },
            "chaincodes": [
                "marbles:2.0"
            ],
            "x-blockDelay": 1000
        }
    },
    "organizations": {
        "org1MSP": {
            "mspid": "org1MSP",
            "peers": [
                "fabric-peer-org1"
            ],
            "certificateAuthorities": [
                "fabric-ca-org1"
            ]
        }
    },
    "orderers": {
        "fabric-orderer": {
            "url": "grpcs://orderer0:32614",
            "grpcOptions": {
                "ssl-target-name-override":
"orderer0.1548174479236.svc.cluster.local",
                "grpc.http2.keepalive_time": 300,
                "grpc.keepalive_time_ms": 300000,
                "grpc.http2.keepalive_timeout": 35,
                "grpc.keepalive_timeout_ms": 3500
            },
            "tlsCACerts": {
                "path": "../../../../GLS/crypto-
config/ordererOrganizations/1548174479236.svc.cluster.local/orderers/orderer0.15481744
79236.svc.cluster.local/tls/ca.crt"
            }
        }
    },
    "peers": {
        "fabric-peer-org1": {
            "url": "grpcs://peer0:32762",
            "eventUrl": "grpcs://peer0:31382",
            "grpcOptions": {
               "ssl-target-name-override": "peer0-
org1.1548174479236.svc.cluster.local",
                "grpc.http2.keepalive_time": 300,
                "grpc.keepalive_time_ms": 300000,
                "grpc.http2.keepalive_timeout": 35,
                "grpc.keepalive_timeout_ms": 3500
            },
            "tlsCACerts": {
                "path": "../../../../GLS/crypto-
config/peerOrganizations/org1.1548174479236.svc.cluster.local/peers/peer0-
org1.1548174479236.svc.cluster.local/tls/ca.crt"
            }
        }
    },
    "certificateAuthorities": {
        "fabric-ca-org1": {
            "url": "https://ca-org1:31635",
            "httpOptions": {
                "ssl-target-name-override": "ca-org1.1548174479236.svc.cluster.local",
                "verify": true
            },
            "tlsCACerts": {
                "path": "../../../../GLS/crypto-
config/peerOrganizations/org1.1548174479236.svc.cluster.local/ca/ca-
org1.1548174479236.svc.cluster.local-cert.pem"
            },
            "registrar": [
                {
                    "enrollId": "admin",
                    "enrollSecret": "RzhejLYq"
                }
            ],
            "caName": ""
        }
    }
}
```

The only information that is unknown is the enrollSecret. That is the certificate authority's administrator password. This information can be found in the file called 'config.json' (found in the configuration files tarball downloaded from the BCaaS) like so:

```
admin1@inetlx2:~/gls/GLS$ cat config.json | python3 -m json.tool  | grep admin
                "auth": "admin:RzhejLYq",
                "auth": "admin:t92G81sn",
```

The first line is the administrator password for organization 1 certificate authority, the second line is the password for organization 2 certificate authority.

4.4. Use the application

The command to launch the application is 'gulp marblesX'. Upon startup, launch a web browser to connect to the NodeJS server on port 300X to start trading Marbles.

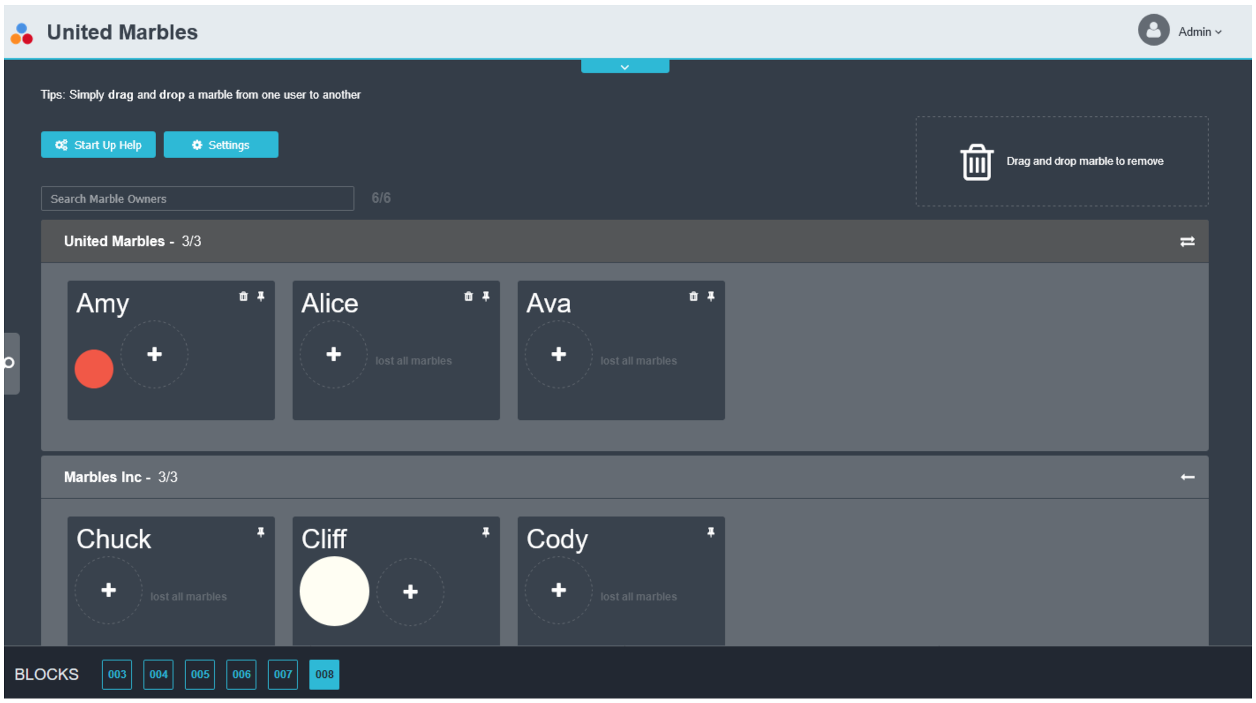
```
admin1@inetlx2:~/gls/src/zmarbles/marblesUI$ gulp marbles1
[17:15:00] Using gulpfile ~/gls/src/zmarbles/marblesUI/gulpfile.js
[17:15:00] Starting 'env_tls'...
[17:15:00] Finished 'env_tls' after 69 μs
[17:15:00] Starting 'build-sass'...
[17:15:00] Finished 'build-sass' after 13 ms
[17:15:00] Starting 'watch-sass'...
[17:15:00] Finished 'watch-sass' after 12 ms
[17:15:00] Starting 'watch-server'...
[17:15:00] Finished 'watch-server' after 3.91 ms
[17:15:00] Starting 'server'...
info: Checking connection profile is done
info: Loaded config file /home/admin1/gls/src/zmarbles/marblesUI/config/marbles1.json
info: Loaded connection profile file
/home/admin1/gls/src/zmarbles/marblesUI/config/connection_profile1.json



Connection Profile Lib Functions:()
```

```
  getNetworkName()
  getNetworkCredFileName()
  buildTlsOpts()
  getFirstChannelId()
  getChannelId()
  loadPem() […]

----------------------------- Chaincode found on channel "marbles" -------------------
----------

info: Checking chaincode and ui compatibility...
debug: [fcw] Querying Chaincode: read()
debug: [fcw] Sending query req: chaincodeId=marbles, fcn=read, args=[marbles_ui],
txId=null
warn: [fcw] warning - query resp is not json, might be okay: string 4.0.1
debug: [fcw] Successful query transaction.
info: Chaincode version is good
info: Checking ledger for marble owners listed in the config file

info: Fetching EVERYTHING...
debug: [fcw] Querying Chaincode: read_everything()
debug: [fcw] Sending query req: chaincodeId=marbles, fcn=read_everything, args=[],
txId=null
debug: [fcw] Peer Query Response - len: 371 type: object
debug: [fcw] Successful query transaction.
debug: This company has registered marble owners
debug: Looking for marble owner: amy
debug: Looking for marble owner: alice
debug: Looking for marble owner: ava
info: Everything is in place

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
debug: Detected that we have launched successfully before
debug: Welcome back - Marbles is ready
debug: Open your browser to http://localhost:3001 and login as "admin"
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

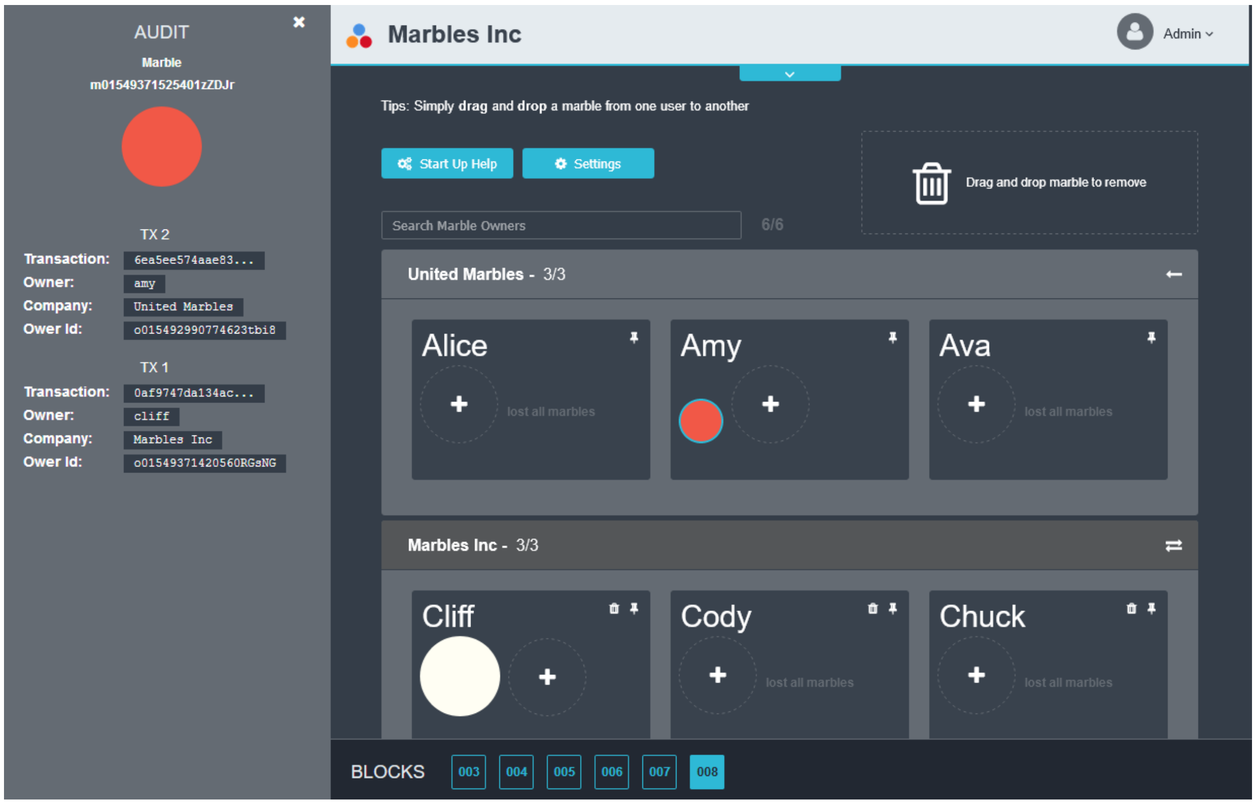
The Marbles application can be launched using the second configuration file, as organization 2.

Once the initial setup is done, the Marbles application user interface is brought up, allowing the users to create and trade Marbles. Each time a marble is created, a new block is cut, as well as when a marble changes ownership.

This is the Marbles User Interface for organization 1 (or “United Marbles”). Users of the same company can trade marbles among themselves or initiate a trade with a user from the other company.



However, it is not possible to initiate a trade from “Marbles Inc” to “United Marbles” from this panel. Such a trade needs to be initiated from the “Marbles Inc” panel:



AUDIT

Marble
m01549371525401zZDJr

TX 2

Transaction:	6ea5ee574aae83...
Owner:	amy
Company:	United Marbles
Ower Id:	o015492990774623tbi8

TX 1

Transaction:	0af9747da134ac...
Owner:	cliff
Company:	Marbles Inc
Ower Id:	o01549371420560RGsNG