SmartQuora

Table of Contents

Introduction

SmartQuora is an application that enables knowledge sharing among participants while incentivizing answers that are meaningful and well-explained. *Inquirers* pose questions with a reward for the best answers and a due-date by which they are looking for an answer. *Responders* compete with each other to provide the best answers. Participants can like or dislike answers. When the due-date arrives the answers are tallied and the reward is shared proportionately among the responders such that the best answers gets the most earnings. To avoid abuse of the platform, inquirers cannot answer their own questions and respondents cannot vote for their own answers.

Technically speaking, SmartQuora is a *DApp* (Decentralized Application) built on top of the HLF - Hyperledger Fabric Blockchain decentralized peer-to-peer network. It uses Smart Contracts built using HLF Composer API to represent Questions and Answers which contains rules to manage the process and payout.

SmartQuora uses a Javascript-based front-end web application to communicate withe the Blockchain platform on which the Smart Contracts reside using a RESTful interface. It uses Passport for authentication of participants using OAuth protocol and allows maintenance of their digital wallets through which the participants can manage their Digital Identities. These Digital Identities are generated and managed using the Hyperledger Fabric platform.

Please note that the intent behind this application was to demonstrate building Smart Contracts on the Hyperledger Fabric platform. The process documented here is not an exhaustive exercise in building a production quality application.

Audience

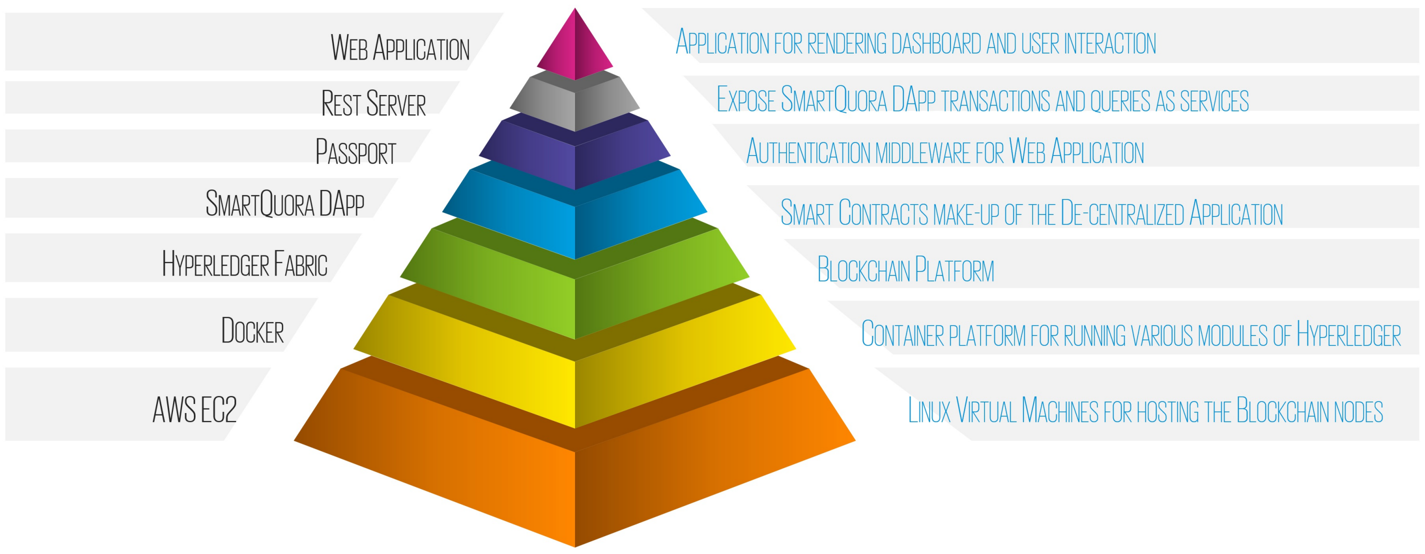
You are a Developer or Solutions Architect wanting to learn and build robust, secure and scaleable decentralized applications using open source framework such as Hyperledger Fabric to take full advantage of the autonomy that a blockchain protocol provides without a central point of failure.

As you are building this application, you will learn about the Hyperledger Fabric blockchain framework, Hyperledger Fabric Composer API, Passport authentication middleware, Docker containerization and various development scaffolding frameworks such as Yeoman, Loopback etc. discussed below.

Architecture

The Hyperledger Fabric blockchain platform that SmartQuora uses for this demonstration is built on top of AWS EC2 platform. It uses Docker containers to host various parts of the Hyperledger Fabric components such as the endorser, committer, ledger (store) , orderer (consensus service) and the chain-code.

The following diagram provides a high-level component diagram of the SmartQuora DApp.


Web Application
Application for rendering dashboard and user interaction
Rest Server
Expose SmartQuora DApp transactions and queries as services
Passport
Authentication middleware for Web Application
SmartQuora DApp
Smart Contracts make-up of the De-centralized Application
Hyperledger Fabric
Blockchain Platform
Docker
Container platform for running various modules of Hyperledger
AWS EC2
Linux Virtual Machines for hosting the Blockchain nodes

As such, the blockchain is a decentralized system consisting of many nodes that communicate with each other on a peer-to-peer network. The blockchain runs programs called chaincode, which holds state and ledger data, and executes transactions. The chaincode is the central element as transactions are operations invoked on the chaincode. Transactions have to be “endorsed” and only endorsed transactions may be committed and have an effect on the state.

Storage

DApps such as SmartQuora consists of Smart Contracts that are translated to chaincode. This chaincode is then deployed into the blockchain. Storage of chaincode and state information differs across blockchain platforms. The default mechanism of persistent state storage in Hyperledger Fabric is LevelDB or CouchDB. In addition to supporting chaincode operations to store and retrieve assets, CouchDB allows performing complex rich queries against the data stored in the blockchain. SmartQuora uses CouchDB.

Service Layer

Aside from storage, a DApp requires a service layer to communicate with the chaincode on the blockchain and a front-end for user interaction. SmartQuora takes advantage of the Hyperledger Composer REST server which uses Loopback to generate a REST server and maps it to the transactions and queries on the DApp.

Authentication

SmartQuora DApp uses the OAuth authentication strategy of the Passport authentication middleware to secure the REST server. Specifically, it uses the Passport Google OAuth delegated authentication strategy allowing users to authenticate themselves using their Google account.

All information regarding authenticated users and their wallets is persisted in a LoopBack data source by using a LoopBack connector. By default, the REST server uses the LoopBack “memory” connector to persist user information, which is lost when the REST server is terminated. To enable persistent storage of the authenticated users and their wallets, SmartQuora uses a MongoDB LoopBack connector that stores data in a highly available MongoDB data source online at mLab. It should be noted that a local MongoDB cluster can be used instead of MLab as well. MLab was chosen to ease the implementation process.

Digital Identity & Wallets

Blockchain uses Digital Identities to represent participants in the network. A identity is a digital certificate and private key. These identities are used to sign transactions on behalf of the participants on the blockchain network. Identities are assembled in an envelope called *business network cards* along with the metadata and connection profile of the participant in Hyperledger Fabric. These business cards are then stored in wallets. As such, a participant can have multiple business cards in their wallet.

Nodes

The decentralized nature of a blockchain platform is what gives it the power to tolerate system failures, record transactions that cannot be altered retroactively without the alteration of subsequent blocks and the collusion of the network. Decentralization or peer-to-peer network requires multiple nodes to be added to the blockchain network. These nodes are the communication entities of the blockchain. There are three types of nodes in the Hyperledger Fabric blockchain:

1. **Client** : The client represents the entity that acts on behalf of an end-user. It orchestrates data and transactions between peers.
2. **Peer**: A peer receives ordered state updates in the form of *blocks* from the ordering service and maintain the state and the ledger. Peers can additionally take up a special role of an **endorser** which uses endorsement policies to sign the transactions. Once the client receives enough signatures to satisfy the endorsement policy, it can submit the transaction (and the signatures) to be added to the

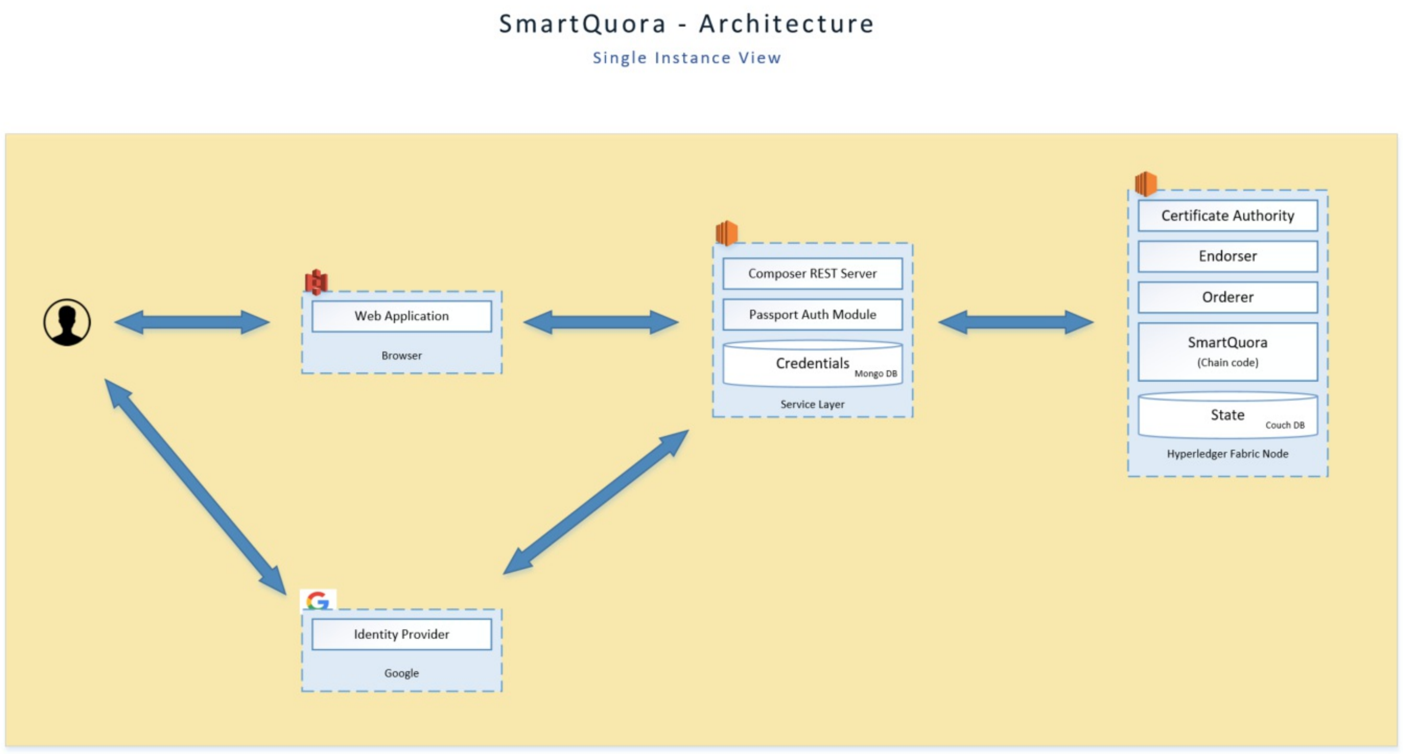
ledger.

3. **Orderer**: The orderer provides a shared *communication channel* to clients and peers, offering a broadcast service for messages containing transactions and implements a delivery guarantee.

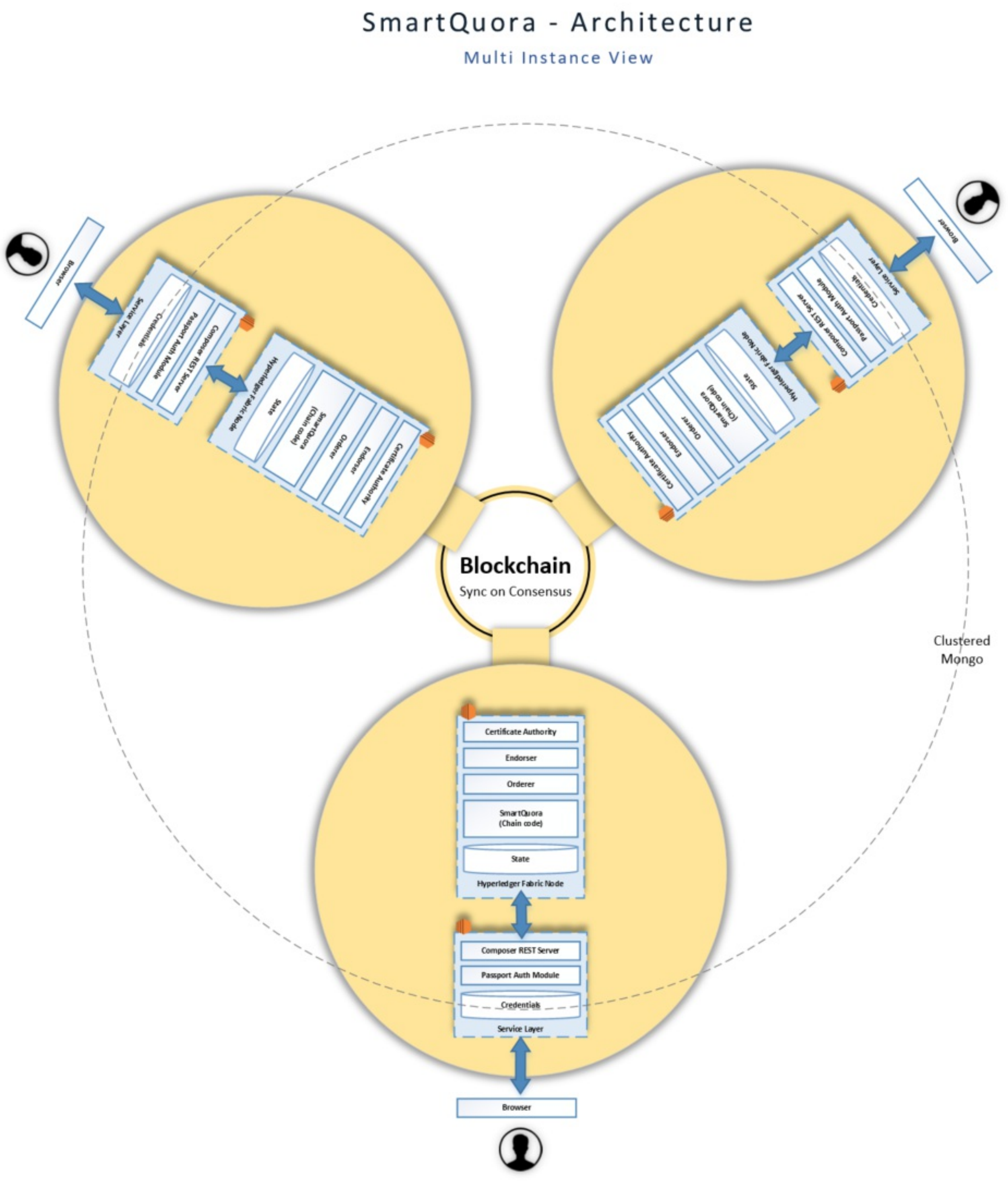
For further details on the basic workflow of a transaction inside a Hyperledger Fabric blockchain please refer to this document.

Single and Multi Deployment View of the SmartQuora DApp

The following diagram depicts a single-node view of the SmartQuora DApp.



The following diagram depicts the SmartQuora DApp when deployed on a multiple hosts to derive the benefits of the blockchain. For the sake of simplicity, in this tutorial, we will deploy the SmartQuora DApp as a logical instance on single host.



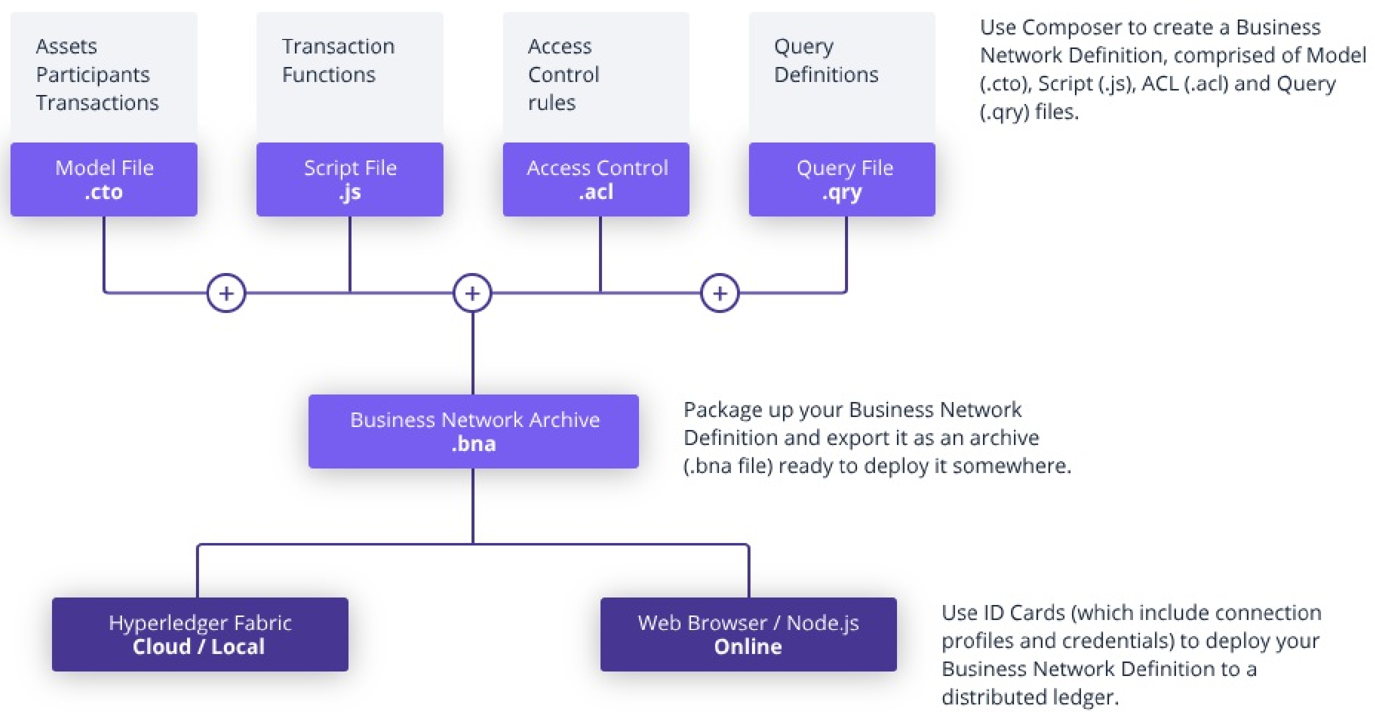
DApps - Decentralized Applications

Understanding Blockchain

Before we break-down a DApp it is essential to understand its underlying technology - the Blockchain. Blockchain is a continously growing digital ledger of records organized in *blocks* that are linked together by cryptographic validation. The key is to understand that this ledger is neither stored in a centralized location nor managed by any single entity, hence its *decentralized nature*. The block validation system results in new transactions being added irreversibly and old transactions preserved forever for all authorized participants to see, hence its transparency and resilience. Applications built on top of blockchain technology is called DApps. For an overview of blockchain and its underlying technologies, please review the following whitepaper.

Anatomy of the DApp (or BNA)

In the world of Hyperledger Fabric, DApps are called BNA or Business Network Applications. We will use the Hyperledger Composer to model the SmartQuora BNA - Business Network Application. Composer is an extensive, open development toolset and framework to make developing blockchain applications easier on the Hyperledger Fabric platform. Hence, I will use the term DApp or BNA interchangeably throughout the rest of the documentation.


Assets
Participants
Transactions
Model File
.cto
Transaction
Functions
Script File
.js
Access
Control
rules
Access Control
.acl
Query
Definitions
Query File
.qry
Use Composer to create a Business Network Definition, comprised of Model (.cto), Script (.js), ACL (.acl) and Query (.qry) files.
Business Network Archive
.bna
Package up your Business Network Definition and export it as an archive (.bna file) ready to deploy it somewhere.
Hyperledger Fabric
Cloud / Local
Web Browser / Node.js
Online
Use ID Cards (which include connection profiles and credentials) to deploy your Business Network Definition to a distributed ledger.

Hyperledger Composer Components

Before we understand the components of the SmartQuora app, it is essential to understand the Hyperledger Composer framework. A Hyperledger Composer consists of the following elements:

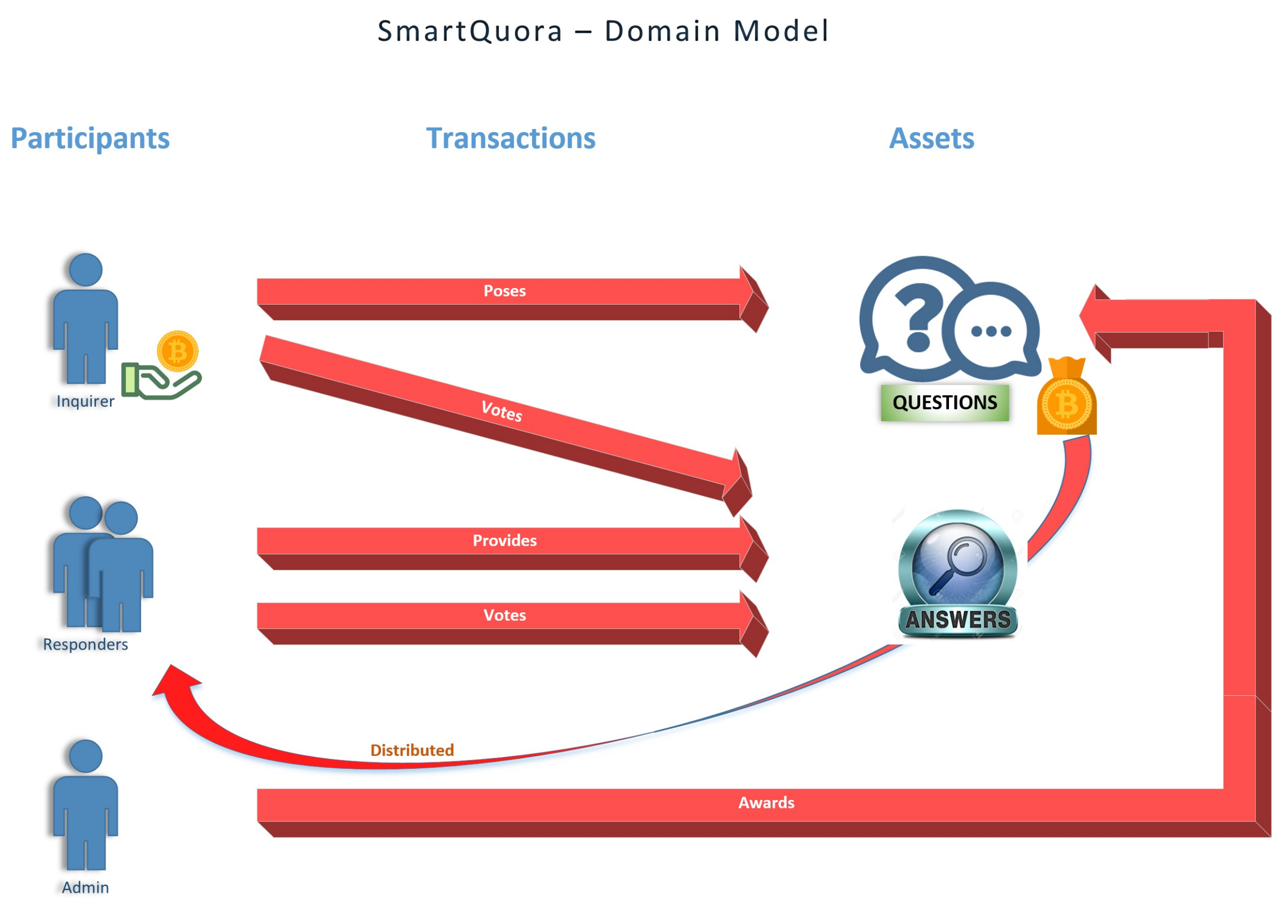
1. Participants
2. Resources comprising of Assets, Transactions, Events, Enumerated Types and Concepts.
3. Queries
4. Access Control Declarations.

- **Participiants** - Participants represents users who interact with assets.
- **Assets** - Assets represents entities which could represent place or things.

- **Transactions** - Transactions are actions that participants can carry out on assets.
- **Events** - Events are emitted by Hyperledger Composer as a result of transactions.
- **Queries** - SQL-like queries that can be used to search for assets based on its attributes.
- **Access Control** - Access Control provides declarative access control over the elements of the domain model. By defining ACL rules you can determine which users/roles are permitted to create, read, update or delete elements in a business network’s domain model.

SmartQuora Domain Model

Now that we have an understanding of the components that make up a Hyperledger BNA, let us take a look at the components of the SmartQuora BNA. The following diagram provides a high-level view of the SmartQuora Domain Model.



SmartQuora Participants

There are three types of Participants in SmartQuora BNA. They are: Inquirers, Responders and Administrators. Inquirers and Responders are represented as *QuoraUsers* in the application because their function can interchange - an inquirer can respond to questions from other inquirers or a respondent for a question can pose his/her own questions. Participants maintain tokens that is placed as a stake when asking questions.

The following rules are enforced by the system currently:

1. Inquirers cannot answer their own questions.
2. Responders cannot vote for their own answers.
3. Responders cannot vote more than once for the same answer.

4. Answers will not be accepted after the due date.
5. Only administrators can award questions and distribute the reward among the voted answers after the due date.

SmartQuora Assets

There are two types of Assets modeled in the SmartQuora application. They are: Question and Answer.

- **Question** - A question consists of an unique id, question description, owner, status (CREATED, ANSWERED, AWARDED, or DEFAULTED), reward amount and a list of answers. If a question is answered and is voted for, the stake is equally distributed among the owners of the voted answers.

- **Answer** - An answer consists of an unique id, answer description, owner, status (CREATED, VOTED, AWARDED), earnings, and a list of voters. When a question is awarded, the earnings attribute reflect the earnings that was generated by that particular answer for the respondent.

SmartQuora Transactions

The following type of transactions are available in the SmartQuora application.

- **CreateQuestion** - A *CreateQuestion* transaction is invoked by the Inquirer to pose a question. The reward amount and time by which answers are due has to accompany the request. The *CreateQuestion* transaction generates the *QuestionCreated* event.

- **CreateAnswer** - A *CreateAnswer* transaction is invoked when a respondent provides an answer to an existing question. The *CreateAnswer* transaction should be invoked with the identifier of the associated question and the answer description. The *CreateAnswer* transaction generates an *AnswerCreated* event. This transaction ensures that the question owners cannot answer their own questions.

- **VoteAnswer** - A *VoteAnswer* transaction is invoked to vote up or vote down an existing answer. As a result, it should be invoked with a reference to an existing answer and the direction of the vote (up or down). A *VoteAnswer* transaction generates the *AnswerVoted* event. This transaction ensures that respondents cannot vote for their own answers or vote multiple tims for an answer.

- **AwardQuestion** - A *AwardQuestion* transaction is invoked to find out the highest voted answers and distribute the reward proportionately amongst the voted answers. This transaction generates the *QuestionAwarded* event.

SmartQuora Events

Events in a Hyperledger Fabric BNA is pushed on a Web Socket. Any system listening to the Web Socket will get notified of all the events. As a result, the listeners have to filter out events that are not relevant to it.

SmartQuora events are as follows:

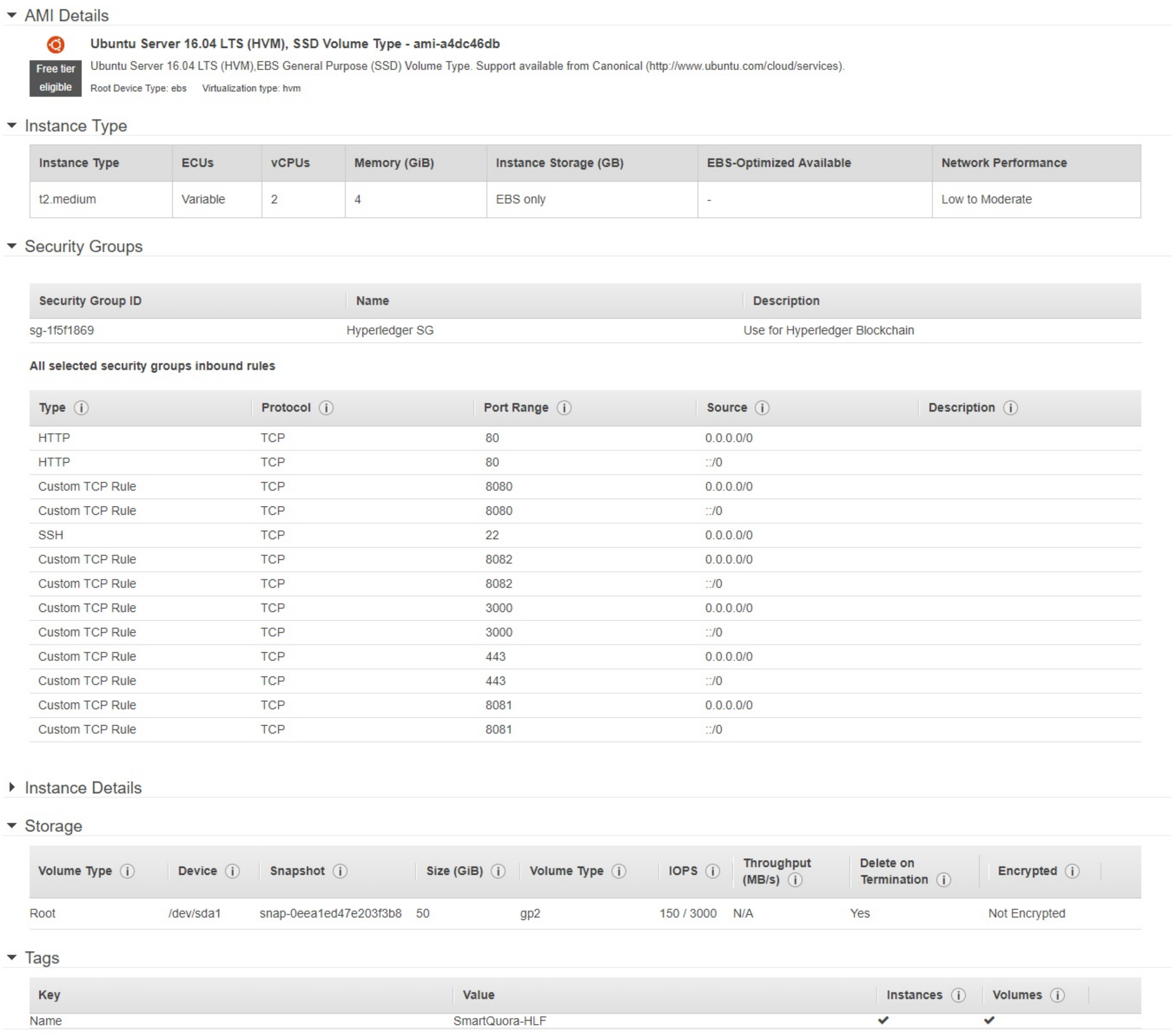
- **QuestionCreated** - Generated by the *CreateQuestion* transaction.
- **AnswerCreated** - Generated by the *CreateAnswer* transaction.
- **AnswerVoted** - Generated by the *VoteAnswer* transaction.
- **QuestionAwarded** - Generated by the *AwardQuestion* transaction.

Setup Instructions

Installing Hyperledger Fabric on AWS

Launch an EC2 Instance

For this exercise, we will deploy Hyperledger Fabric on a clean Ubuntu 16.04 LTS 64-bit environment on a AWS EC2 instance as per the configuration shown in the picture below.
Make sure to open ports: 22, 80, 443, 8080-81 & 3000.


AMI Details

Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-a4dc46db

Free tier eligible

Ubuntu Server 16.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Root Device Type: ebs   Virtualization type: hvm

Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.medium	Variable	2	4	EBS only	-	Low to Moderate

Security Groups

Security Group ID	Name	Description
sg-1f5f1869	Hyperledger SG	Use for Hyperledger Blockchain

All selected security groups inbound rules

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	
HTTP	TCP	80	::/0	
Custom TCP Rule	TCP	8080	0.0.0.0/0	
Custom TCP Rule	TCP	8080	::/0	
SSH	TCP	22	0.0.0.0/0	
Custom TCP Rule	TCP	8082	0.0.0.0/0	
Custom TCP Rule	TCP	8082	::/0	
Custom TCP Rule	TCP	3000	0.0.0.0/0	
Custom TCP Rule	TCP	3000	::/0	
Custom TCP Rule	TCP	443	0.0.0.0/0	
Custom TCP Rule	TCP	443	::/0	
Custom TCP Rule	TCP	8081	0.0.0.0/0	
Custom TCP Rule	TCP	8081	::/0	

Instance Details

Storage

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/sda1	snap-0eea1ed47e203f3b8	50	gp2	150 / 3000	N/A	Yes	Not Encrypted

Tags

Key	Value	Instances	Volumes
Name	SmartQuora-HLF	✓	✓

Associate an Elastic IP

Allocate a new Elastic IP address and associate it with the newly created instance. This is not mandatory but will help maintain sanity while working on the scripts between relaunches of the instance.

Create Hyperuser

Hyperledger Fabric installation prohibits using the root identity to install the software. Therefore, create a new user and add that user to the *sudo* group. You can choose any user name. I am using *hyperuser*

```
$ sudo adduser hyperuser

$ sudo adduser hyperuser sudo
```

Switch user to hyperuser and add the current working directory "." to the PATH environment variable in .profile.

```
su - hyperuser

vi .profile

PATH=".:$HOME/bin:$HOME/.local/bin:$PATH"
```

Note: Always use *su -* to switch user going forward to enable the environment variables are sourced from your *.profile*

Note: While this section goes over the installation of HLF on a compute instance on AWS EC2, the installation can be done on a compute instance on any public cloud such as Microsoft Azure, Google Cloud, IBM Bluemix or your own virtual machine on your local desktop/laptop.

Install the Hyperledger Fabric Pre-requisites

Use the following commands or follow the installing pre-requisites guide to prep the environment for Hyperledger Fabric installation.

```
curl -O https://hyperledger.github.io/composer/latest/prereqs
```

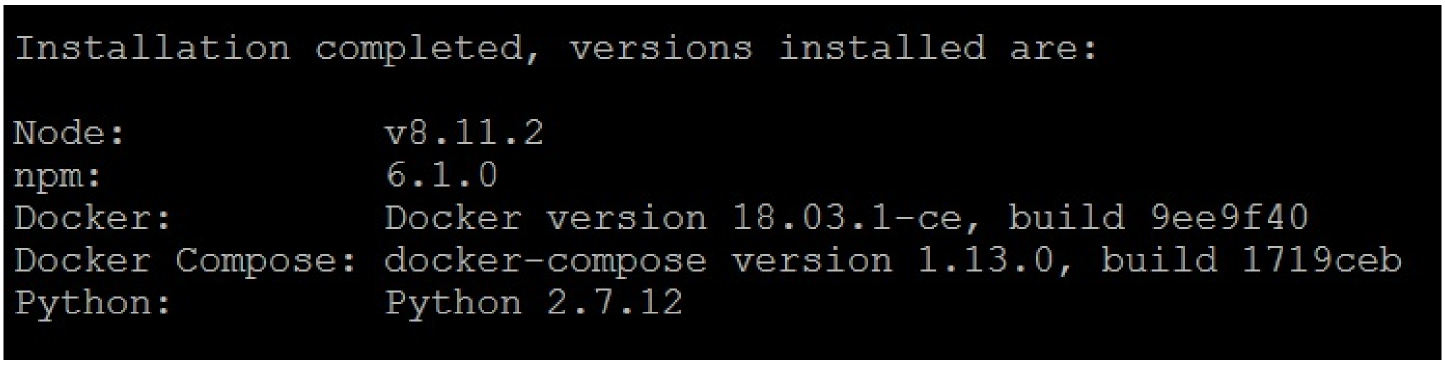
```
-ubuntu.sh

chmod u+x prereqs-ubuntu.sh
```

Next run the script to install the pre-requisites.

```
./prereqs-ubuntu.sh
```

After the installation is complete you should see a message stating that the following components are installed:

1. node
2. npm
3. docker
4. python



```
Installation completed, versions installed are:

Node:            v8.11.2
npm:             6.1.0
Docker:          Docker version 18.03.1-ce, build 9ee9f40
Docker Compose:  docker-compose version 1.13.0, build 1719ceb
Python:          Python 2.7.12
```

At this point remember to log out and log back in to ensure you have access to the newly deployed binaries.

Verify that you have access to the binaries installed as follows:

```
docker -v
npm -v
node -v
python
```

Install the Hyperledger Development Environment

Follow the instructions below or use the HLF Development Tools installation guide to install the HLF development environment.

1. Composer CLI tools:

   ```
   npm install -g composer-cli
   ```

2. Utility for running a REST Server on your machine to expose your business networks as RESTful APIs:

   ```
   npm install -g composer-rest-server
   ```

3. Useful utility for generating application assets:

   ```
   npm install -g generator-hyperledger-composer
   ```

4. Yeoman is a tool for generating applications, which utilises `generator-hyperledger-composer`:

```
npm install -g yo
```

5. Install the Playground app used for editing and testing Business Networks:

```
npm install -g composer-playground
```

6. Download the tool that installs Hyperledger Fabric:

```
mkdir ~/fabric-dev-servers && cd ~/fabric-dev-servers
curl -O https://raw.githubusercontent.com/hyperledger/c
omposer-tools/master/packages/fabric-dev-servers/fabric
-dev-servers.tar.gz
tar -xvf fabric-dev-servers.tar.gz
```

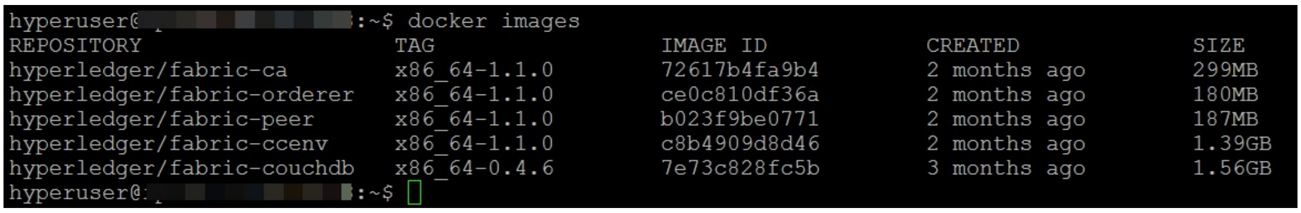
7. Install Hyperledger Fabric runtime:

```
cd ~/fabric-dev-servers
./downloadFabric.sh
```

8. Ensure five Docker images are downloaded:

```
docker images
```

You should see the following:



```
hyperuser@[redacted]:~$ docker images
REPOSITORY                   TAG                 IMAGE ID            CREATED             SIZE
hyperledger/fabric-ca        x86_64-1.1.0        72617b4fa9b4        2 months ago        299MB
hyperledger/fabric-orderer   x86_64-1.1.0        ce0c810df36a        2 months ago        180MB
hyperledger/fabric-peer      x86_64-1.1.0        b023f9be0771        2 months ago        187MB
hyperledger/fabric-ccenv     x86_64-1.1.0        c8b4909d8d46        2 months ago        1.39GB
hyperledger/fabric-couchdb   x86_64-0.4.6        7e73c828fc5b        3 months ago        1.56GB
hyperuser@[redacted]:~$
```

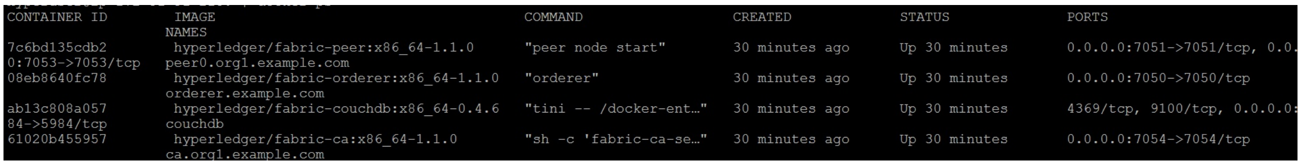
9. The first time you start up a new runtime, you'll need to run the start script, then generate a PeerAdmin card which will be used to connect to the Hyperledger Fabric environment:

```
    cd ~/fabric-dev-servers
    ./startFabric.sh
    ./createPeerAdminCard.sh
```

10. Verify that the Hyperledger Fabric environment started successfully by running the following:

```
docker ps
```

This should show four containers running which are the core of the Hyperledger Fabric environment.

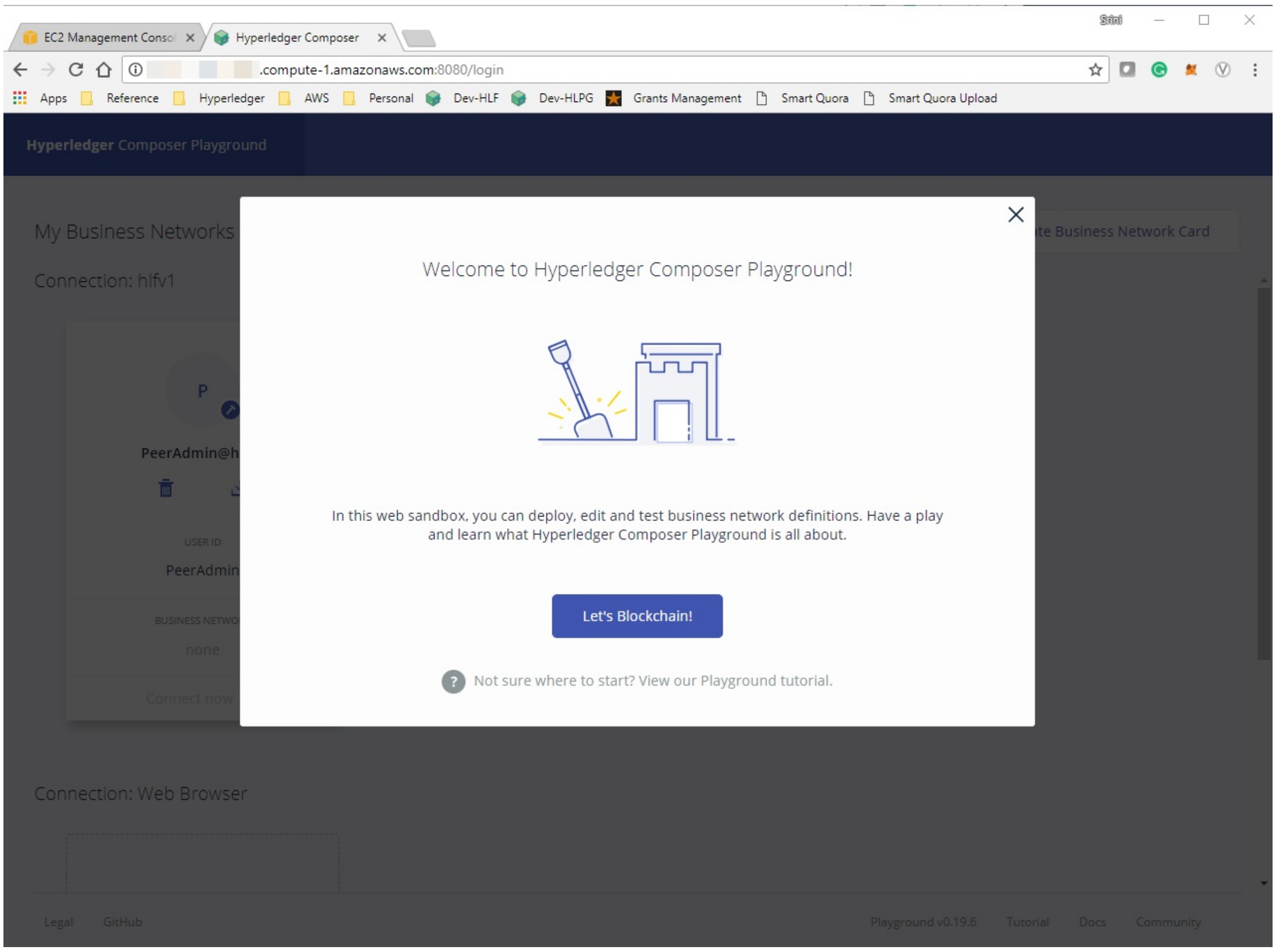


```
CONTAINER ID        IMAGE                                     COMMAND                  CREATED             STATUS              PORTS
                   NAMES
7c6bd135cdb2        hyperledger/fabric-peer:x86_64-1.1.0      "peer node start"        30 minutes ago      Up 30 minutes       0.0.0.0:7051->7051/tcp, 0.0.
0:7053->7053/tcp   peer0.org1.example.com
08eb8640fc78        hyperledger/fabric-orderer:x86_64-1.1.0   "orderer"                30 minutes ago      Up 30 minutes       0.0.0.0:7050->7050/tcp
                   orderer.example.com
ab13c808a057        hyperledger/fabric-couchdb:x86_64-0.4.6   "tini -- /docker-ent…"   30 minutes ago      Up 30 minutes       4369/tcp, 9100/tcp, 0.0.0.0:
84->5984/tcp       couchdb
61020b455957        hyperledger/fabric-ca:x86_64-1.1.0        "sh -c 'fabric-ca-se…"   30 minutes ago      Up 30 minutes       0.0.0.0:7054->7054/tcp
                   ca.org1.example.com
```

11. You can also verify that your Hyperledger Fabric is functioning normally by interacting with it on the Playground web application. To do this, start the web application on port 8080 as follows:

```
composer-playground
```

12. On your favorite browser, navigate to port 8080. If you see the following, your installation of Hyperledger Fabric runtime is successful.


EC2 Management Consol
Hyperledger Composer
.compute-1.amazonaws.com:8080/login
Apps Reference Hyperledger AWS Personal Dev-HLF Dev-HLPG Grants Management Smart Quora Smart Quora Upload
Hyperledger Composer Playground
My Business Networks
Connection: hlfv1
PeerAdmin@h
USER ID
PeerAdmin
BUSINESS NETWO
none
Connect now
Connection: Web Browser
Welcome to Hyperledger Composer Playground!
In this web sandbox, you can deploy, edit and test business network definitions. Have a play and learn what Hyperledger Composer Playground is all about.
Let's Blockchain!
Not sure where to start? View our Playground tutorial.
Legal GitHub Playground v0.19.6 Tutorial Docs Community

Starting and Stopping Hyperledger Fabric

1. To *start* the Hyperledger Fabric environment, use the following:

```
cd ~/fabric-dev-servers
./startFabric.sh
```

2. To *stop* the Hyperledger Fabric environment, use the following:

   ```
   cd ~/fabric-dev-servers
   ./stopFabric.sh
   ```

Creating a Hello World on Hyperledger Fabric

Now that we have installed Hyperledger Fabric runtime, let us create a Hello-World application to understand the principles and elements of Hyperledger Fabric.

1. In your home directory, use Yeoman to create a business application *hello_bna* using:

```
    yo hyperledger-composer
```

Use the following instructions to respond to the menu options:
Select Business Network

> ***Business network name****: hello-bna Description: My first BNA on HLF*
>
> ***Author name****: your-name*
>
> ***Author email****: your-name@email.com*
>
> ***License****: Press enter to accept Apache-2.0*
>
> ***Namespace****: Press enter to accept default*
>
> ***Do you want to generate an empty template network?****: Select No to generate a populated network*

This will create the necessary code in the *hello_bna* directory.

2. Create a subdirectory called *dist* inside the new *hello_bna* directory and step into it.

```
mkdir dist; cd dist
```

3. Create an archive by providing the type of archive as *dir* and pointing to the parent directory.

```
composer archive create -t dir -n ../
```

4. Install version 0.0.1 of the business app using the peer admin card created earlier.

```
composer network install -a hello-bna@0.0.1.bna -c PeerAdmin@
hlfv1
```

5. Create a business network admin card named *admin@hello-bna* by providing the *PeerAdmin@hlfv1* card and password *adminpw* to start version 0.0.1 of the app. This will also create the admin@hello-bna.card in the current directory.

```
composer network start  -A admin -S adminpw -c PeerAdmin@hlfv
1 -n hello-bna -V 0.0.1
```

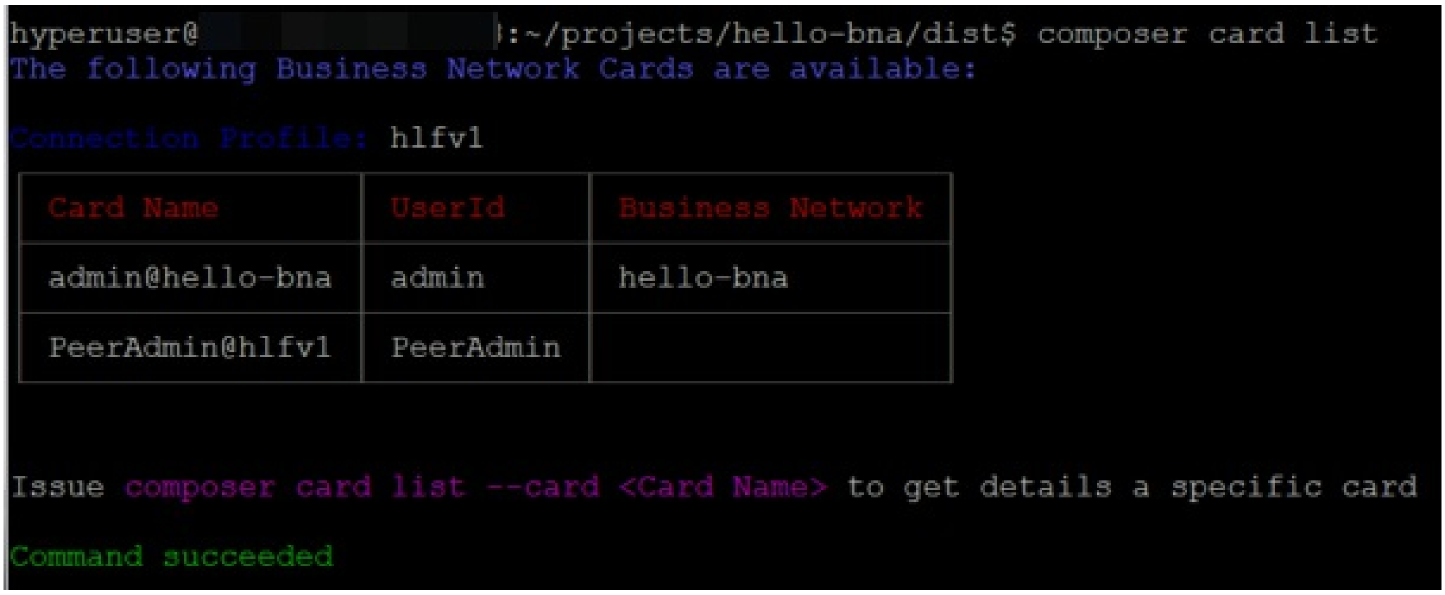
6. Import the admin@hello-bna.card using the composer card import command

```
composer card import -f ./admin@hello-bna.card
```

7. Ensure the new admin@hello-bna.card is imported using the composer-card-list command.

```
composer card list
```

This should display the existing card in your runtime as follows:



```
hyperuser@[redacted]:~/projects/hello-bna/dist$ composer card list
The following Business Network Cards are available:

Connection Profile: hlfv1
┌──────────────────┬───────────┬──────────────────┐
│ Card Name        │ UserId    │ Business Network │
├──────────────────┼───────────┼──────────────────┤
│ admin@hello-bna  │ admin     │ hello-bna        │
├──────────────────┼───────────┼──────────────────┤
│ PeerAdmin@hlfv1  │ PeerAdmin │                  │
└──────────────────┴───────────┴──────────────────┘


Issue composer card list --card <Card Name> to get details a specific card

Command succeeded
```

8. Ensure that the application is running using the composer network ping command

```
composer network ping -c admin@hello-bna
```

9. Use docker ps to ensure that the docker container running the peer to serve the hello-bna application is running

```
docker ps
```

10. Open the BNA in Playground and connect with your application. Explore the model, participant, access control layer and transaction logic.

```
composer-playground
```

11. Start the REST server & browse through the Swagger UI

```
composer-rest-server -c admin@hello-bna -n always -w true
```

Navigate to http://your-host-name:3000/explorer

Installing SmartQuora

Now that we have seen a chance to take a sample application for a spin, it is time to install the SmartQuora application.

1. Clone the SmartQuora Git on your host.

```
git clone https://github.com/skarlekar/smart-quora.git
```

2. Create a distribution directory.

```
cd smart-quora/smartquora-bna
mkdir dist
cd dist
```

3. Create an archive by providing the type of archive as *dir* and

pointing to the parent directory.

```
composer archive create -t dir -n ../
```

3. Install version 0.0.1 of the business app using the peer admin card created earlier.

```
composer network install -a smartquora-bna@0.0.1.bna -c PeerA
dmin@hlfv1
```

4. Create a business network admin card named *admin@smartquora-bna* by providing the *PeerAdmin@hlfv1* card and password *adminpw* to start version 0.0.1 of the app. This will also create the admin@smartquora-bna.card in the current directory.

```
composer network start  -A admin -S adminpw -c PeerAdmin@hlfv
1 -n smartquora-bna -V 0.0.1
```

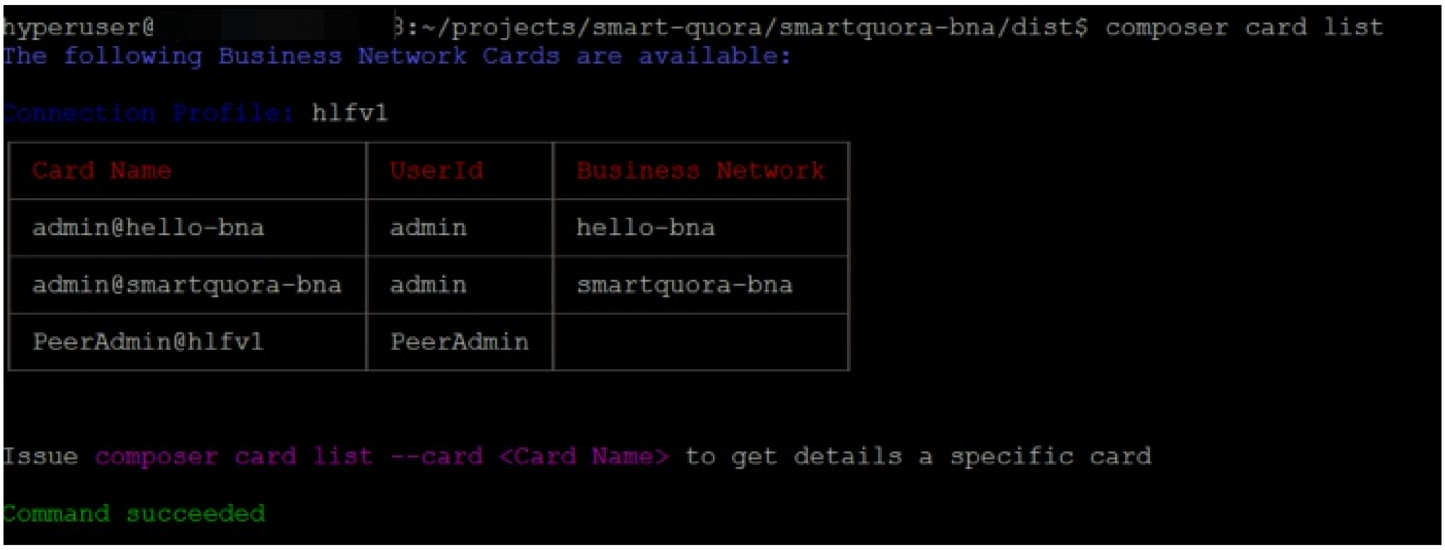
5. Import the admin@smartquora-bna.card using the composer card import command

```
composer card import -f ./admin@smartquora-bna.card
```

6. Ensure the new admin@smartquora-bna.card is imported using the composer-card-list command.

```
composer card list
```

This should display the existing card in your runtime as follows:



```
hyperuser@[redacted]:~/projects/smart-quora/smartquora-bna/dist$ composer card list
The following Business Network Cards are available:

Connection Profile: hlfv1
┌───────────────────────────┬───────────┬──────────────────┐
│ Card Name                 │ UserId    │ Business Network │
├───────────────────────────┼───────────┼──────────────────┤
│ admin@hello-bna           │ admin     │ hello-bna        │
├───────────────────────────┼───────────┼──────────────────┤
│ admin@smartquora-bna      │ admin     │ smartquora-bna   │
├───────────────────────────┼───────────┼──────────────────┤
│ PeerAdmin@hlfv1           │ PeerAdmin │                  │
└───────────────────────────┴───────────┴──────────────────┘


Issue composer card list --card <Card Name> to get details a specific card

Command succeeded
```

7. Ensure that the application is running using the composer network ping command

```
composer network ping -c admin@smartquora-bna
```

8. Use docker ps to ensure that the docker container running the peer to serve the hello-bna application is running

```
docker ps
```

9. Open the BNA in Playground and connect with your application. Explore the model, participant, access control layer and transaction logic.

```
composer-playground
```

10. Start the REST server & browse through the Swagger UI. Note that we have started the REST server using Secure Socket

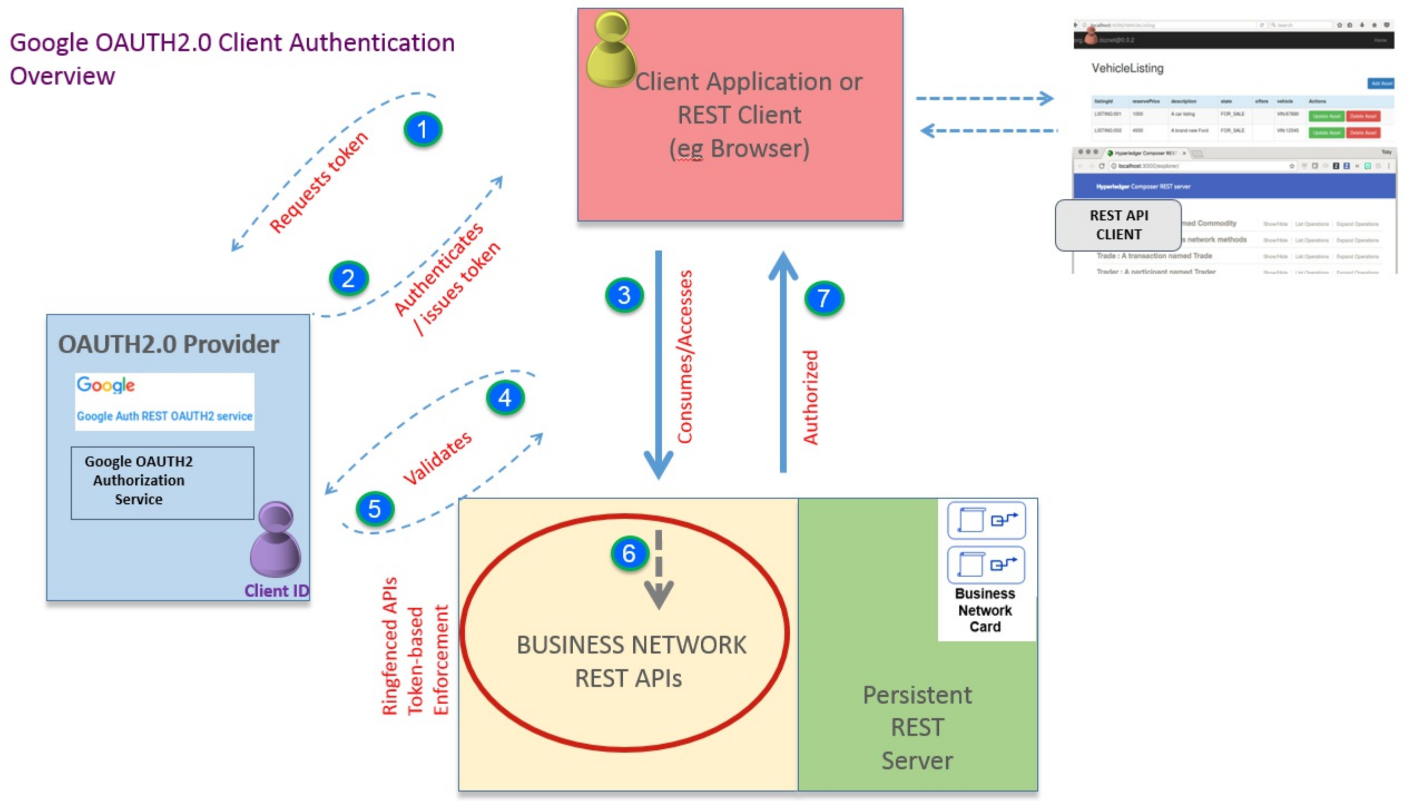
Layer this time using the *start-resh.sh* script. Hence, change the protocol from HTTP to HTTPS when browsing the REST Explorer.

```
cd ..
./start-rest.sh
```

Navigate to https://your-host-name:3000/explorer

Securing the REST Server

Now that we have a RESTful interface to our SmartQuora application, it is time to secure it using the Passport Google OAUTH2.0 delegated authentication strategy.


Google OAUTH2.0 Client Authentication Overview
1
Requests token
2
Authenticates / issues token
3
Consumes/Accesses
4
Validates
5
6
7
Authorized
Client Application or REST Client (eg Browser)
VehicleListing
REST API CLIENT
OAUTH2.0 Provider
Google
Google Auth REST OAUTH2 service
Google OAUTH2 Authorization Service
Client ID
Ringfenced APIs Token-based Enforcement
BUSINESS NETWORK REST APIs
Business Network Card
Persistent REST Server

While there are many Passport authentication strategy to choose from such as JWT, SAML, LDAP, AD etc, we will use Google+ API as the authentication provider for this exercise. The above diagram provides an overview of the authentication strategy. Here, the Composer REST server's role is to provide access to business network resources, which are protected by the Google+ API OAuth2.0 scheme. The resource owner is the Google+ API user account we set up. Its role is to grant consent (or otherwise) to the client application. The Google+ authorization server requests consent of the resource owner and issues access tokens to REST clients to enable them to access the protected resources. An access key is granted following consent in form of a token. This token allows a client to access the APIs protected by OAuth2.0.

In OAuth 2.0, these access tokens are called "bearer tokens", and can be used alone, with no signature or cryptography, to access the information. Furthermore, the access token is stored in a cookie in the local storage of the user's web browser. When the user makes a subsequent request, the access token is retrieved from the cookie, and the access token is validated, instead of reauthenticating the user.

Google+ Authentication Configuration & Setup

1. Setup Google+ authentication using the guide here.

2. Now copy the *client id* and *client secret* from Google+ and enter this in the *COMPOSER_PROVIDERS* section of the *start-smartquora.sh* script.

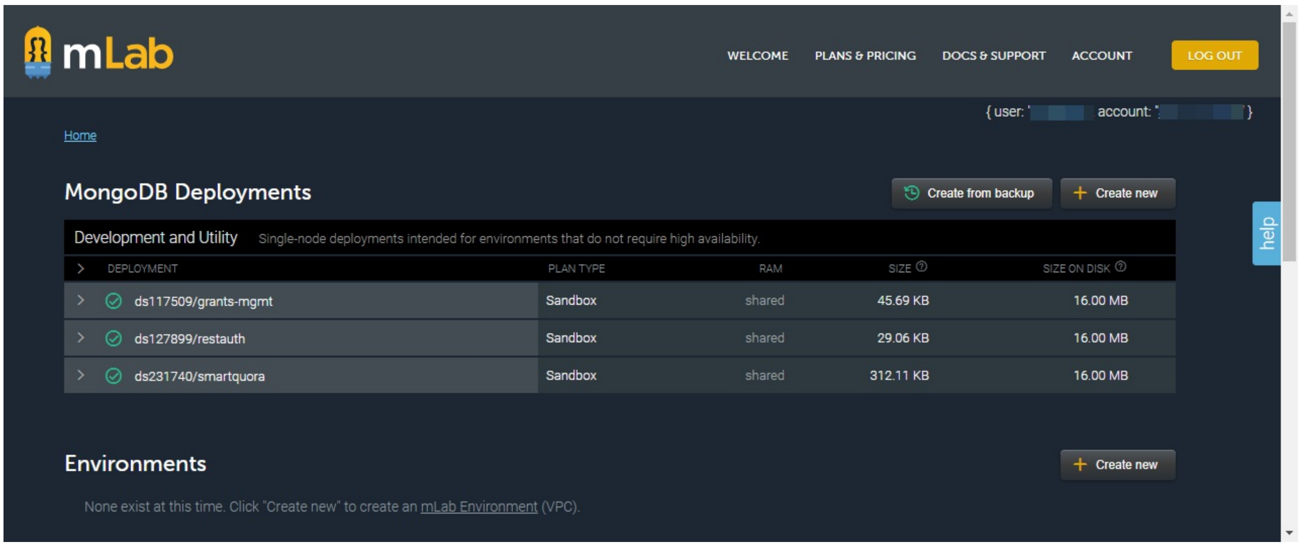
3. Now install the Passport Google OAuth2.0 strategy.

```
npm install -g passport-google-oauth2
```

Setting up the Credentials & Wallet Data Store using MongoDB

The REST Server itself is configured to persist the business network cards (required to connect to the network) using the MongoDB store. For this exercise we will use the MongoDB database hosted on http://mlab.com to keep it simple.

1. Point your browser to http://mlab.com and log into your account (create an account if you don't already have one).



{ user: '[redacted] account: '[redacted]' }

MongoDB Deployments

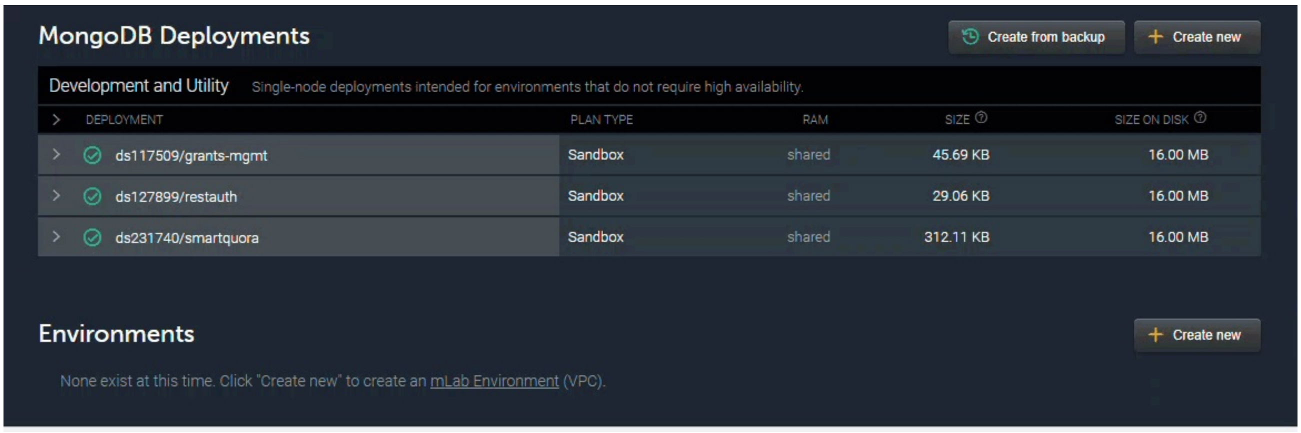
Development and Utility Single-node deployments intended for environments that do not require high availability.

DEPLOYMENT	PLAN TYPE	RAM	SIZE	SIZE ON DISK
ds117509/grants-mgmt	Sandbox	shared	45.69 KB	16.00 MB
ds127899/restauth	Sandbox	shared	29.06 KB	16.00 MB
ds231740/smartquora	Sandbox	shared	312.11 KB	16.00 MB

Environments

None exist at this time. Click "Create new" to create an mLab Environment (VPC).

2. Create a new database, select a cloud provider, select the free sandbox plan, click continue and select an available region. Click continue and provide a database name as shown below. For this exercise we will use a db called *quora-auth*.

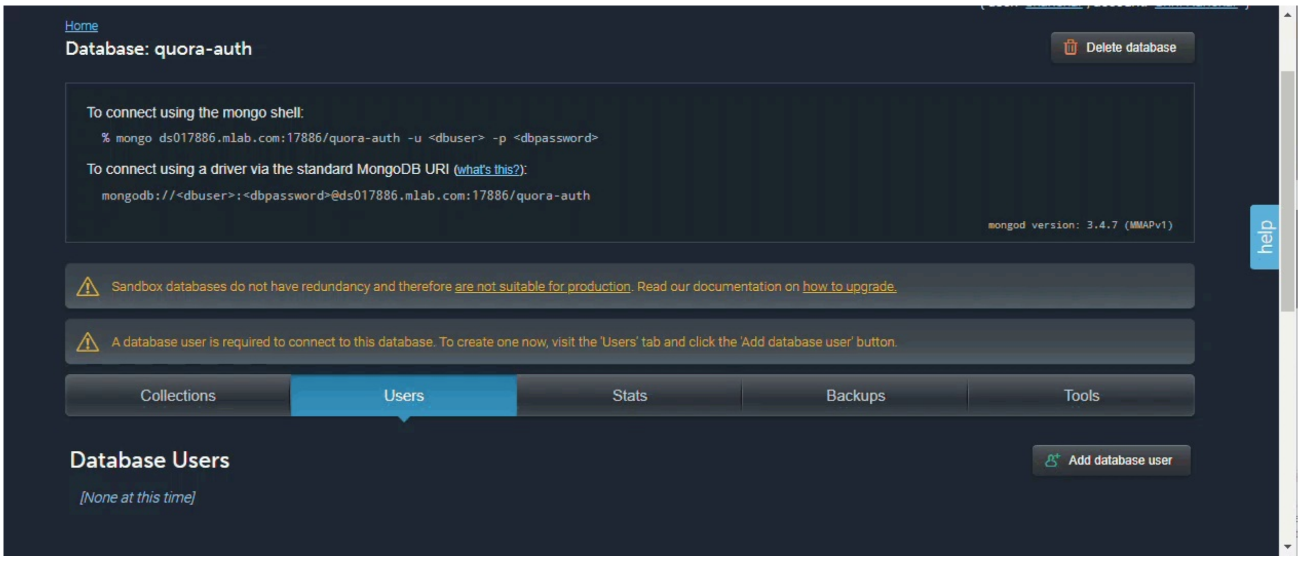


Development and Utility Single-node deployments intended for environments that do not require high availability.

DEPLOYMENT	PLAN TYPE	RAM	SIZE	SIZE ON DISK
ds117509/grants-mgmt	Sandbox	shared	45.69 KB	16.00 MB
ds127899/restauth	Sandbox	shared	29.06 KB	16.00 MB
ds231740/smartquora	Sandbox	shared	312.11 KB	16.00 MB

None exist at this time. Click "Create new" to create an mLab Environment (VPC).

3. Create an user in the database. This user's credentials will be used by the REST server to connect to the database and use it for storing the credentials. For this exercise we will create a user called *test* with password *test123*.



Database: quora-auth

To connect using the mongo shell:

```
% mongo ds017886.mlab.com:17886/quora-auth -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@ds017886.mlab.com:17886/quora-auth
```

mongod version: 3.4.7 (MMAPv1)

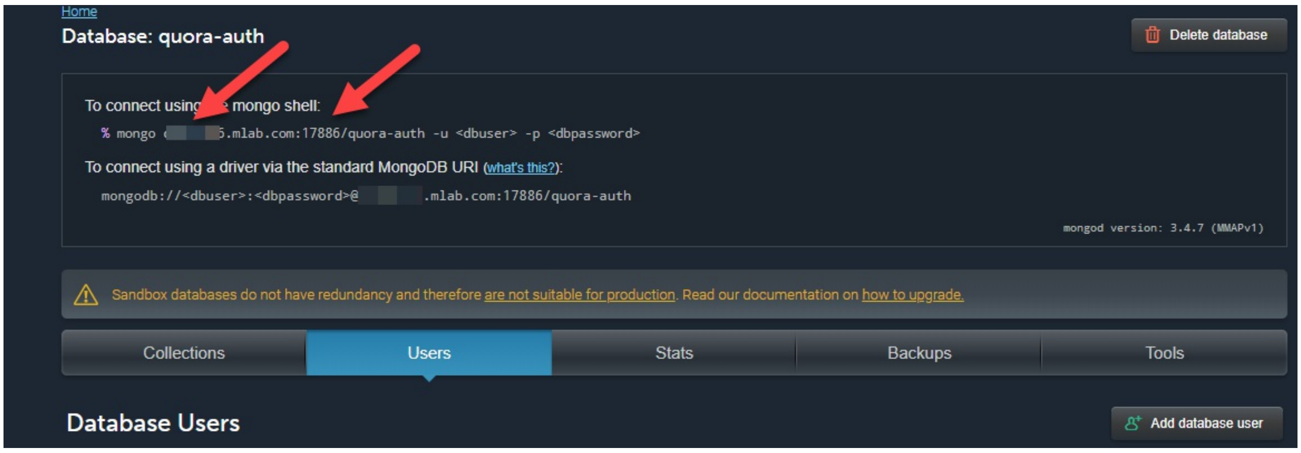
Sandbox databases do not have redundancy and therefore are not suitable for production. Read our documentation on how to upgrade.

A database user is required to connect to this database. To create one now, visit the 'Users' tab and click the 'Add database user' button.

Database Users

[None at this time]

4. Now copy the database name and port from MLab and enter this in the *COMPOSER_DATASOURCES* section of the *start-smartquora.sh* script.



To connect using the mongo shell:

```
% mongo [redacted]5.mlab.com:17886/quora-auth -u <dbuser> -p <dbpassword>
```

To connect using a driver via the standard MongoDB URI (what's this?):

```
mongodb://<dbuser>:<dbpassword>@[redacted].mlab.com:17886/quora-auth
```

```
mongod version: 3.4.7 (MMAPv1)
```

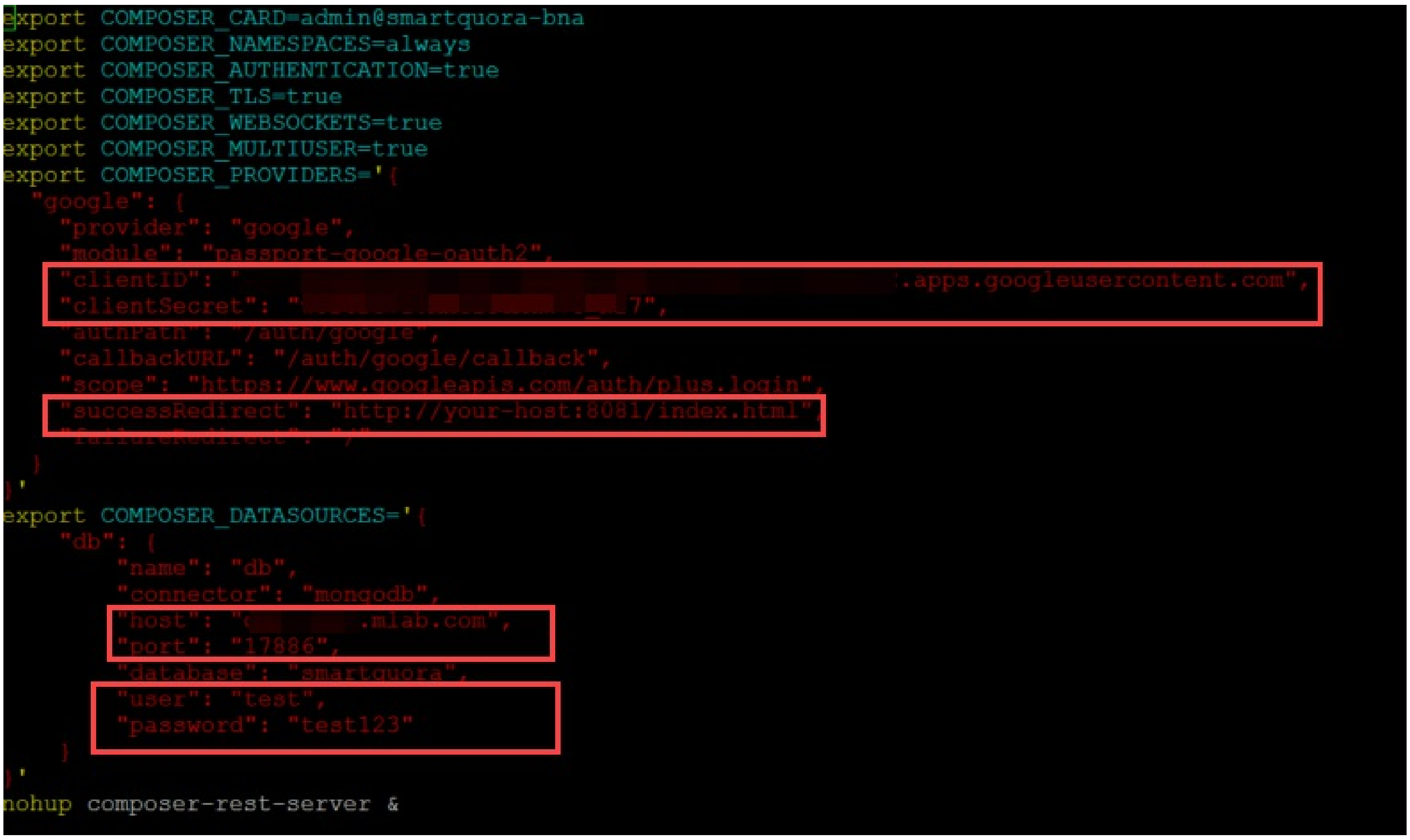
Sandbox databases do not have redundancy and therefore are not suitable for production. Read our documentation on how to upgrade.

5. Install the MongoDB Loopback connector plugin.

```
npm install -g loopback-connector-mongodb
```

Start the REST server

After the changes from the above two sections, your *start-smartquora.sh* should look like this:



```
export COMPOSER_CARD=admin@smartquora-bna
export COMPOSER_NAMESPACES=always
export COMPOSER_AUTHENTICATION=true
export COMPOSER_TLS=true
export COMPOSER_WEBSOCKETS=true
export COMPOSER_MULTIUSER=true
export COMPOSER_PROVIDERS='{
  "google": {
    "provider": "google",
    "module": "passport-google-oauth2",
    "clientID": "[redacted].apps.googleusercontent.com",
    "clientSecret": "[redacted]7",
    "authPath": "/auth/google",
    "callbackURL": "/auth/google/callback",
    "scope": "https://www.googleapis.com/auth/plus.login",
    "successRedirect": "http://your-host:8081/index.html",
    "failureRedirect": "/"
  }
}'
export COMPOSER_DATASOURCES='{
    "db": {
        "name": "db",
        "connector": "mongodb",
        "host": "[redacted].mlab.com",
        "port": "17886",
        "database": "smartquora",
        "user": "test",
        "password": "test123"
    }
}'
nohup composer-rest-server &
```

Before you restart the REST server, shut it down if it is already running.

```
lsof -i :3000
```

This will give the process-id of the server attached to port 3000 (if it is running).

Kill the process:

```
kill <pid>
```

Start the REST server by running the *start-smartquora.sh* script.

```
cd <repo>/smartquora-bna
./start-smartquora.sh
```

Installing the Web Server

1. For testing the BNA we will use a light-weight Node.js based web-server. Install and start it as follows:

```
npm install -g http-server
```

2. Before we start the webserver, replace the ‘your-host-name’ string in the *html* files with the host name of the machine where you have installed this software. Note: Only provide the domain name. Do not

add the protocol or port here.

```
cd <repo>/scripts
replace-host-name.sh ec2-sample-01.amazonaws.com
```

3. Start the web server.

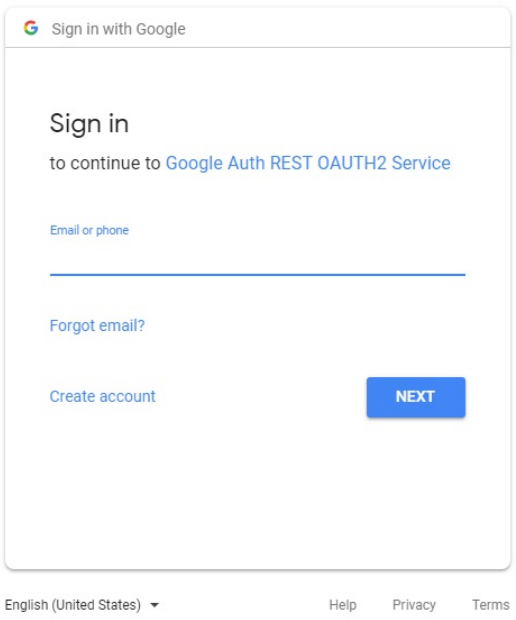
```
cd <repo>/www
http-server -p 8081
```

Running SmartQuora

Signing into SmartQuora

Now, open your browser in **incognito** mode and point your browser to: http://your-host-name:8081/index.html

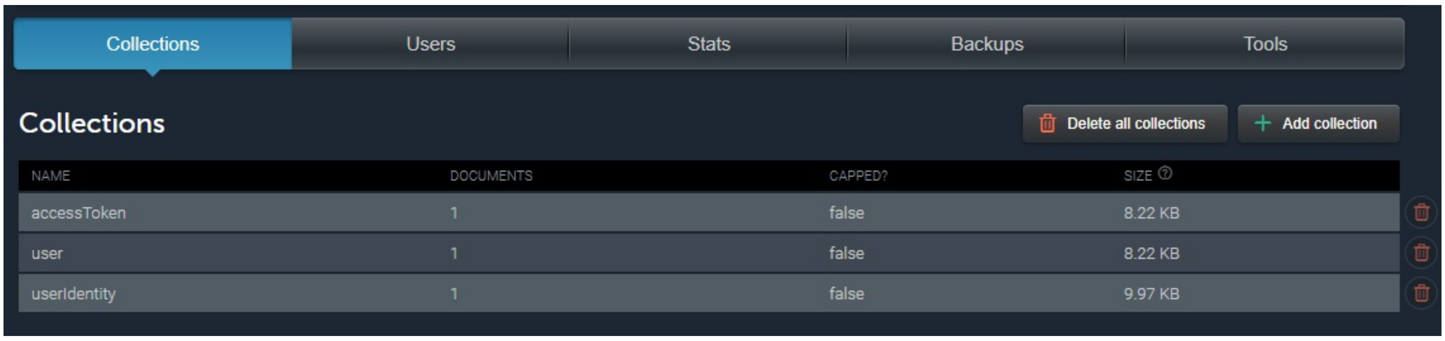
If everything went well, you browser should redirect to Google authentication page.



After authentication, Google Auth should redirect you back to the application.

If you are doing this for the first time you will land in a page asking you to upload your digital identity to your wallet.

Note that your wallet is maintained on the MongoDB database on your behalf by the REST server. If you check your MongoDB database now, you will see three collections viz., accessToken, user & userIdentity. But the collection for storing your Digital Identity called Card is not there yet as shown below



NAME	DOCUMENTS	CAPPED?	SIZE
accessToken	1	false	8.22 KB
user	1	false	8.22 KB
userIdentity	1	false	9.97 KB

Creating Digital Identity Cards

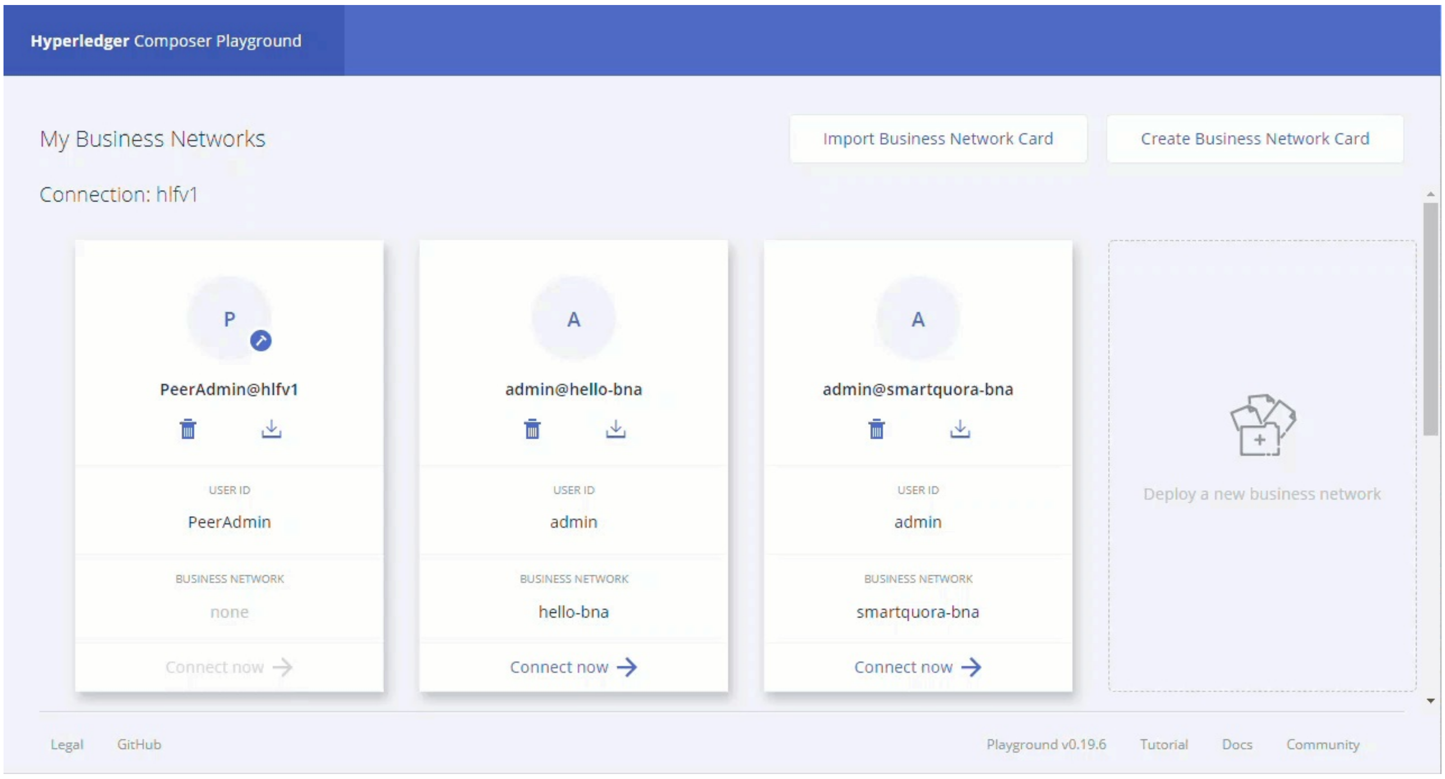
We will use the Composer Playground to create a few digital identity cards as demonstrated in the animation below.

1. Start the Playground if it is not already up.

```
composer-playground
```

2. Point your browser to http://your-host-name:8080
3. Connect to the *smartquora-bna*

4. Create a new *Participant*.
5. Navigate to the *ID Registry*.
6. Click *Issue New ID*.
7. Provide a ID Name for the *Participant*.
8. Choose a *Participant* from the list.
9. Click *Create New* and choose 'Send it to someone else_.
10. Click *Export Business Network Card*.


Hyperledger Composer Playground
My Business Networks
Import Business Network Card
Create Business Network Card
Deploy a new business network
Connect now
Legal
GitHub
Playground v0.19.6
Tutorial
Docs
Community

Connection: hlfv1

PeerAdmin@hlfv1
USER ID
PeerAdmin
BUSINESS NETWORK
none

admin@hello-bna
USER ID
admin
BUSINESS NETWORK
hello-bna

admin@smartquora-bna
USER ID
admin
BUSINESS NETWORK
smartquora-bna

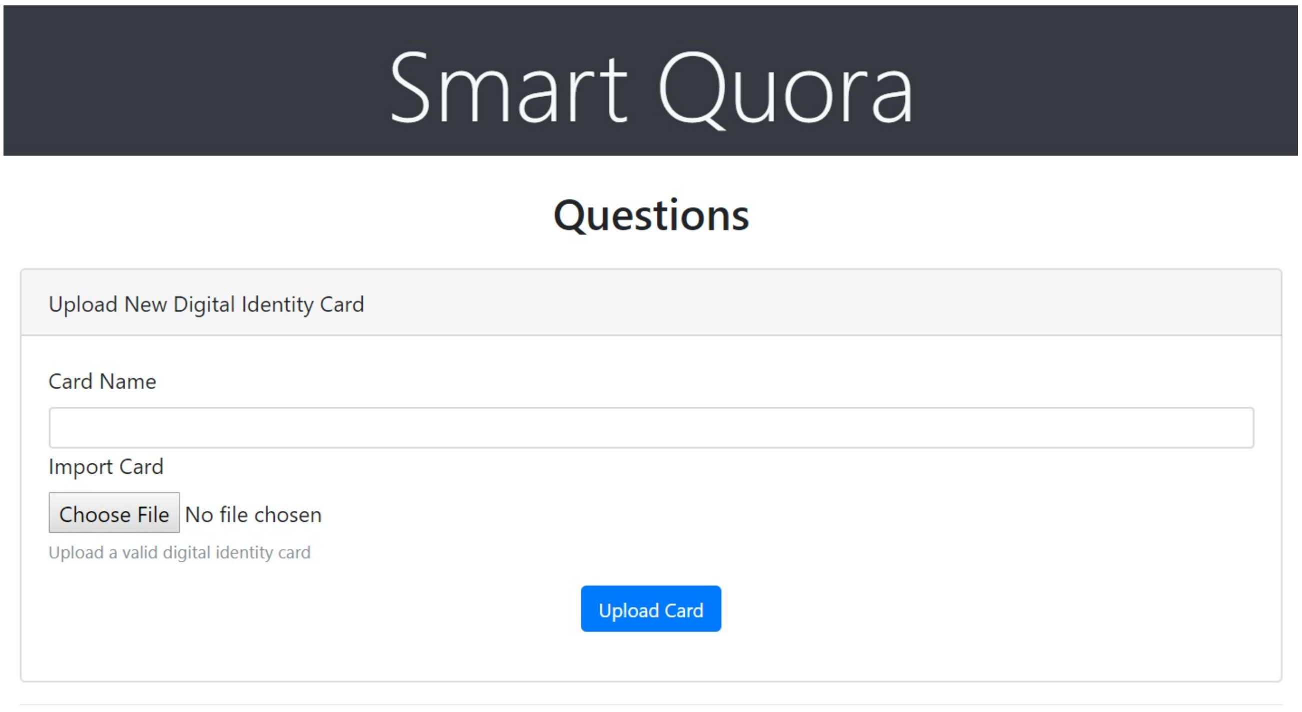
Repeat this step a few times for multiple participants. I have created five participants.

Adding the Digital Identity to your Wallet

Now that we have created a few Digital Identities, let us import them into the application.

1. Point your browser to http://your-host-name:8081/upload-card.html

2. In the form that comes up, enter a name for the card and upload a card that was exported from Composer Playground.
   Note: Due to a bug in the upload-card code, you will not receive an acknowledgement of a successful upload.
3. Repeat the step above for the remaining cards.



Smart Quora

Questions

Upload New Digital Identity Card

Card Name

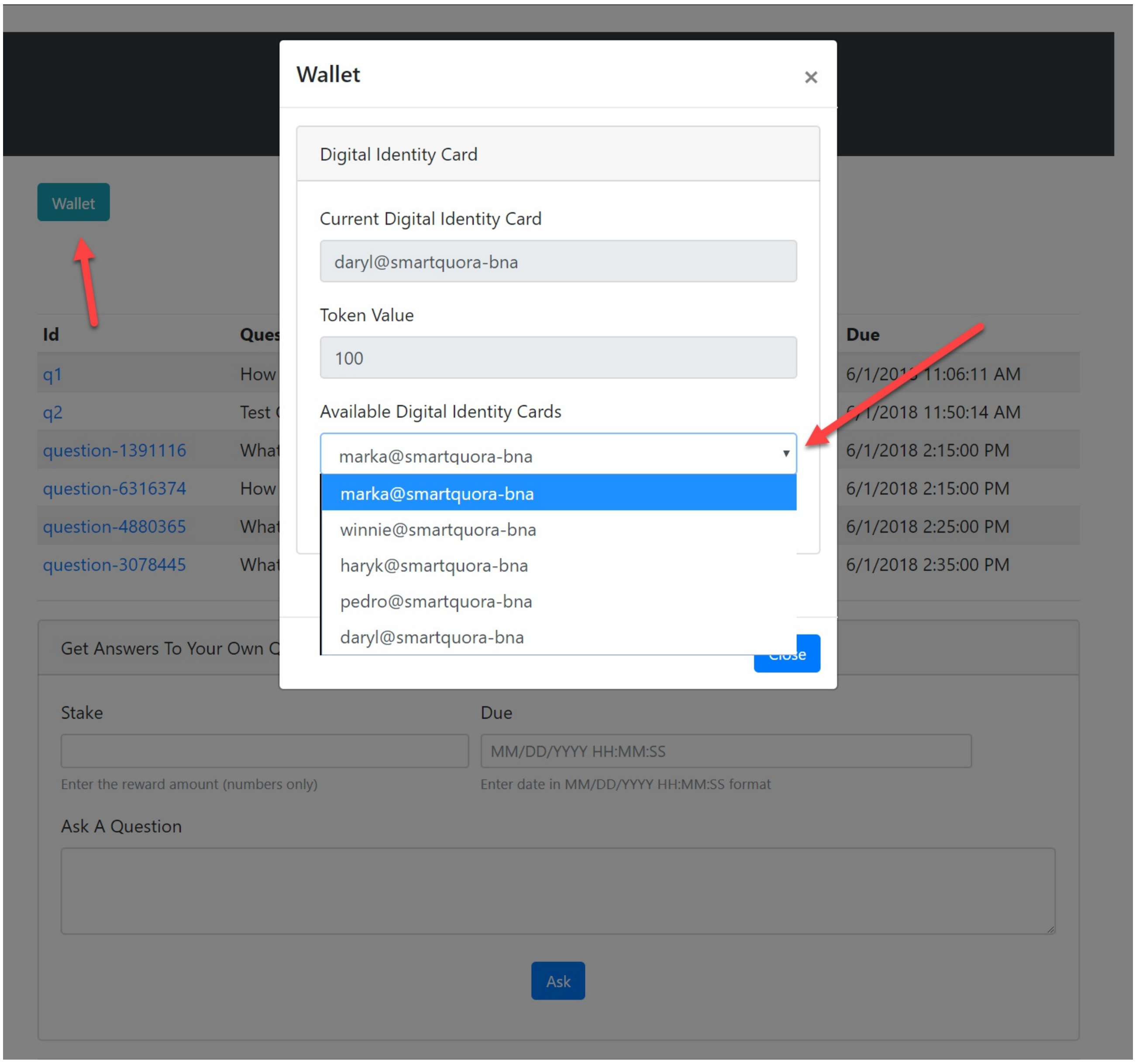
Import Card

Upload a valid digital identity card

Selecting a Digital Avatar

Now that we have a few digital identities in our wallet, let us choose one to use.

1. Point your browser to home page of Smart Quora: http://your-host-name:8081/index.html
2. Click on the *Wallet* button on the top left.
3. Choose a digital identity from the drop-down.
4. Click on *Change Card* button.
5. You should get a message stating that the digital identity was changed to the card that was selected.


Wallet ×

Digital Identity Card

Current Digital Identity Card

daryl@smartquora-bna

Token Value

100

Available Digital Identity Cards

marka@smartquora-bna

- marka@smartquora-bna
- winnie@smartquora-bna
- haryk@smartquora-bna
- pedro@smartquora-bna
- daryl@smartquora-bna

Close

Wallet

Id	Ques	Due
q1	How	6/1/2018 11:06:11 AM
q2	Test (	6/1/2018 11:50:14 AM
question-1391116	What	6/1/2018 2:15:00 PM
question-6316374	How	6/1/2018 2:15:00 PM
question-4880365	What	6/1/2018 2:25:00 PM
question-3078445	What	6/1/2018 2:35:00 PM

Get Answers To Your Own Q

Stake

Enter the reward amount (numbers only)

Due

MM/DD/YYYY HH:MM:SS

Enter date in MM/DD/YYYY HH:MM:SS format

Ask A Question

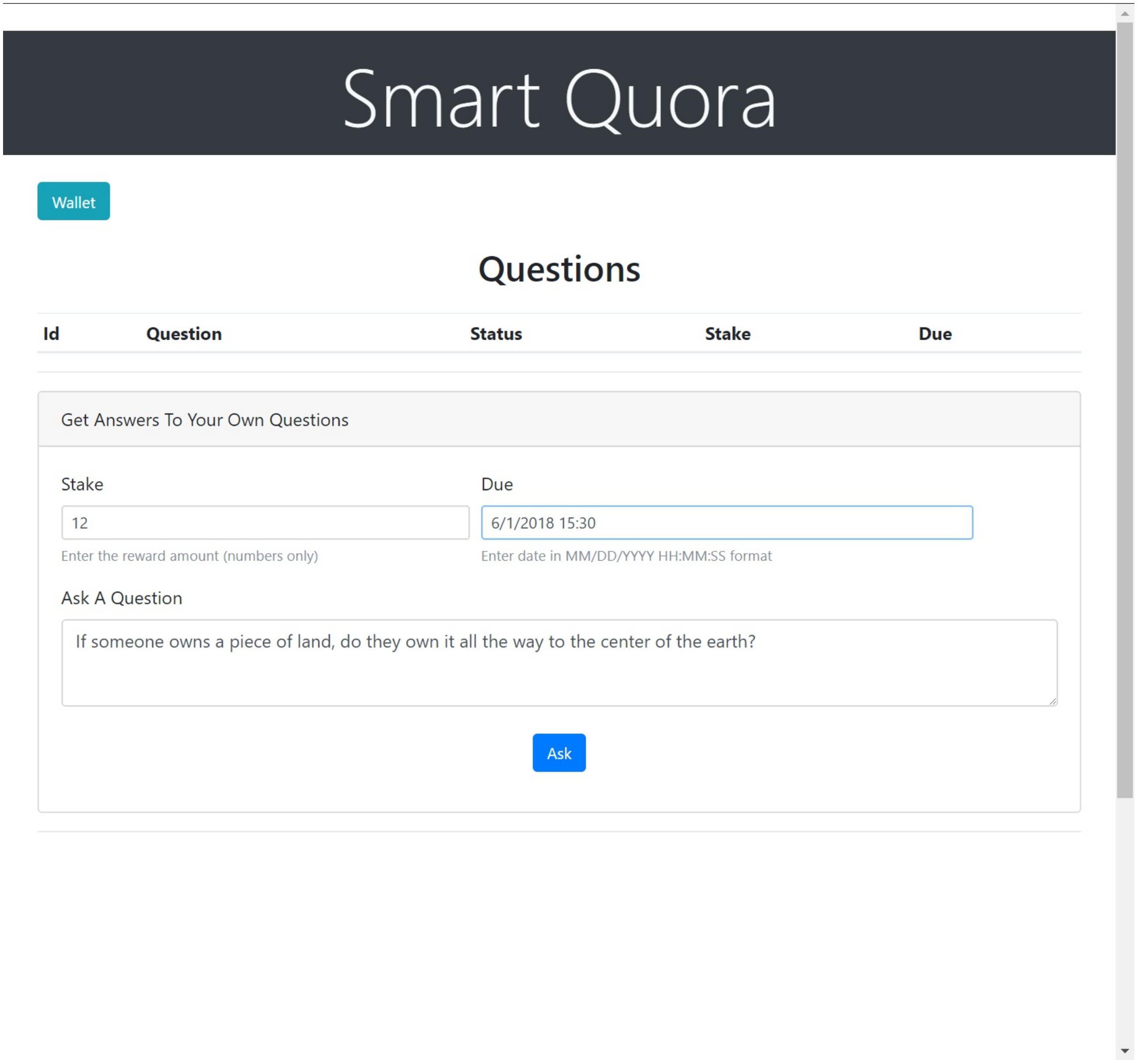
Ask

Ask a Question

To ask a question:

1. Point your browser to home page of Smart Quora: http://your-host-name:8081/index.html
2. In the form *Get Answers To Your Own Questions*, enter a reward amount, due date for the question to be answered by in 24-hour format in the *Due* field, your question in the *Ask A Question* field and click the *Ask* button.
3. If everything goes well, you should get a notification stating that

the transaction was posted successfully and your question should pop-up in the list of questions above the form as shown in the picture below.



Smart Quora

Questions

Id	Question	Status	Stake	Due

Get Answers To Your Own Questions

Stake: 12
Enter the reward amount (numbers only)

Due: 6/1/2018 15:30
Enter date in MM/DD/YYYY HH:MM:SS format

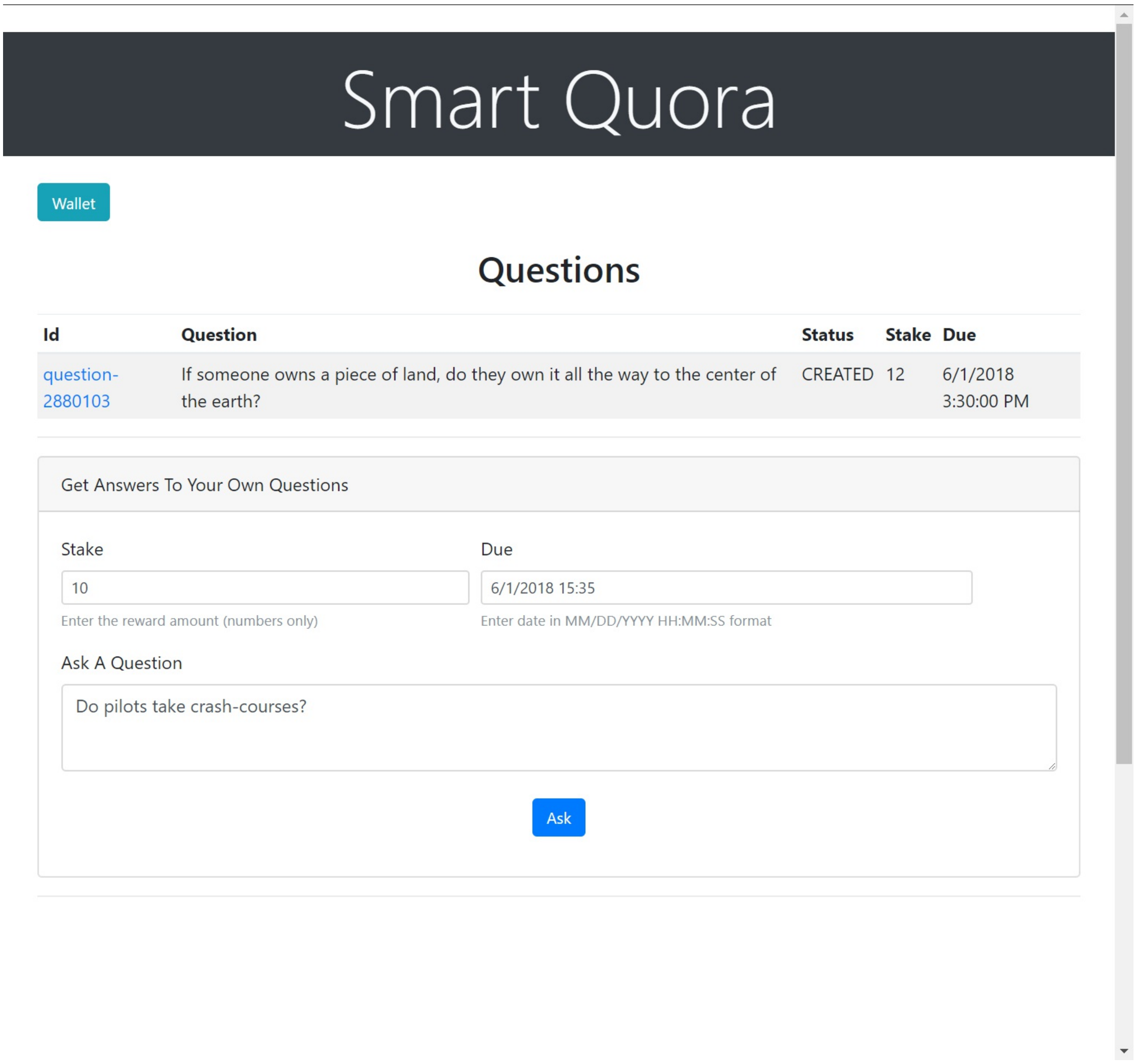
Ask A Question: If someone owns a piece of land, do they own it all the way to the center of the earth?

Provide an Answer

To answer a question:

1. Click on the question link in the home page of Smart Quora: http://your-host-name:8081/index.html
2. In the resulting page, enter your response in the *Your Answer* section and click on the *Answer* button.

3. If everything goes well, your answer should pop up in the list of answers above and you should get a notification that your transaction was posted successfully.



Questions

Id	Question	Status	Stake	Due
question-2880103	If someone owns a piece of land, do they own it all the way to the center of the earth?	CREATED	12	6/1/2018 3:30:00 PM

Get Answers To Your Own Questions

Stake: 10
Enter the reward amount (numbers only)

Due: 6/1/2018 15:35
Enter date in MM/DD/YYYY HH:MM:SS format

Ask A Question: Do pilots take crash-courses?

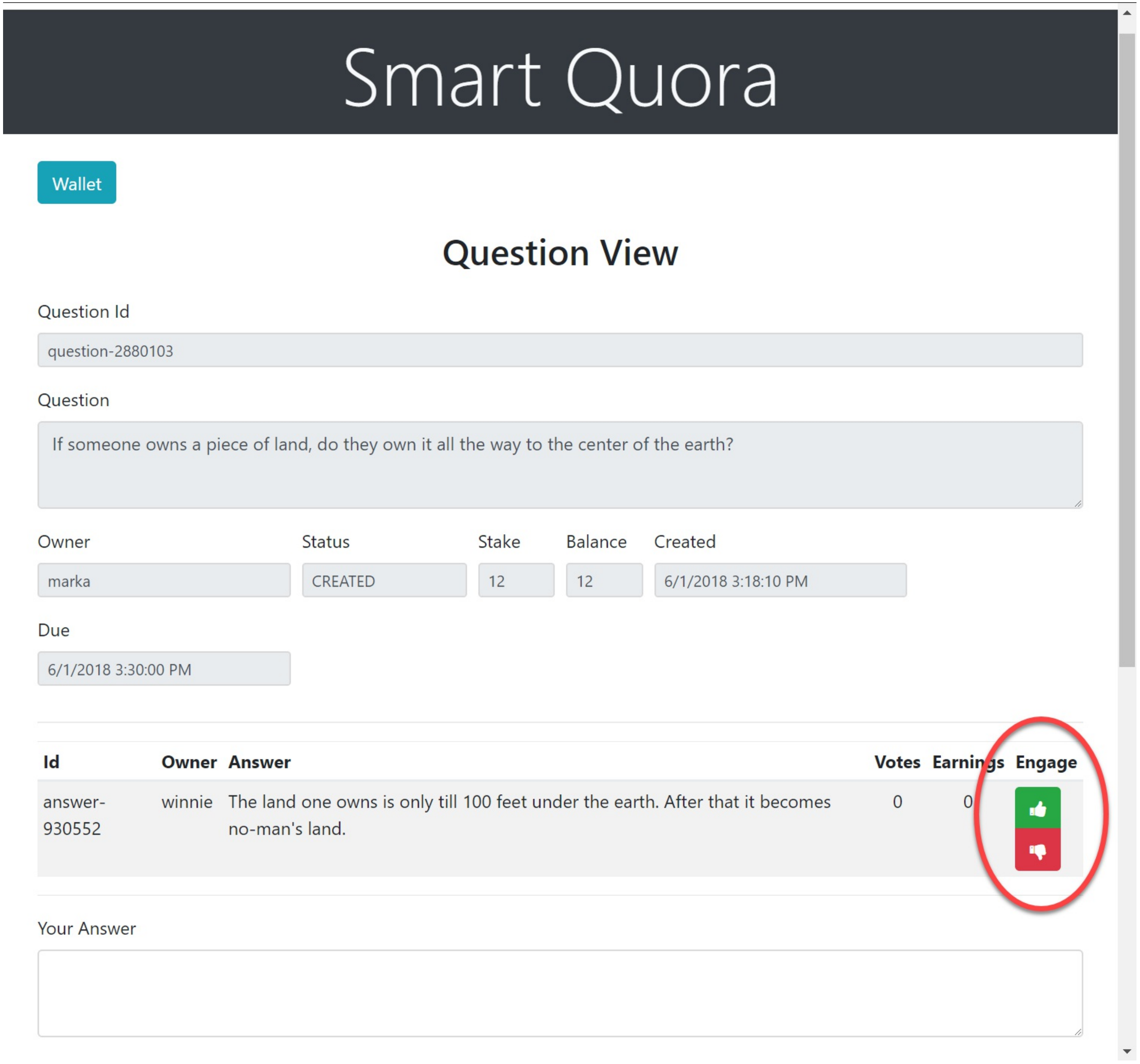
Vote for an Answer

To vote for an answer:

1. Click on the question link in the home page of Smart Quora: http://your-host-name:8081/index.html
2. In the resulting page, enter your click on the *thumbs up* or *thumbs down* button next to the answer. Please note that you cannot vote

for your own answers or vote for an answer after the due date for the question.

3. If everything goes well, your vote should be accepted and the vote count should change accordingly. Additonally, you should get a notification stating that your transaction was posted successfully.



Question View

Question Id: question-2880103

Question: If someone owns a piece of land, do they own it all the way to the center of the earth?

Owner	Status	Stake	Balance	Created
marka	CREATED	12	12	6/1/2018 3:18:10 PM

Due: 6/1/2018 3:30:00 PM

Id	Owner	Answer	Votes	Earnings	Engage
answer-930552	winnie	The land one owns is only till 100 feet under the earth. After that it becomes no-man's land.	0	0	👍 👎

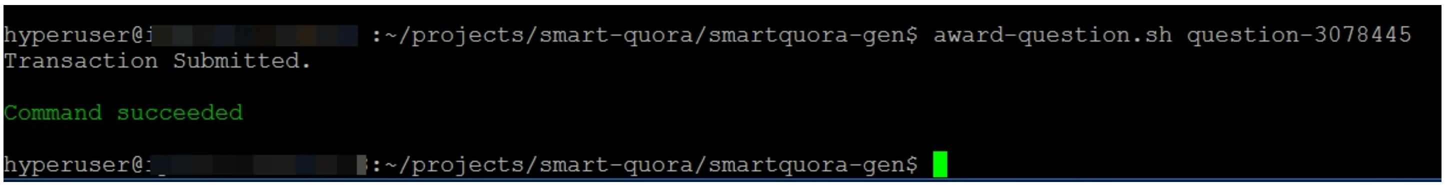
Your Answer

Award an Answer

Awarding an answer is an administrator function and has to be performed in the back-end. Ideally, this will be triggered by a lambda function that finds all the question that are past current time and invokes the award transaction. For now, adding a query to retrieve past due questions and award them through a lambda function is left as an exercise to the reader.

To award an answer, on the server go to the _smartquora-gen_directory and run the *award_question.sh* script. The script takes a question id as the only parameter.

```
cd <repo>/smartquora-gen
award-question.sh your-question-id
```



```
hyperuser@[redacted] :~/projects/smart-quora/smartquora-gen$ award-question.sh question-3078445
Transaction Submitted.

Command succeeded

hyperuser@[redacted]:~/projects/smart-quora/smartquora-gen$
```

Troubleshooting