

Hyperledger Fabric con Hyperledger Composer

Propósito

Este documento contiene los elementos esenciales para la implementación de una POC sobre la firma de documentos digitales construyendo una red de Hyperledger Fabric y almacenando los datos en una DLT con los conceptos de una Blockchain. La red está desarrollada con la herramienta de Hyperledger Composer para definir una red de negocios. La parte complementaria de la POC está implementada con una aplicación cliente en .net la cual se conecta con Hyperledger Fabric. La aplicación cliente consume las APIs expuestas para realizar la firma del documento, también consume ciertas APIs para hacer búsquedas dentro de blockchain y validar si un documento se encuentra firmado en Hyperledger Fabric. Este documento está hecho para exponer algunos casos de uso posibles con Hyperledger Fabric y Hyperledger Composer.

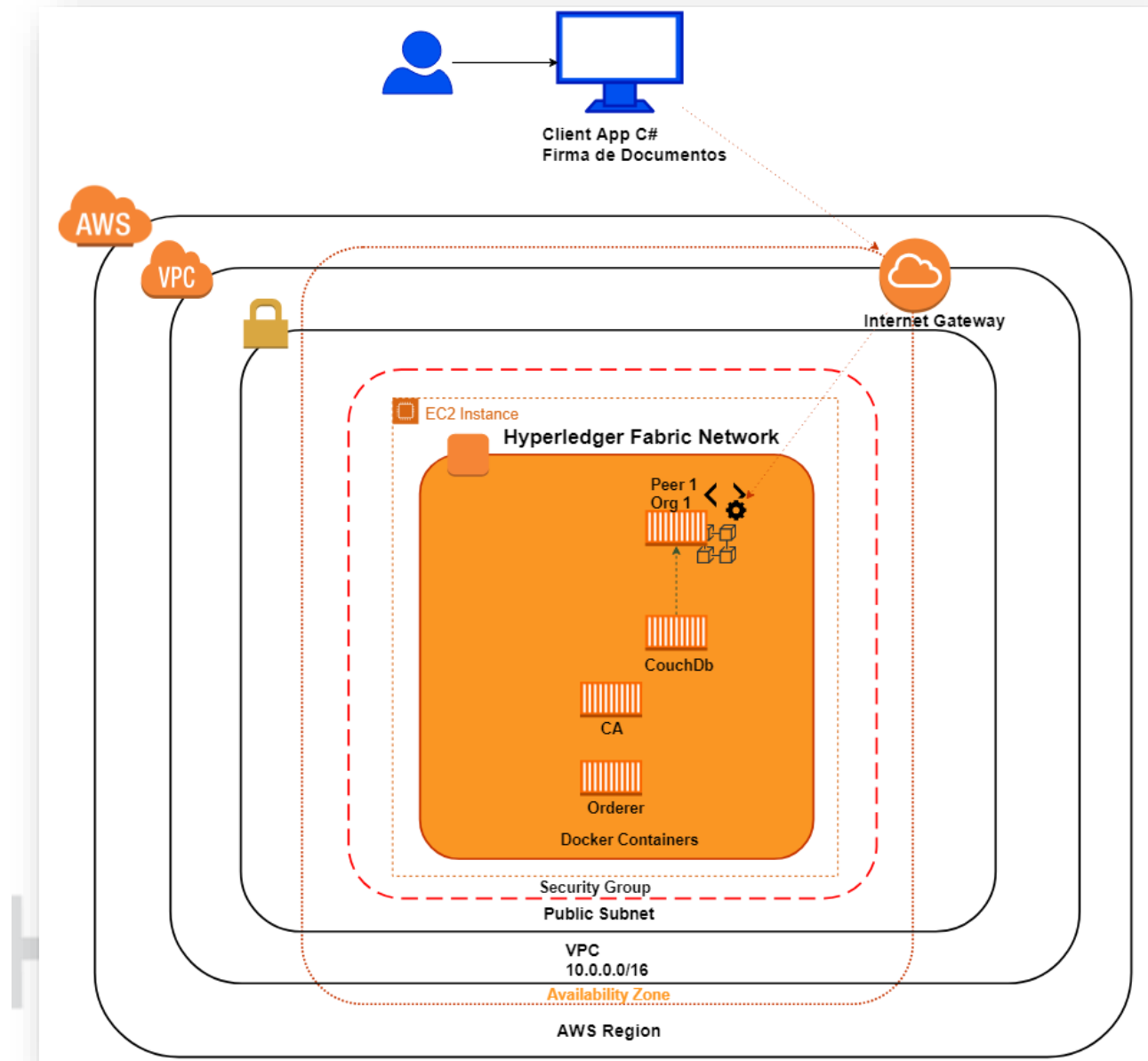
Importante:

Este documento contiene dos partes, la primera detalla la generación de un BND (Business Network Definition) para realizar la implementación de la red, la segunda contiene los pasos para construir una red de Hyperledger Fabric a partir de un BND y de esta manera agregar y consultar la información a través de las APIs desde una aplicación cliente desarrollada en .net WPF almacenada en Github.

Arquitectura

La arquitectura consiste en una instancia EC2 en AWS la cual contiene la infraestructura de la red de Hyperledger Fabric en contenedores de Docker. La instancia EC2 está dentro de un Security Group el cual a su vez se encuentra dentro de una Subnet. En AWS es necesario una VPC para construir una Subnet, la VPC se construye dentro de un segmento en una zona de disponibilidad ubicada en una región de AWS.

La aplicación cliente consume las APIs a través de una IP pública que expone un REST Server dentro de un contenedor o Peer 1 en el caso del diagrama. Las peticiones llegan a un Internet Gateway que redirecciona el tráfico hacia el Peer que contiene el servidor que está proporcionando las APIs dentro de la red de Hyperledger Fabric.



BND

El BND o Business Network Definition es un conjunto de archivos que definen la red de negocio en Hyperledger Fabric, el BND se construye mediante la herramienta denominada Hyperledger Composer y existen cuatro archivos importantes en Hyperledger Composer que debemos elaborar para que se defina la red completamente.

El model file o archivo de modelado es un archivo con extensión .cto en donde se definen los esquemas. Este lenguaje de modelado es orientado a objetos, donde se manejan nombres de espacio, variables, tipos de dato, herencia, importaciones, tipos primitivos, arreglos y otras características que no tocaremos a fondo en este documento. Los esquemas principales que se definen son: **transacciones**, **eventos**, **activos** y **participantes**. En esta POC definimos el activo principal que es el documento al cual identificamos por un id único. El participante lo definimos como firmante, este participante será quien realice el firmado de los documentos y al igual que el activo lo identificamos con un id único. Para las transacciones es necesario definir las partes que interactúan en la transacción, en nuestro caso lo que definimos es un activo y un participante que son los que mencionamos anteriormente como el firmante y el documento, la transacción consiste en agregar un firmante al documento. Finalmente, los eventos también deben definirse. Se define un evento que prácticamente se dispara cuando se agrega un firmante a un documento, lo que definimos es solamente una instancia del documento para que el evento detecte cuando el documento es modificado al agregar el firmante. A continuación, en la imagen podemos observar como definir los elementos y la sintaxis que utiliza Hyperledger Composer.

```
1  /**
2   * My asset signing network
3   */
4  namespace org.example.signedassetsnetwork
5
6  asset Document identified by idDocument {
7      o String idDocument
8      o String hashOriginal
9      --> Firmador firmador optional
10 }
11 participant Firmador identified by firmadorId {
12     o String firmadorId
13     o String name
14     o String lastName
15     o String certificate
16     o String firma
17 }
18
19 transaction AgregarFirmador {
20     --> Document document
21     --> Firmador firmador
22 }
23 event NotificacionFirmador {
24     --> Document document
25 }
```

Como se puede notar en la imagen, agregamos propiedades adicionales. En el activo almacenamos el hash original del documento y ponemos al firmante como opcional ya que primero generamos el documento y posteriormente le agregamos el firmante. El firmante también contiene otros parámetros que nos interesan como el nombre, certificado que usó para la firma y la firma del documento.

Script File

El archivo de script se encarga de la lógica de negocio, el script está programado en JavaScript y contiene la lógica de la transacción que se definió en nuestro modelo como Agregar Firmador. La transacción tiene como propósito agregar a un participante o firmante a un documento para mostrar que el archivo fue firmado por el mismo.

```
1  /**
2   * Add the Signer to the Document
3   * @param {org.example.signedassetsnetwork.AgregarFirmador} agregaFirmador - addsigner
4   * @transaction
5   */
6   async function agregaFirmador(agregaFirmador) {
7
8       // set the new signer of the document
9       agregaFirmador.document.firmador = agregaFirmador.firmador;
10      let assetRegistry = await getAssetRegistry('org.example.signedassetsnetwork.Document');
11
12      // emit a notification that a new signer has occurred
13      let addSignerNotification = getFactory().newEvent('org.example.signedassetsnetwork', 'NotificacionFirmador');
14      addSignerNotification.document = agregaFirmador.document;
15      emit(addSignerNotification);
16
17      // persist the state of the document
18      await assetRegistry.update(agregaFirmador.document);
19  }
```

En la imagen se muestra el código del script el cual realiza tres procesos importantes:

1. Agrega el firmante al documento
2. Emite una notificación sobre el evento ocurrido
3. Persiste el estado del documento

Query File

El archivo de consultas es otro de los archivos a implementar, como lo dice su nombre es útil para definir consultas que nos sirvan de acorde a lo que se esté desarrollando. En nuestro archivo de lo que se define son tres consultas.

```
1  /**
2   * Queries from Documents
3   */
4   query selectDocs {
5       description: "Select all documents"
6       statement:
7           SELECT org.example.signedassetsnetwork.Document
8   }
9
10  query selectDocsByOwner {
11      description: "Select all docs based on their owner"
12      statement:
13          SELECT org.example.signedassetsnetwork.Document
14             WHERE (firmador == _$firmador)
15  }
16
17  query selectDocsByHash {
18      description: "Select all docs based on their original hash"
19      statement:
20          SELECT org.example.signedassetsnetwork.Document
21             WHERE (hashOriginal == _$hashOriginal)
22  }
```



HYPERLEDGER

La primera consulta que se define es para extraer todos los documentos que se encuentren almacenados, al decir documentos no significa que son los documentos físicos, recordemos que estamos almacenando solamente ciertas propiedades de estos. La segunda consulta es para extraer los documentos en base al propietario o firmante. Finalmente, la última consulta es para obtener un documento basado en el hash original del mismo esto nos servirá para validar si un documento existe dentro de la red.

ACL

El último archivo necesario para tener nuestro BND completo es el Access Control List o ACL. En este archivo definimos los permisos de los participantes dentro de la red y las reglas que aplican para los recursos. No nos meteremos a fondo en este archivo ya que en esta POC solamente tenemos una organización. Por esta ocasión, daremos permiso al administrador de la red a todos los recursos por default.

```
15 rule NetworkAdminUser {
16     description: "Grant business network administrators full access to user resources"
17     participant: "org.hyperledger.composer.system.NetworkAdmin"
18     operation: ALL
19     resource: "***"
20     action: ALLOW
21 }
22
23 rule NetworkAdminSystem {
24     description: "Grant business network administrators full access to system resources"
25     participant: "org.hyperledger.composer.system.NetworkAdmin"
26     operation: ALL
27     resource: "org.hyperledger.composer.system.*)"
28     action: ALLOW
29 }
```

Conclusión

Ahora con el BND completo puedes empezar a jugar con [Hyperledger Composer Playground](#) en la creación de participantes, documentos, realizar la transacción de agregar un firmante y ver los eventos que se generan. Para ver la funcionalidad completa y visual será necesario la parte complementaria de este tutorial para montar tu propia red y visualizar las APIs REST para consumirlas directamente desde el navegador o un aplicativo.

Construir la red con Hyperledger Composer

Antes de empezar nosotros recomendamos usar la aplicación “bitvise ssh client” para manipular nuestra maquina ya que se encuentra en AWS. Descargamos la pem, y la agregamos en el client key manager y el usuario es Ubuntu.

Ya que témenos el archivo (.BNA) en nuestra computadora procedemos a construir la red donde lo vamos a montar, en este caso lo usaremos con Ubuntu 16 ya que es de las versiones probadas más estables.

a continuación en nuestra consola escribimos los siguientes comandos:

Este comando nos permite bajar todos los requisitos para la instalación de una red Hyperledger para poder desplegar nuestro archivo BNA o BND

```
$ curl -O https://hyperledger.github.io/composer/v0.19/prereqs-ubuntu.sh
$ chmod u+x prereqs-ubuntu.sh

$ ./prereqs-ubuntu.sh
```

Este comando nos instala:

Node: v8.16.0

npm: 6.4.1

Docker: Docker version 18.09.5, build e8ff056

Docker Compose: docker-compose version 1.13.0, build 1719ceb

Python: Python 2.7.12

Después de que se complete este comando hay que cerrar sesión en la consola y volverlo a abrir.

Agregamos también este comando que nos permite usar Docker sin necesitar root

```
$ sudo usermod -a -G docker $USER
```

Después instalamos los componentes de composer que nos ayudaran a gestionar la red

```
$ npm install -g composer-cli@0.19
$ npm install -g composer-rest-server@0.19
$ npm install -g generator-hyperledger-composer@0.19
$ npm install -g yo
```

Después creamos una carpeta para trabajar dentro de ella

```
$ mkdir ~/fabric-dev-servers && cd ~/fabric-dev-servers
```

ya dentro de la carpeta fabric-dev-servers descargamos todos los archivos necesarios para iniciar nuestra red.

```
$ curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz
```

Se descarga y luego la descomprimos con el siguiente comando:

```
$ tar -xvf fabric-dev-servers.tar.gz
```

ya que descomprimos todos los archivos necesarios tenemos que exportar la versión específica de la fabric que queremos usar en este caso será la 1.1.

```
$ export FABRIC_VERSION=hlfv11
```

Para descargar todos los archivos necesarios para levantar una red de Hyperledger Composer

```
$ sudo ./downloadFabric.sh
```

hasta este momento hemos instalado todo lo necesario para el entorno de desarrollador típico. Vamos a seguir al siguiente paso.

Controlando tu entorno de desarrollo

Para iniciar con la red ejecutamos el siguiente comando:

```
$ sudo ./startFabric.sh
```

luego de que la red está funcionando tenemos que crear credenciales para este nuevo Peer.

```
$ ./createPeerAdminCard.sh
```

Nos generara las credenciales necesarias para comunicarnos con los archivos del composer y manipular el funcionamiento y creación de la red.

Hyperledger fabric por defecto crea credenciales desde que iniciamos la red, las cuales nos sirven para comenzar a manipularla **PeerAdmin@hlfv1**

```
Successfully created business network card file to
  Output file: /tmp/PeerAdmin@hlfv1.card

Command succeeded

Successfully imported business network card
Card file: /tmp/PeerAdmin@hlfv1.card
Card name: PeerAdmin@hlfv1

Command succeeded

The following Business Network Cards are available:
Connection Profile: hlfv1
```

Card Name	UserId	Business Network
PeerAdmin@hlfv1	PeerAdmin	

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

Crear una estructura de negocios

El concepto clave para Hyperledger Composer es la definición de red de negocios (BND). Define el modelo de datos, la lógica de transacción y las reglas de control de acceso para su solución de blockchain. Para crear un BND, necesitamos crear una estructura de proyecto adecuada en el disco.

La forma más fácil de comenzar es usar el generador de Yeoman para crear una red empresarial esquelética. Esto creará un directorio que contiene todos los componentes de una red de negocios.

\$yo hyperledger-composer:businessnetwork

cuando ejecutemos este comando nos pedirá los siguientes requisitos

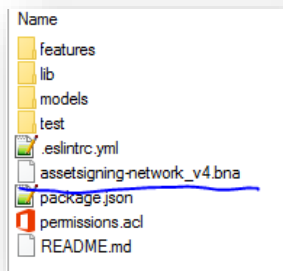
(para este tutorial hay que asignar los mismos valores en verde)

- 1 Business network name: **assetSigning-network_v4**
- 2 Description: **blockchain Hyperledger de prueba version 1**
- 3 Author name: **(Nombres Opcionales)**
- 4 Author email: **Hyperledger@latam.com**
- 5 License: **Apache-2.0**
- 6 Namespace: **org.example.signedassetsnetwork**
- 7 Do you want to generate an empty template network? (Use arrow keys)
 - Yes: generate an empty template network
 - > No: generate a populated sample network**

Y con esta información ya creamos nuestra red de negocio :D

```
drwxrwxr-x 4 ubuntu ubuntu 4096 Apr 22 14:41 ./
drwxr-xr-x 10 ubuntu ubuntu 4096 Apr 22 14:41 ../
drwxrwxr-x 6 ubuntu ubuntu 4096 Apr 22 14:41 assetSigning-network_v4/
-rwxr-xr-x 1 ubuntu ubuntu 726 Sep 29 2018 createComposerProfile.sh*
-rwxr-xr-x 1 ubuntu ubuntu 625 Sep 29 2018 createPeerAdminCard.sh*
-rw-rw-r-- 1 ubuntu ubuntu 1299 Apr 22 14:30 DevServer_connection.json
-rwxr-xr-x 1 ubuntu ubuntu 625 Sep 29 2018 downloadFabric.sh*
-rw-rw-r-- 1 ubuntu ubuntu 29471 Apr 22 14:26 fabric-dev-servers.tar.gz
drwxrwxr-x 5 ubuntu ubuntu 4096 Apr 22 14:27 fabric-scripts/
-rwxr-xr-x 1 ubuntu ubuntu 1531 Sep 29 2018 _loader.sh*
-rw-r--r-- 1 ubuntu ubuntu 1305 Oct 12 2018 package.json
-rwxr-xr-x 1 ubuntu ubuntu 743 Sep 29 2018 startFabric.sh*
-rwxr-xr-x 1 ubuntu ubuntu 625 Sep 29 2018 stopFabric.sh*
-rwxr-xr-x 1 ubuntu ubuntu 1573 Sep 29 2018 teardownAllDocker.sh*
-rwxr-xr-x 1 ubuntu ubuntu 625 Sep 29 2018 teardownFabric.sh*
```

Ahora dentro de nuestro fichero establecemos nuestro archivo BNA creado anteriormente.



Despliegue de la red

La implementación de una red de negocios en el Hyperledger Fabric requiere que la red de negocios de Hyperledger Composer se instale en el Peer, luego se puede iniciar la red de negocios y se debe crear una nueva associated card, y asociarla para que sea el administrador de la red. Finalmente, la tarjeta de red empresarial del administrador de red debe importarse para su uso, y luego se puede hacer ping a la red para verificar que está respondiendo.

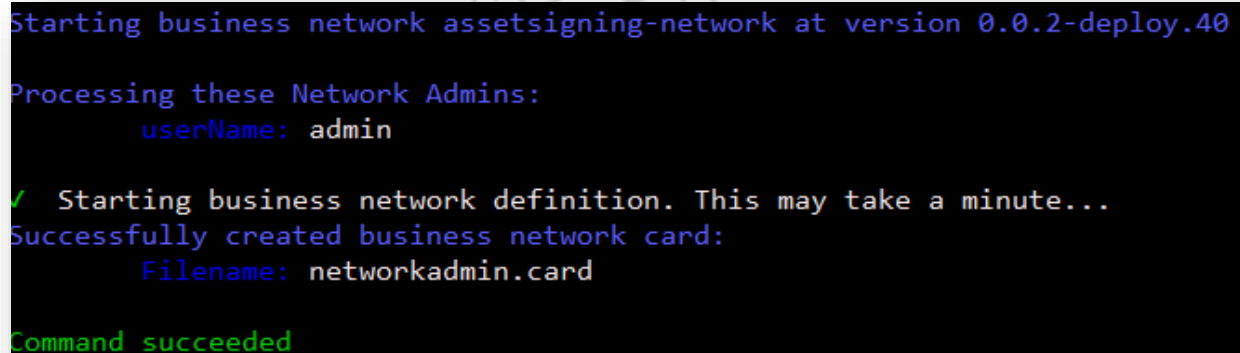
Debemos situarnos dentro del directorio para ejecutar el siguiente comando:

```
$ composer network install --card PeerAdmin@hlfv1 --archiveFile assetsigning-network_v4.bna
```

ahora ya que instalamos la tarjeta de credenciales al .BNA, vamos a deployar la red con el siguiente comando:

```
$ composer network start --networkName assetsigning-network --networkVersion 0.0.2-deploy.40 --  
networkAdmin admin --networkAdminEnrollSecret adminpw --card PeerAdmin@hlfv1 --file networkadmin.card
```

si todo sale bien veremos algo como lo siguiente:



```
Starting business network assetsigning-network at version 0.0.2-deploy.40  
  
Processing these Network Admins:  
  userName: admin  
  
✓ Starting business network definition. This may take a minute...  
Successfully created business network card:  
  Filename: networkadmin.card  
  
Command succeeded
```

El composer network start requiere una tarjeta de red de negocios, así como el nombre de la identidad de administrador de la red de negocios, el nombre y la versión de la red de negocios y el nombre del archivo que se creará listo para importar como una tarjeta de red de negocios.

Para importar la identidad del administrador de red como una tarjeta de red comercial utilizable, ejecute el siguiente comando:

```
$ composer card import --file networkadmin.card
```

Para verificar que la red de negocios se haya implementado correctamente, ejecute el siguiente comando para hacer ping a la red:

```
$ composer network ping --card admin@assetsigning-network
```

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

Command succeeded

ubuntu@ip-172-31-89-240:~/fabric-dev-servers/assetsigning-network_v4$ composer network ping --card admin@assetsigning-network
The connection to the network was successfully tested: assetsigning-network
  Business network version: 0.0.2-deploy.40
  Composer runtime version: 0.19.20
  participant: org.hyperledger.composer.system.NetworkAdmin#admin
  identity: org.hyperledger.composer.system.Identity#5653a20a100523156a1ea551b3e821a3678210b439f4c68fe80ffca0879e3f0c

Command succeeded
```

Ya que tenemos una red trabajando con una tarjeta de identificación dentro de un Peer, vamos a crear un servidor Rest, y después una aplicación que corra con este servidor.

\$ composer-rest-server

Y seleccionamos las siguientes opciones:

- ? Enter the name of the business network card to use: **admin@assetsigning-network**
- ? Specify if you want namespaces in the generated REST API: **never use namespaces**
- ? Specify if you want to use an API key to secure the REST API: **No**
- ? Specify if you want to enable authentication for the REST API using Passport: **No**
- ? Specify if you want to enable the explorer test interface: **Yes**
- ? Specify a key if you want to enable dynamic logging:
- ? Specify if you want to enable event publication over WebSockets: **Yes**
- ? Specify if you want to enable TLS security for the REST API: **No**

En este momento, ya tenemos nuestro servidor rest, corriendo en nuestro localhost:3000

Se vería de la siguiente manera:

```
To restart the REST server using the same options, issue the following command:
composer-rest-server -c admin@assetsigning-network -n never -u true -w true

Discovering types from business network definition ...
Discovering the Returning Transactions..
Discovered types from business network definition
Generating schemas for all types in business network definition ...
Registering named query: selectDocs
Registering named query: selectDocsByOwner
Registering named query: selectDocsByHash
Generated schemas for all types in business network definition
Adding schemas for all types to Loopback ...
Added schemas for all types to Loopback
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
```

Hyperledger Composer REST server		
AgregarFirmador : A transaction named AgregarFirmador	ShowHide	List Operations Expand Operations
Document : An asset named Document	ShowHide	List Operations Expand Operations
Firmador : A participant named Firmador	ShowHide	List Operations Expand Operations
Query : Named queries	ShowHide	List Operations Expand Operations
System : General business network methods	ShowHide	List Operations Expand Operations
[BASE URL: /api , API VERSION: 0.0.1]		

En este momento podemos interactuar directamente con la red de Hyperledger Composer desde nuestro navegador, en este caso en especial creamos una aplicación .exe, con la cual interactuamos con la red.

Consumo de APIs desde una aplicación Desktop

Una vez que tenemos el servidor Rest de Composer arriba podemos ver detalladamente las APIs, existen APIs para la transacción, assests, participantes y queries. Esto quiere decir que podemos interactuar con la red mediante todas estas APIs.

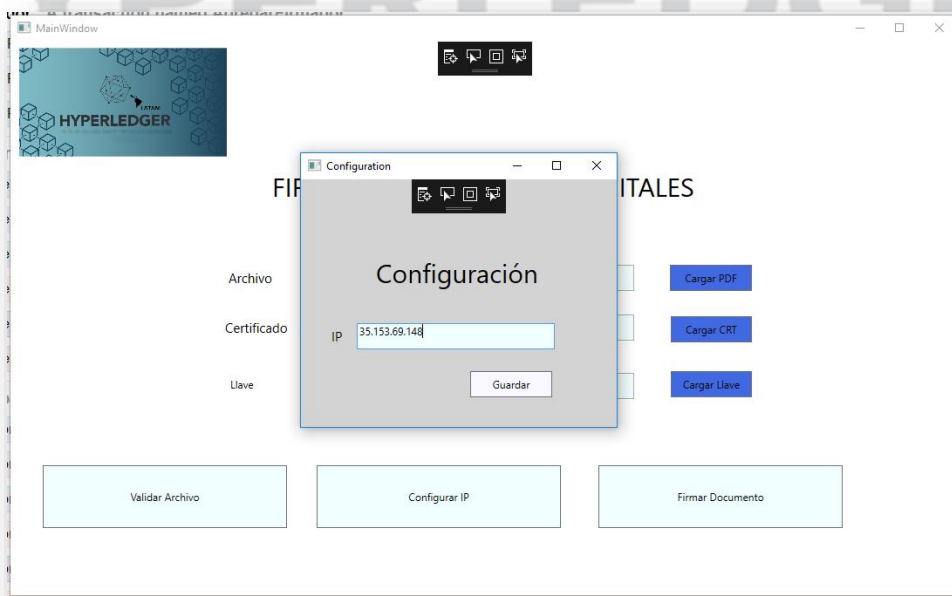
AgregarFirmador : A transaction named AgregarFirmador	
GET	/AgregarFirmador
POST	/AgregarFirmador
GET	/AgregarFirmador/{id}
Document : An asset named Document	
GET	/Document
POST	/Document
GET	/Document/{id}
HEAD	/Document/{id}
PUT	/Document/{id}
DELETE	/Document/{id}
Firmador : A participant named Firmador	
GET	/Firmador
POST	/Firmador
GET	/Firmador/{id}
HEAD	/Firmador/{id}
PUT	/Firmador/{id}
DELETE	/Firmador/{id}
Query : Named queries	

Una de las características de blockchain es la inmutabilidad de los datos, como se puede observar en la API de la transacción no existe un método Delete como en las demás, esto es porque una transacción no se puede borrar. En Hyperledger existe una base de estado que se complementa con el libro contable y en el caso de los activos y participantes podemos ejecutar el método de borrado, sin embargo, esto no quiere decir que se borre el registro por completo, en el historial de transacciones existe la trazabilidad de quien realiza estas modificaciones.

La aplicación desktop nos permite hacer el consumo de los métodos anteriores desde una interfaz gráfica, lo primero que se realiza es la carga del PDF, certificado y llave privada. Normalmente un certificado se verifica con una cadena de confianza que apunta al certificado root para validar su emisión. En nuestro caso se hace localmente ya que los certificados creados son locales usando la herramienta de OPENSSL.



Una vez que tenemos cargados los archivos podemos realizar la firma del documento, pero antes configuraremos la IP publica en donde se encuentran las APIs.

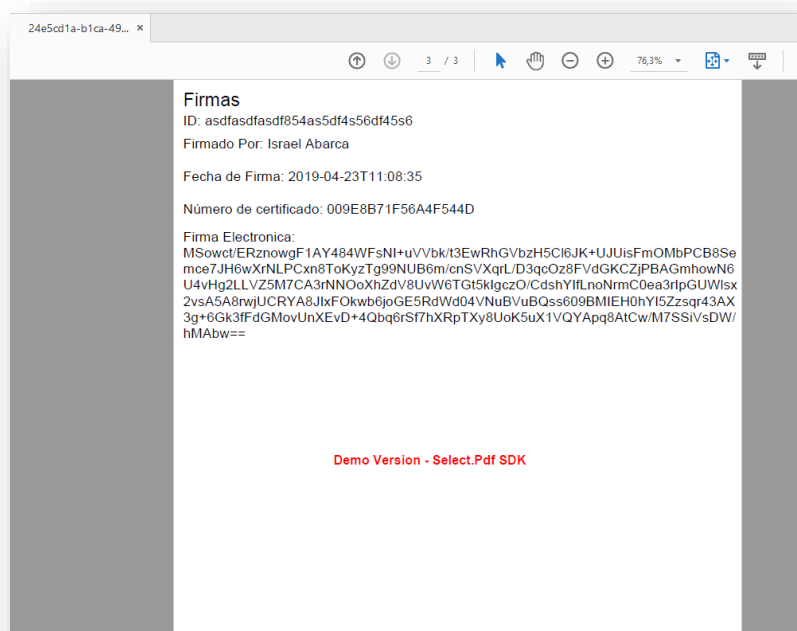


Una vez configurada la IP procedemos a dar clic en el botón de [Firmar Documento] si la ejecución es correcta se mostrará el mensaje de éxito como se puede observar en la imagen de abajo.

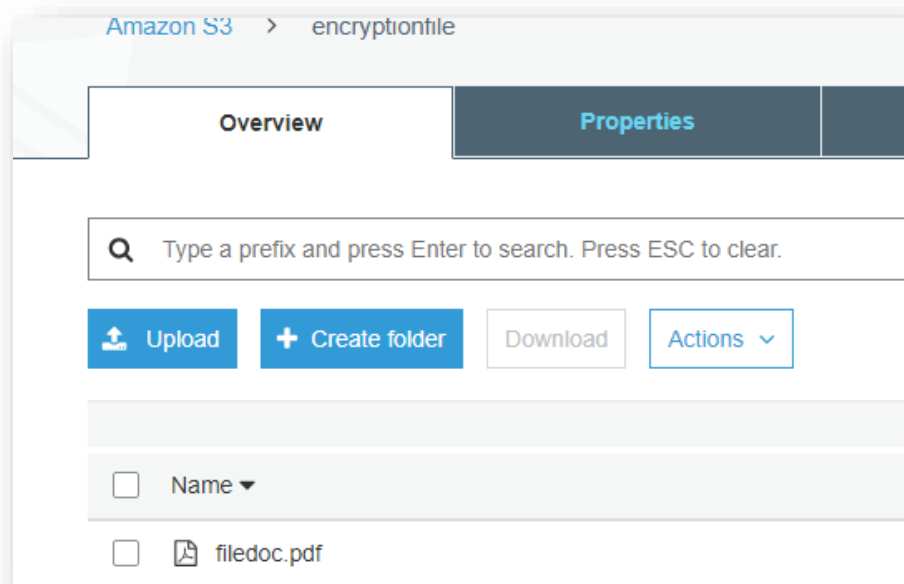


El proceso de firmado consiste en cuatro procesos importantes los cuales mencionaremos a continuación:

1. Generación de un PDF con la información del documento original y anexo una hoja sobre el detalle de la firma de este.



3. Recordemos que Blockchain no es la mejor opción para almacenar archivos pesados, sin embargo, podemos almacenar en servicios como S3 para tener el archivo resguardado.



4. Almacenamos la información en Blockchain para mantener un registro y trazabilidad. Así mismo si tuviéramos otras organizaciones conectadas sobre un canal, estas otras organizaciones pueden consultar la información y validar las transacciones que se agreguen al libro o ledger.

Hyperledger Composer REST server

```
[
  {
    "$class": "org.example.signedassetsnetwork.AgregarFirmador",
    "document": {},
    "firmador": {},
    "transactionId": "string",
    "timestamp": "2019-04-23T15:59:41.975Z"
  }
]
```

Response Content Type

Parameters

Parameter	Value	Description
filter	<input type="text"/>	Filter defining fields, where, include, order, offset, and limit - ([("something"."value")])

[Try it out!](#) [Hide Response](#)

Curl

```
curl -X GET --header 'Accept: application/json' 'http://35.153.69.148:3000/api/AgregarFirmador'
```

Request URL

```
http://35.153.69.148:3000/api/AgregarFirmador
```

Response Body

```
[
  {
    "$class": "org.example.signedassetsnetwork.AgregarFirmador",
    "document": "resource:org.example.signedassetsnetwork.Document#daf38c69-acha-4c49-989e-36c7e924e084",
    "firmador": "resource:org.example.signedassetsnetwork.Firmador#b61d5a89-845f-4a6f-a266-575f1c159bcb",
    "transactionId": "650004ac59096a77e1e68a81ab4eba762a15946d3dc4391bf2552a67c2d02d9d",
    "timestamp": "2019-04-23T16:13:49.714Z"
  }
]
```

Response Code

Como se observa en la imagen si ejecutamos el GET del método Agregar Firmador nos arroja la información relevante de quien firmó el documento, así como la transacción y la fecha de esta misma.

Finalmente, los queries que construimos en el BNA también podemos consumirlos mediante las APIs. Uno de los queries era extraer todos los documentos y si lo ejecutamos nos trae la información de todos los documentos que hemos agregado.

Firmador : A participant named Firmador

Query : Named queries

GET /queries/selectDocs

Response Class (Status 200)
Request was successful

Model Example Value

```
[
  {
    "$class": "org.example.signedassetsnetwork.Document",
    "idDocument": "string",
    "hashOriginal": "string",
    "firmador": {}
  }
]
```

Response Content Type application/json

Try it out! [Hide Response](#)

Curl

```
curl -X GET --header 'Accept: application/json' 'http://35.153.69.148:3000/api/queries/selectDocs'
```

Request URL

```
http://35.153.69.148:3000/api/queries/selectDocs
```

Response Body

```
[
  {
    "$class": "org.example.signedassetsnetwork.Document",
    "idDocument": "daf38c69-acba-4c49-989e-36c7e924e084",
    "hashOriginal": "ee79f61c38a613f1e36c913e8eeae707ee49b82329c1d2268e164321ed2c27f",
    "firmador": "resource:org.example.signedassetsnetwork.Firmador#b61d5a89-845f-4a6f-a266-575f1c159bcb"
  }
]
```

El query selectDocs nos arroja la información del documento que acabamos de firmar, si comparamos el id del documento es el mismo y el hash original también concuerda con la información que se genera en el archivo JSON.

Por último, una funcionalidad agregada que expusimos es validar si un archivo fue firmado y se encuentra en nuestra red, para esto cargamos el mismo archivo pdf y damos clic en el botón de [Validar Archivo], lo que pasa detrás es que la aplicación extrae el hash del archivo y consume la API del query que busca los documentos por su hash original, si lo encuentra nos muestra el mensaje que muestra la imagen de debajo de que el documento si existe y nos muestra el id del mismo.



MainWindow

FIRMA DE DOCUMENTOS DIGITALES

Archivo: C:\Users\Israel Abarca\Desktop\Proyecto Certificados (Excel).pdf Cargar PDF

Certificado: C:\Users\Israel Abarca\Computación en Acción, S.A. de C.V.
\\ITServitv - Documentos\Blockchain\Hunerledner Fabric\Certs. Cargar CRT

Llave: C:\Users\Israel Abarca\Computación en Acción, S.A. de C.V.
\\ITServitv - Documentos\Blockchain\Hunerledner Fabric\Certs. Cargar Llave

Validar Archivo Firmar Documento

Documento encontrado:
resource:org.example.signedassetsnetwork.Firmador#b61d5a89-845f-4
a6f-a266-575f1c159bcb
Id del Documento: daf38c69-acba-4c49-989e-36c7e924e084

Aceptar

C:\Users\Israel Abarca\Desktop\Proyecto Certificados (Excel).pdf



HYPERLEDGER