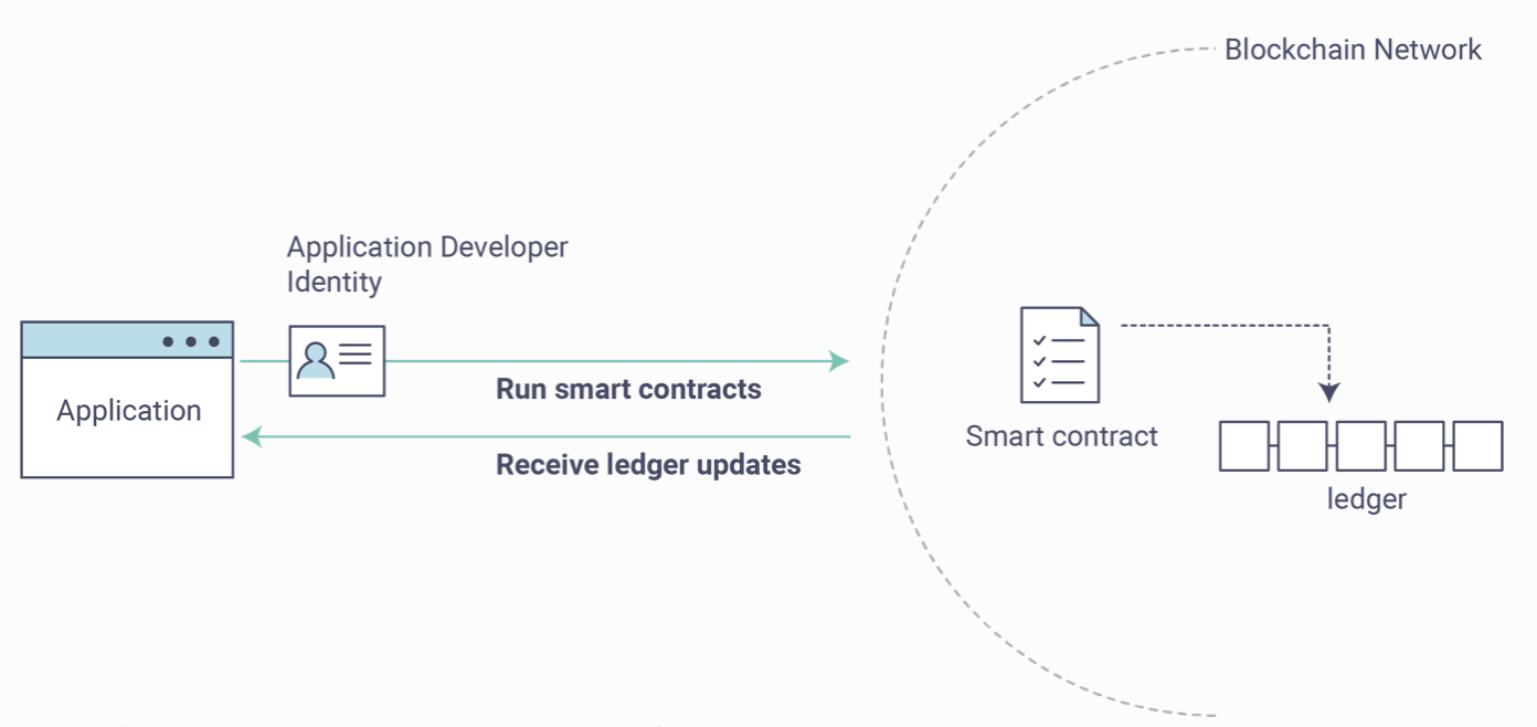
Client application & Commercial paper

Fundamentos de blockchains

René Dávila - Jorge Solano

Índice

Comunicación entre una aplicación cliente y la red de blockchain.



https://hyperledger-fabric.readthedocs.io/en/release-2.0/developapps/application.html

Levantar la red blockchain

En la ubicación cd fabric-samples/fabcar se levanta la red **FabCar**

```
$ ./startFabric.sh javascript
```

Levantar la red blockchain

./startFabric.sh javascript realiza las siguientes tareas:

- Despliega la red **test-network** de Fabric (2 peer nodes y un ordering service).
- Utiliza **autoridades certificadoras** (CA) para crear los certificados y llaves de la aplicación.
- **Instala el smart contract** de FabCar en el canal mychannel en su versión en JavaScript.
- **Ejecuta el smart contract** para traer los datos iniciales al ledger.

Instalar la aplicación

En la carpteta cd fabric-samples/fabcar/javascript se encuentran los programas que serán ejecutados por node.js. Para ello hay que instalar las dependencias necesarias.

```
$ npm install
```

Usuario administrador

Se debe generar la llave pública, la llave privada y el certificado X.509 para admin utilizando el programa enroll.js. Este proceso utiliza una **Solicitud de firma de certificado**, primero se generan las llaves públicas y privadas de manera local y luego se envía la llave pública a la CA, la cual regresa un certificado codificado para ser usado por la aplicación.

```
$ node enrollAdmin.js
```

```
$ ls wallet/
```

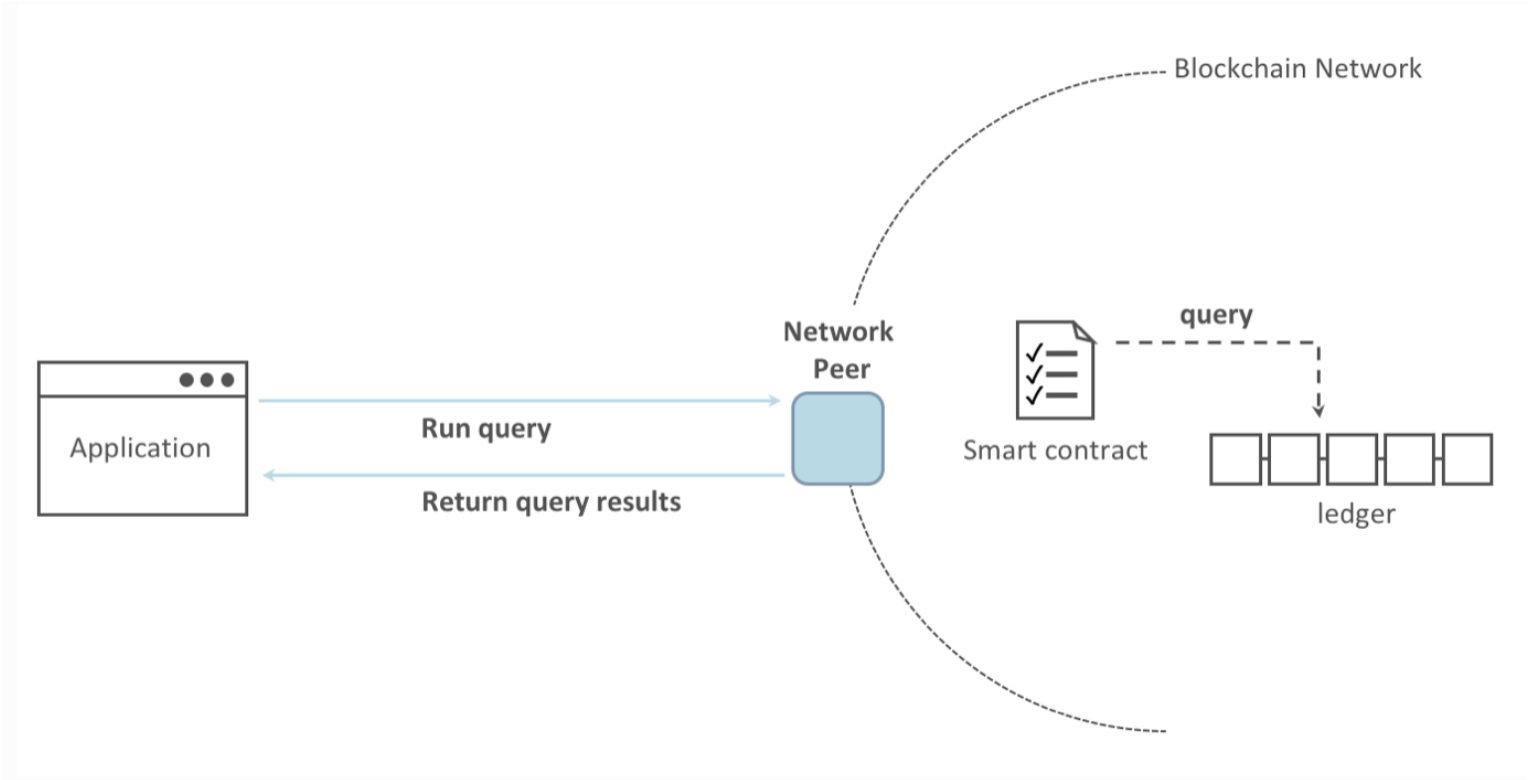
Usuario cliente

Ahora hay que crear una aplicación cliente para interactuar con la block-chain.

```
$ node registerUser.js
```

```
$ ls wallet/
```

Solicitar el ledger

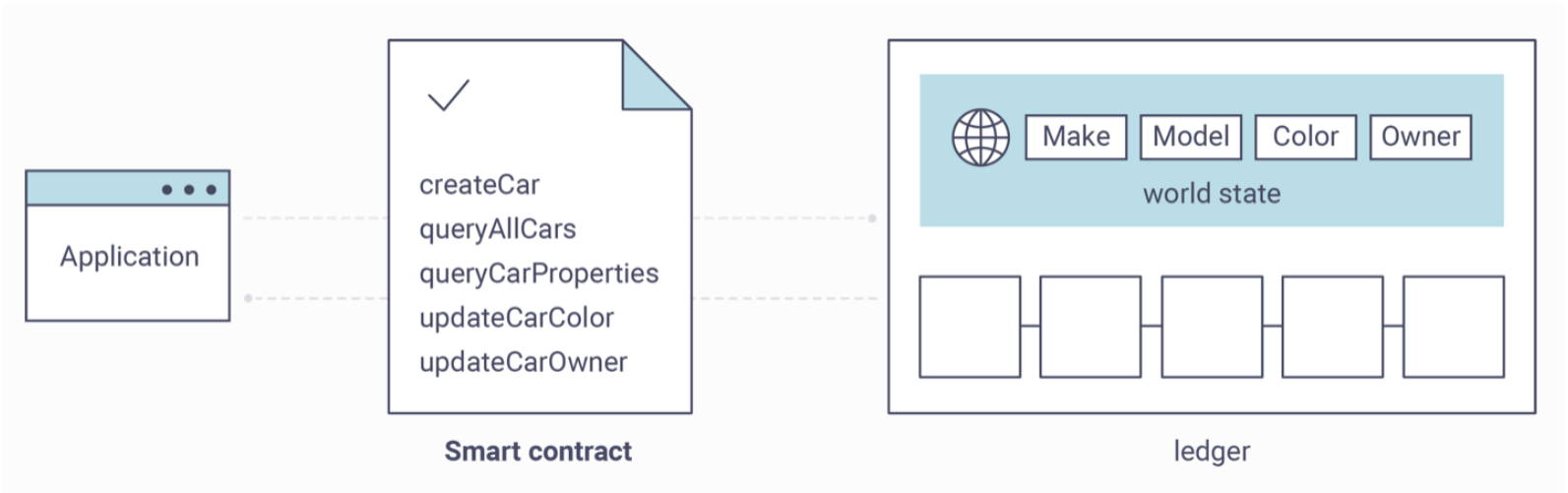


Los valores actuales del ledger están en el **world state**. El world state está representado como un conjunto llave-valor.

Aplicación cliente

El usuario appUser realiza la solicitud de todos los carros del ledger.

```
$ node query.js
```

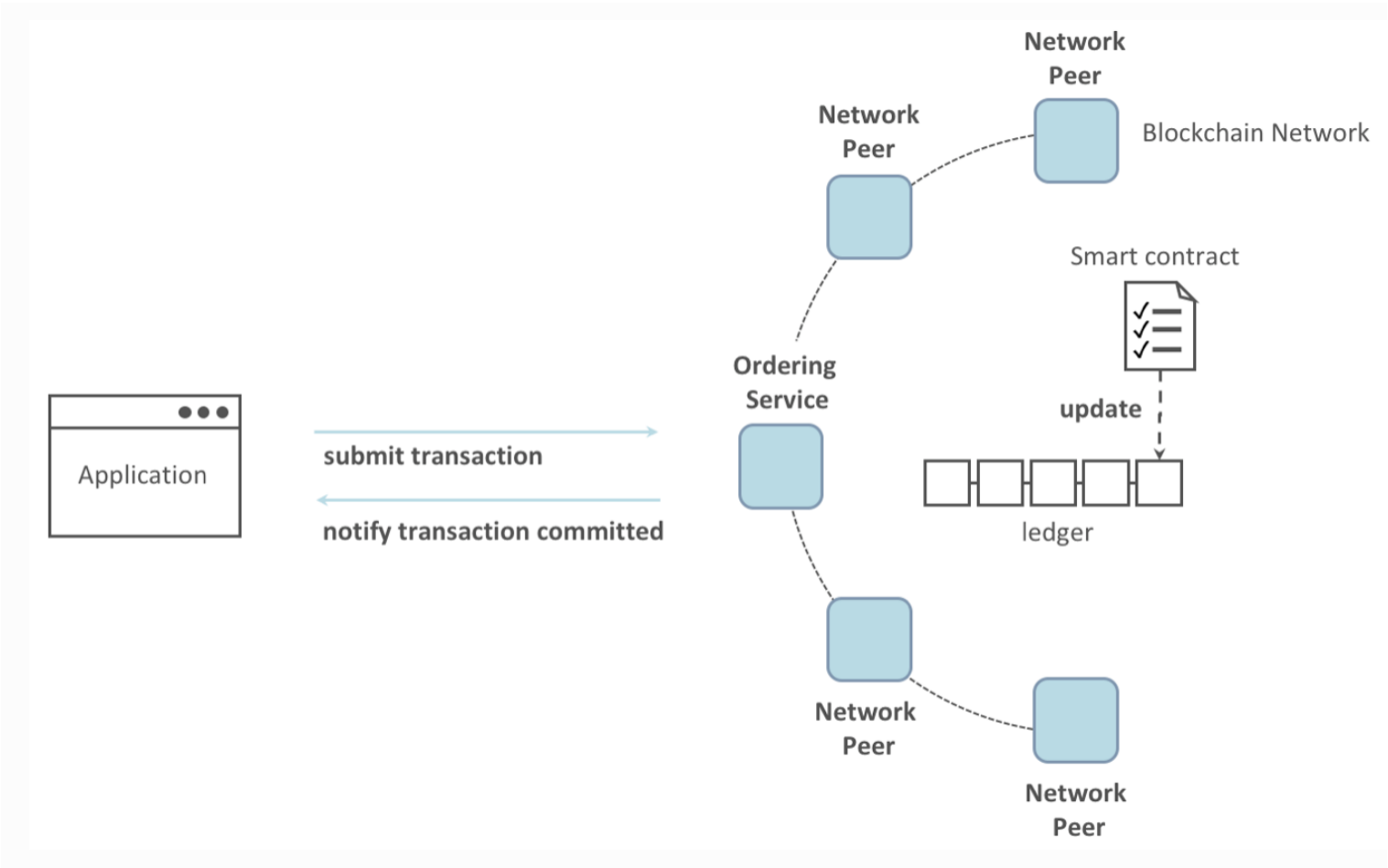


Contrato FabCar

En cd fabric-samples/chaincode/fabcar/javascript/lib se puede ver el smart contract de FabCar.

```
$ code fabcar.js
```

Actualizar el ledger



Crear un nuevo carro

Vamos a crear un nuevo carro, para ello en cd fabric-samples/fabcar/javascri
se tiene el programa invoke.js.

```
$ code invoke.js
```

```
$ node invoke.js
```

Cambio de dueño

Vamos a cambiar de dueño al nuevo carro que se creó, para ello en cd fabric-samples/fabcar/javascript se tiene el programa invoke.js en lugar de createCar se invocará changeCarOwner.

```
$ await contract.submitTransaction('changeCarOwner', 'CAR12',
'Dave');
```

```
$ node invoke.js
```

Limpiar

Para terminar en cd fabric-samples/fabcar, es necesario dar de baja la red, las CA, los nodos peer y ordering y eliminar los wallet de usuario y administrador.

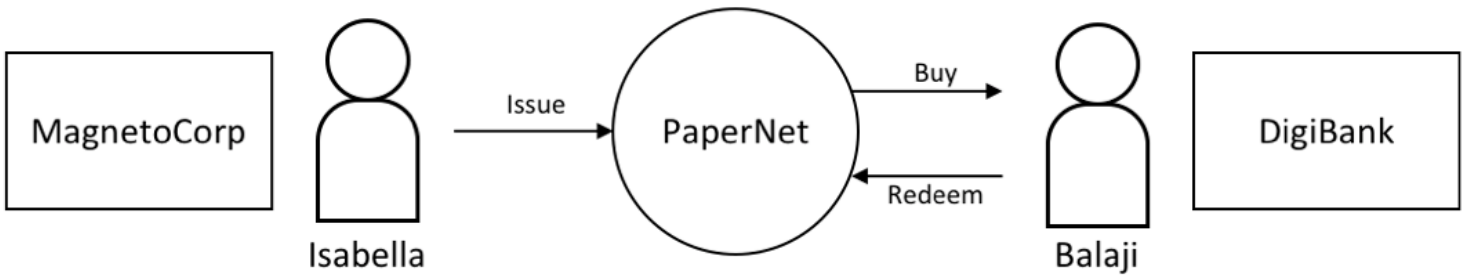
```
$ ./networkDown.sh
```

Referencia

https://hyperledger-fabric.readthedocs.io/en/release-2.0/write_first_app.html

Índice

Diagrama de las organizaciones:



Las organizaciones MagnetoCorp y DigiBank intercambian **papeles comerciales** en la red **PaperNet**.

Índice

Peers y canal de comunicación

Red docker net_test

En cd fabric-samples/commercial-paper

```
(PaperNet admin)$ ./network-starter.sh
```

```
(PaperNet admin)$ docker ps
```

```
(PaperNet admin)$ docker network inspect net_test
```

peer0.org1.example.com → DigiBank
peer0.org2.example.com → MagnetoCorp

Red docker net_test

Es importante validar que en la variable PATH se encuentre la carpeta **bin** de fabric-samples y que en la variable FABRIC_CFG_PATH se encuentre la carpeta **config**.

```
(PaperNet admin)$ export PATH=$PWD/../bin:$PWD:$PATH
(PaperNet admin)$ export FABRIC_CFG_PATH=$PWD/../config/
```

Monitor de la red

Tanto el administrador de la red, como los administradores de cada organización pueden revisar la bitácora de actividades de la red constantemente. Ahora jugaremos el rol de Administrador de MagnetoCorp para monitorear la red, cd commercial-paper/organization/magnetocorp ejecutar:

```
(MagnetoCorp admin)$ ./configuration/cli/monitordocker.sh net_test
```

Índice

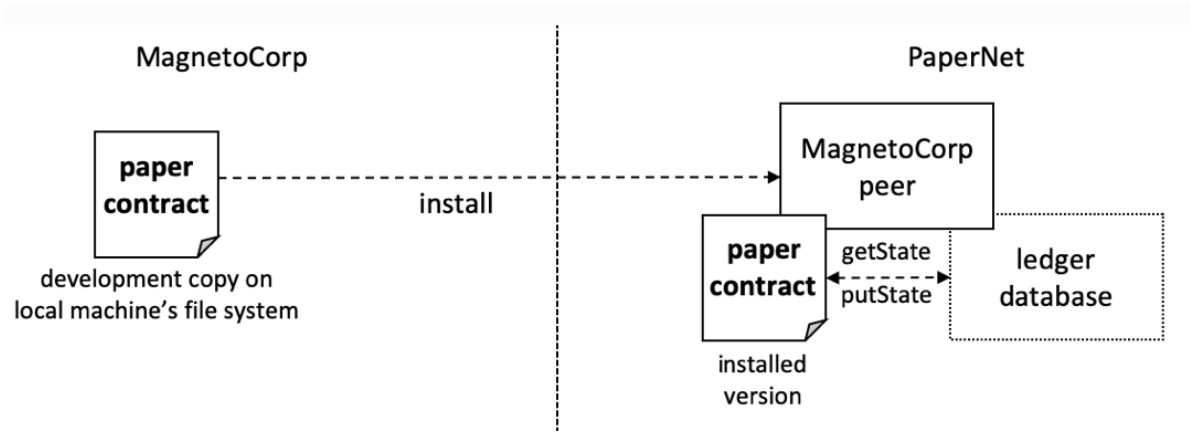
Creación del contrato

En cd commercial-paper/organization/magnetocorp

```
(MagnetoCorp dev)$ code contract
```

Para ver cómo fue diseñado papercontract.js más a detalle se puede consultar: `https://hyperledger-fabric.readthedocs.io/en/release-2.0/developapps/smartcontract.html`

Ciclo de vida en Fabric



- Organización emisora
  - El desarrollador empaqueta el chaincode ([1...n] smart contract)
  - El administrador instalar el chaincode en cada organización.
- Organización receptora
  - El administrador debe aprobar el chaincode.
  - El administrador publica el chaincode en el ledger del canal asociado.

`https://hyperledger-fabric.readthedocs.io/en/release-2.0/chaincode_lifecycle.html#chaincode-lifecycle`

Instalación

Variables de entorno: En cd commercial-paper/organization/magnetocorp hay que establecer las variables de entorno para que el administrador pueda instalar y aprobar el chaincode en la organización MagnetoCorp.

```
(MagnetoCorp admin)$ source magnetocorp.sh
```

Instalación

Empaquetar el chaincode: Para instalar el contrato, primero se debe empaquetar el smart contract en un chaincode.

```
(MagnetoCorp admin)$ peer lifecycle chaincode package cp.tar.gz -
-lang node - -path ./contract - -label cp_0
```

Instalación

Instalar el chaincode: Una vez generado el paquete se puede instalar en el peer node de la organización.

```
(MagnetoCorp admin)$ peer lifecycle chaincode install cp.tar.gz
```

Aprobación

Package ID: Para poder aprobar el contrato primero se debe obtener el packageID del chaincode que se acaba de instalar.

```
(MagnetoCorp admin)$ peer lifecycle chaincode queryinstalled
```

Aprobación

Guardar el package ID: El package ID se va a ocupar en el siguiente paso, se puede guardar en una variable de entorno.

```
(MagnetoCorp admin)$ export PACKAGE_ID=cp_0:ffda9...
```

Aprobación

Aprobar el chaincode: El administrador de MagnetoCorp debe aprobar el papercontract.

```
(MagnetoCorp admin)$ peer lifecycle chaincode approveformyorg -
-orderer localhost:7050 - -ordererTLSHostnameOverride
orderer.example.com - -channelID mychannel - -name papercontract -v
0 - -package-id $PACKAGE_ID - -sequence 1 - -tls - -cafile
$ORDERER_CA
```

Aprobación

La política de aprobación define el número de organizaciones que debe endosar (ejecutar y firmar) una transacción antes de determinarse como válida.

El contrato se aprobó por MagnetoCorp con la política por defecto de la red, la cual requiere que la mayoría de las organizaciones validen la transacción. Todas las transacciones, válidas o inválidas, se guardan en la blockchain, pero solo las válidas modifican el estado (**world state**).

`https://hyperledger-fabric.readthedocs.io/en/release-2.0/ledger/ledger.html`

Índice

Instalación

Por defecto, el ciclo de vida del chaincode en Fabric requiere que la mayoría de las organizaciones apruebe la definición de un chaincode en el canal. En este caso, se tienen 2 organizaciones, por lo tanto, se requiere que las 2 organizaciones apruebe el chaincode.

Instalación

Variables de entorno: En cd commercial-paper/organization/digibank/ hay que establecer las variables de entorno para que el administrador de DigiBank pueda instalar y aprobar el papernet chaincode.

```
(DigiBank admin)$ source digibank.sh
```

Instalación

Empaquetar el chaincode: El smart contract se debe empacar en un chaincode.

```
(DigiBank admin)$ peer lifecycle chaincode package cp.tar.gz - -lang
node - -path ./contract - -label cp_0
```

Instalación

Instalar el chaincode: Ahora se puede instalar el chaincode empaquetado en la organización.

```
(DigiBank admin)$ peer lifecycle chaincode install cp.tar.gz
```

Aprobación

Package ID: Para poder aprobar el contrato primero se debe obtener el packageID del chaincode que se acaba de instalar.

```
(DigiBank admin)$ peer lifecycle chaincode queryinstalled
```

Aprobación

Guardar el package ID: El package ID se va a ocupar en el siguiente paso, se puede guardar en una variable de entorno.

```
(DigiBank admin)$ export PACKAGE_ID=cp_0:ffda9...
```

Aprobación

Aprobar el chaincode: El administrador de DigiBank debe aprobar el papercontract.

```
(DigiBank admin)$ peer lifecycle chaincode approveformyorg –orderer
localhost:7050 - -ordererTLSHostnameOverride orderer.example.com -
-channelID mychannel - -name papercontract -v 0 - -package-id
$PACKAGE_ID - -sequence 1 - -tls - -cafile $ORDERER_CA
```

Índice

Publicación

Una vez que el contrato ha sido aprobado por la mayoría del consorcio (2/2) en el canal, se envía la definición del chaincode al canal.

Ahora, el contrato inteligente CommercialPaper puede ser invocado por alguna aplicación cliente en el canal.

Publicación

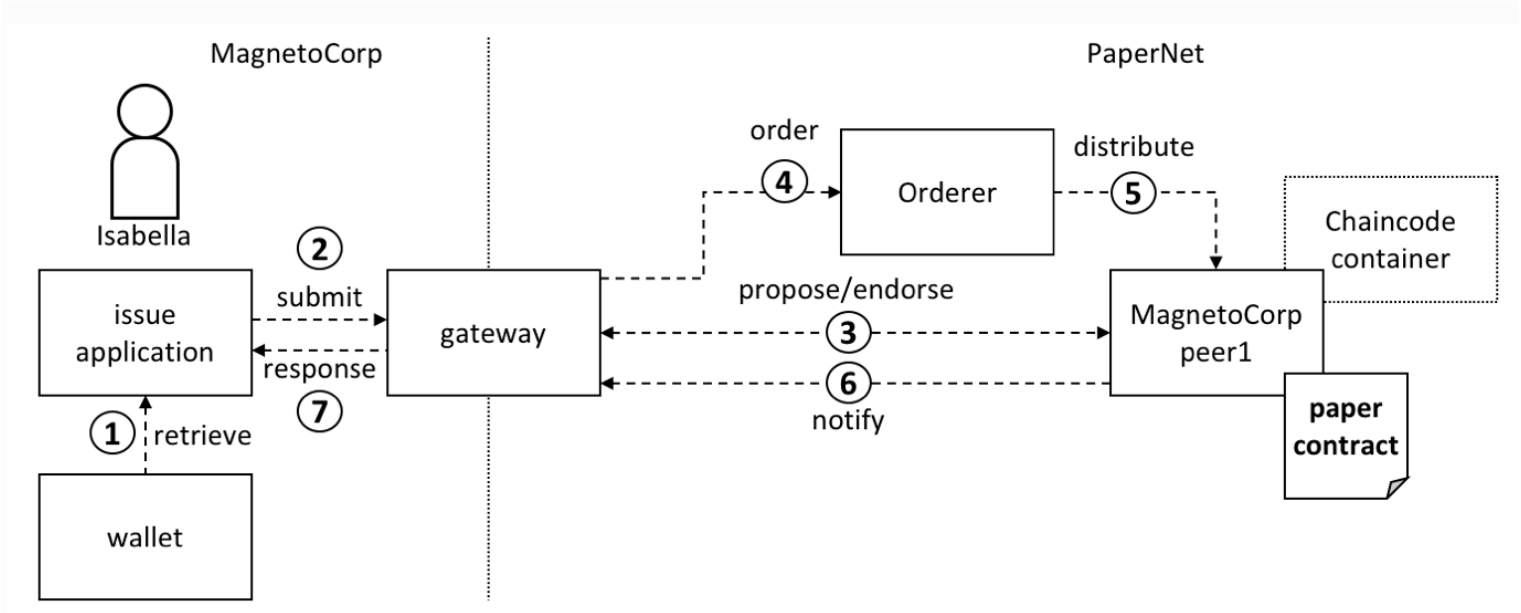
Publicar papercontract: El administrador de cualquier organización puede publicar el smart contract en el canal.

```
(DigiBank admin)$ peer lifecycle chaincode commit -o localhost:7050 -
-ordererTLSHostnameOverride orderer.example.com - -peerAddresses
localhost:7051 - -tlsRootCertFiles $PEER0_ORG1_CA - -peerAddresses
localhost:9051 - -tlsRootCertFiles $PEER0_ORG2_CA - -channelID
mychannel - -name papercontract -v 0 - -sequence 1 - -tls - -cafile
$ORDERER_CA - -waitForEvent
```

Índice

Diagrama de emisión

MagnetoCorp utiliza su aplicación cliente **issue.js** y emitir el papel comercial .



Emisión

En cd commercial-paper/organization/magnetocorp/application/ se pueden ver los archivos **addToWallet.js**, **issue.js** y **package.json**.

Isabella va a utilizar addToWallet.js para agregar su identidad al wallet (Certificado X.509) y luego issue.js utilizará esa identidad para crear el papel comercial a nombre de MagnetoCorp invocando al papercontract.

Emisión

La aplicación cliente **issue.js** está escrita en javascript y se ejecutará en node.js. Los paquetes **js-yaml** y **fabric-network** se deben descargar de npm.

Los paquetes y las versiones necesarias están declaradas en el archivo **package.json**.

```
(MagnetoCorp Isabella)$ npm install
```

Emisión

Por convención, los paquetes descargados por **npm** se guardan en la carpeta **/node_modules**, donde se ejecutó el comando.

```
(MagnetoCorp Isabella)$ ls
```

Emisión

Para que Isabella pueda ejecutar emitir el papel comercial **00001**, se deben agregar su credenciales X.509 a su wallet.

```
(MagnetoCorp Isabella)$ node addToWallet.js
(MagnetoCorp Isabella)$ ls ../identity/user/isabella/wallet/
```

Emisión

Ahora sí, Isabella puede utilizar **issue.js** para enviar la transacción que emitirá el papel comercial **00001** de parte de MagnetoCorp.

```
(Isabella)$ node issue.js
```

Índice

Compra

Para comprar el papel comercial que emitió MagnetoCorp, el usuario de DigiBank (Balaji) debe realizar una transacción de compra desde la aplicación de DigiBank.

Compra

En cd commercial-paper/organization/digibank/application/ se pueden ver los archivos **addToWallet.js**, **buy.js**, **package.json** y **redeem.js**

Balaji va a utilizar buy.js y redeem.js para comprar y, posteriormente, canjear el papel comercial que emitió MagnetoCorp.

Compra

Al igual que en MagnetoCorp, DigiBank debe instalar los paquetes declarados en **package.json** a través de npm.

```
(Balaji)$ npm install
```

Compra

Para que Balaji pueda comprar el papel comercial, debe agregar sus identidad a su wallet ejecutando el programa **addToWallet.js**.

```
(Balaji)$ node addToWallet.js
```

Compra

Finalmente, Balaji puede enviar la transacción **buy.js** para transferir (comprar) el papel comercial de MagnetoCorp a DigiBank.

```
(Balaji)$ node buy.js
```

Índice

Canjear

La transacción final para el papel comercial **00001** será que DigiBank lo intercambie con MagnetoCorp. Para ello Balaji utilizará el **redeem.js**.

```
(Balaji)$ node redeem.js
```

Índice

Limpiar

En cd fabric-samples/commercial-paper.

```
(Balaji)$ ./network-clean.sh
```

Este script dará de baja los peers, los contenedores, el nodo ordering service y el logspout. Además, eliminará las identidades de Isabella y Balaji.

Referencia

https://hyperledger-fabric.readthedocs.io/en/release-2.0/tutorial/commercial_paper.html