Hyperledger práctico

Fundamentos de blockchains

René Dávila - Jorge Solano

Índice

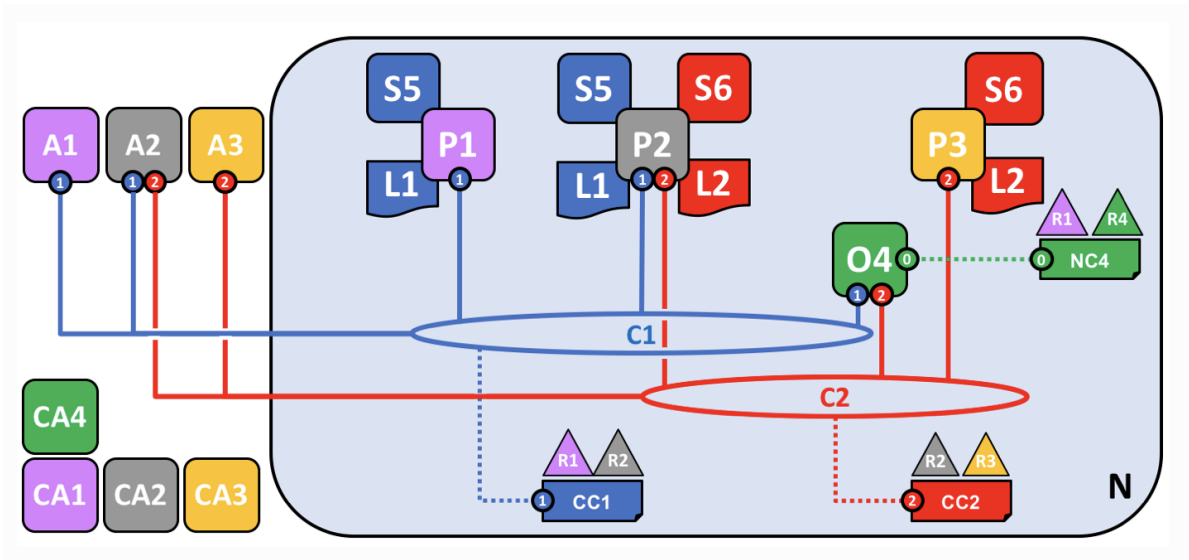
Para iniciar con la implementación de una **blockchain empresarial** (particular) se requiere tener un diagrama general de las entidades **participantes y su interacción**.

A continuación se describirán todos los **elementos** que pueden interactuar **en Hyperledger Fabric**, a partir de un ejemplo genérico.

Índice

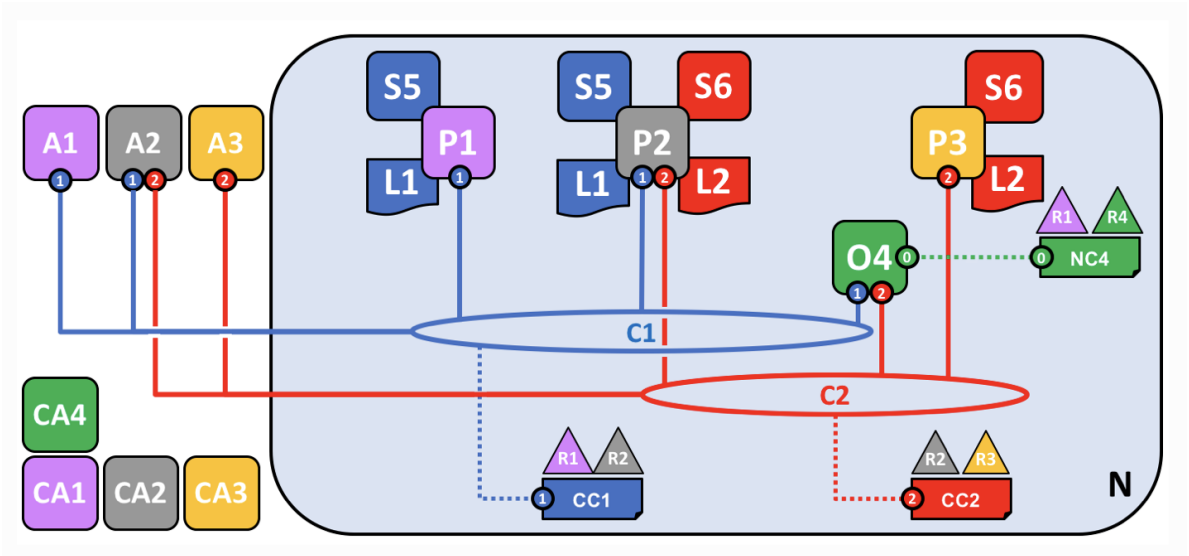
Contexto

Se tienen 4 organizaciones (R1, R2, R3 y R4). R4 es el iniciador de la red (no hará transacciones de negocios). R1 y R2 tienen una comunicación privada en la red. De la misma manera R2 y R3.



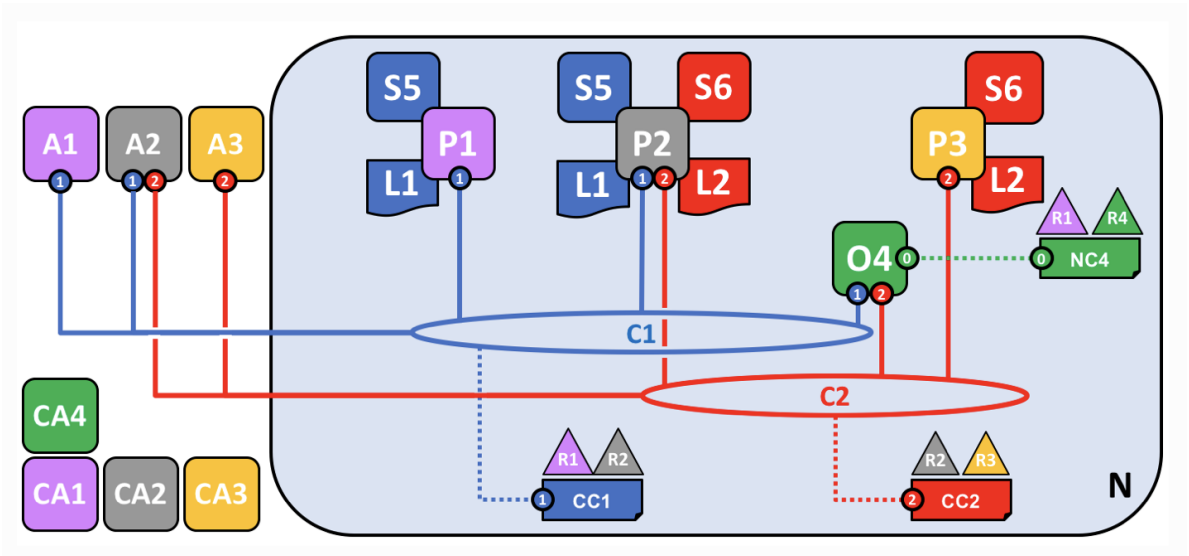
Contexto

R1 tiene una aplicación cliente A1 que hace transacciones en C1. R2 tiene una A2 que hace transacciones en C1 y en C2. R3 tiene una A3 que hace transacciones en el C2.



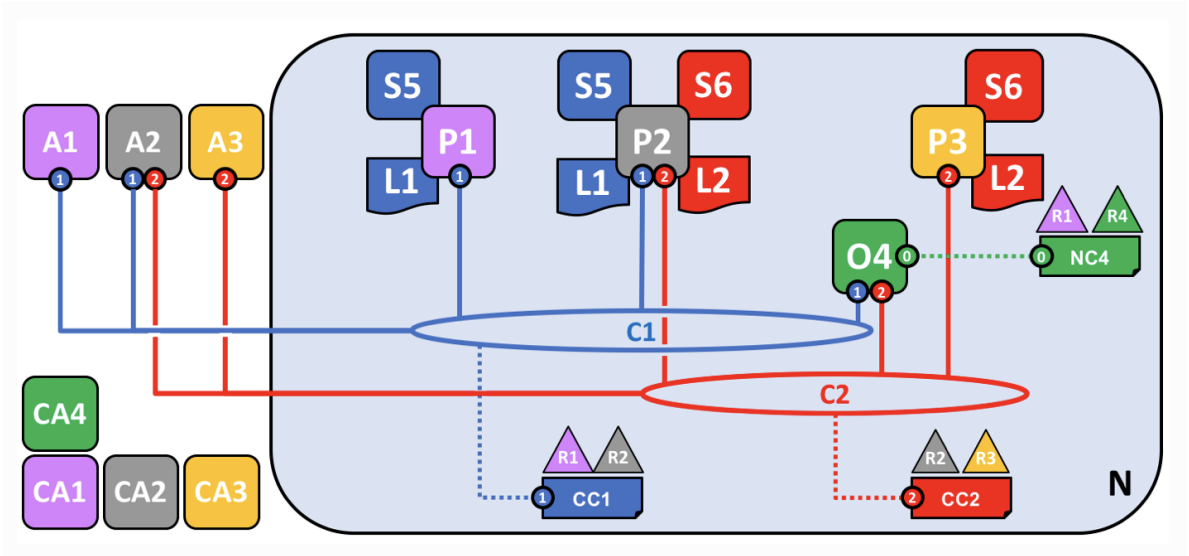
Contexto

El peer node P1 tiene una copia del ledger L1 asociado con C1 y un contrato inteligente S5. El peer node P2 tienen una copia de L1 y S5 asociado a C1 y una copia de L2 y S6 asociado a C2. El peer node P3 tiene una copia de L2 y S6 asociado a C2.



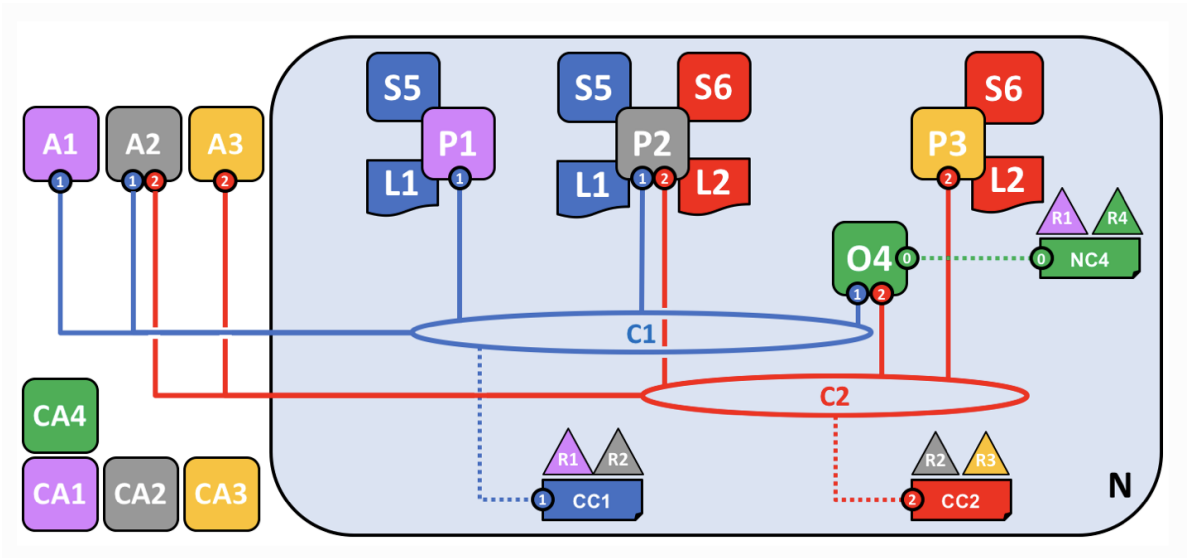
Contexto

La red está gobernada de acuerdo a las políticas especificadas en la configuración de red NC4. Las políticas están bajo el control de R1 y R4.



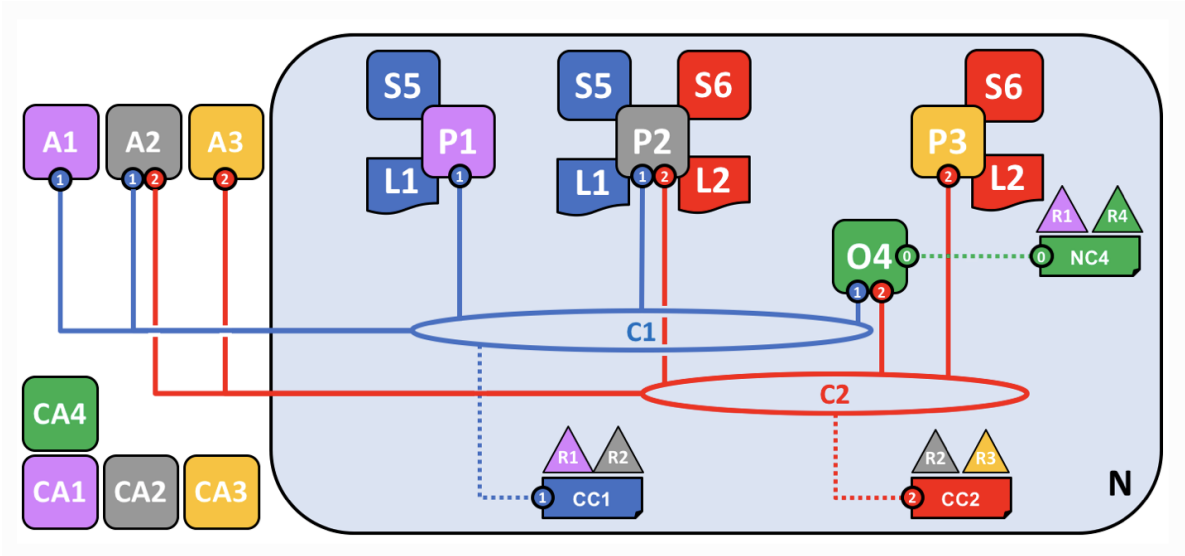
Contexto

C1 está gobernado de acuerdo a las políticas especificadas en la configuración del canal CC1; el canal está gobernado por R1 y R2. C2 está gobernado por las políticas especificadas en la CC2; el canal está gobernado por R2 y R3.



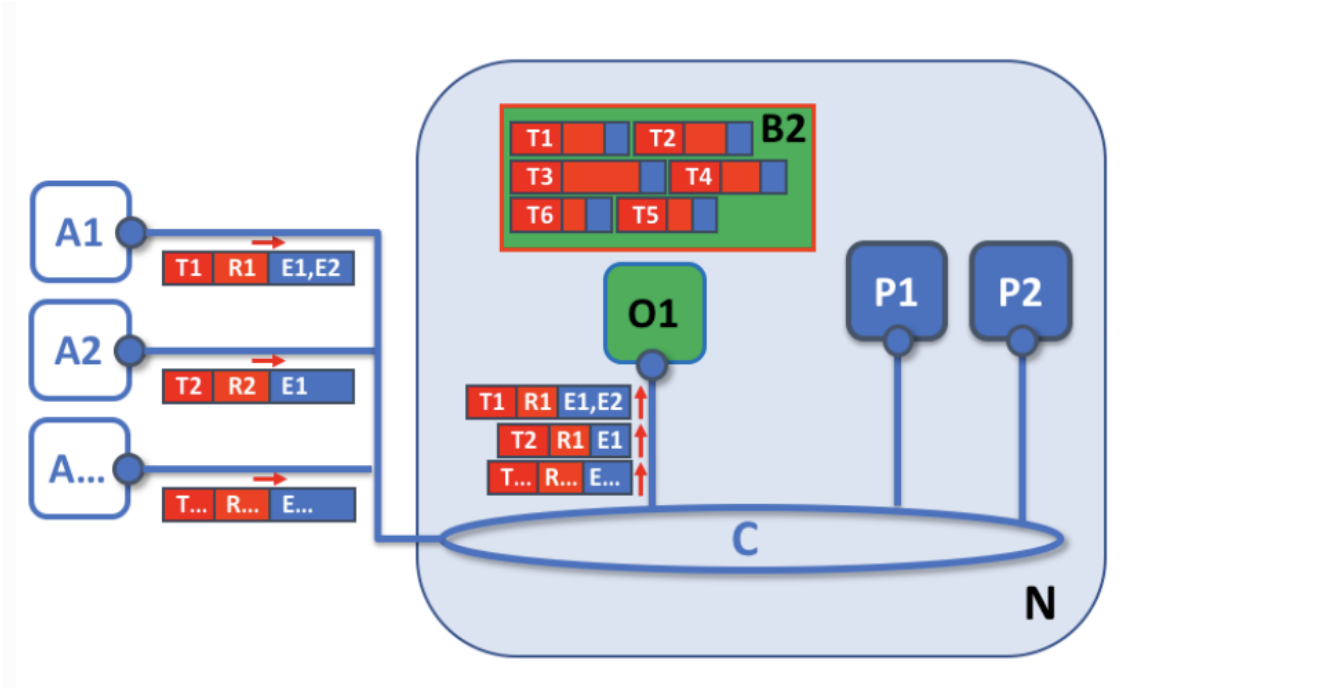
Contexto

Hay un nodo de ordering service O4 que funge como un punto de administración de la red N. O4 soporta los canales de aplicación C1 y C2, con el fin de ordenar los bloques de transacciones para su distrubución.



Contexto

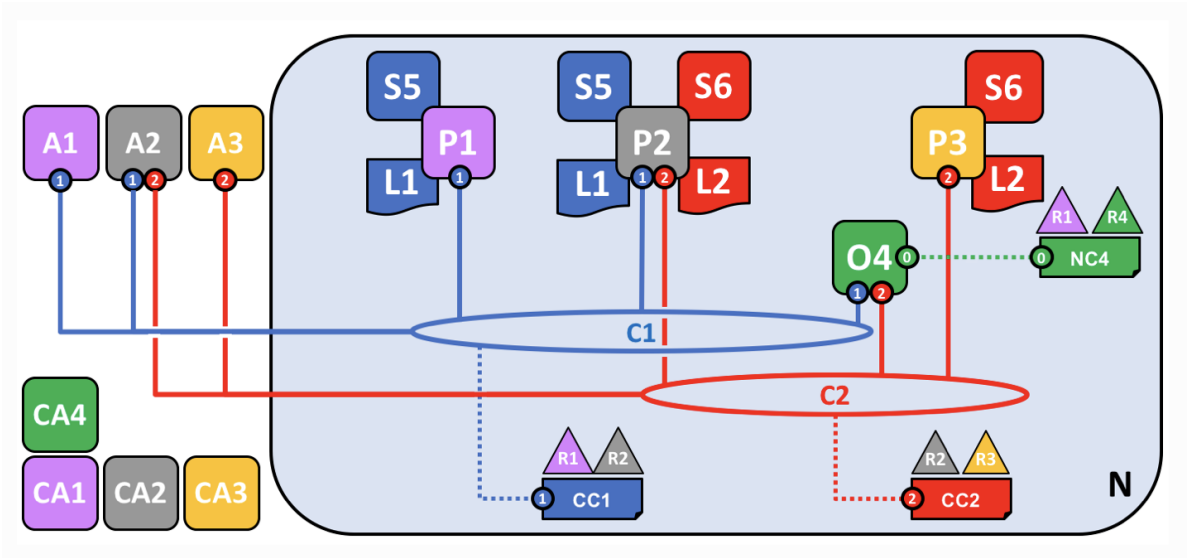
Ordering sevice



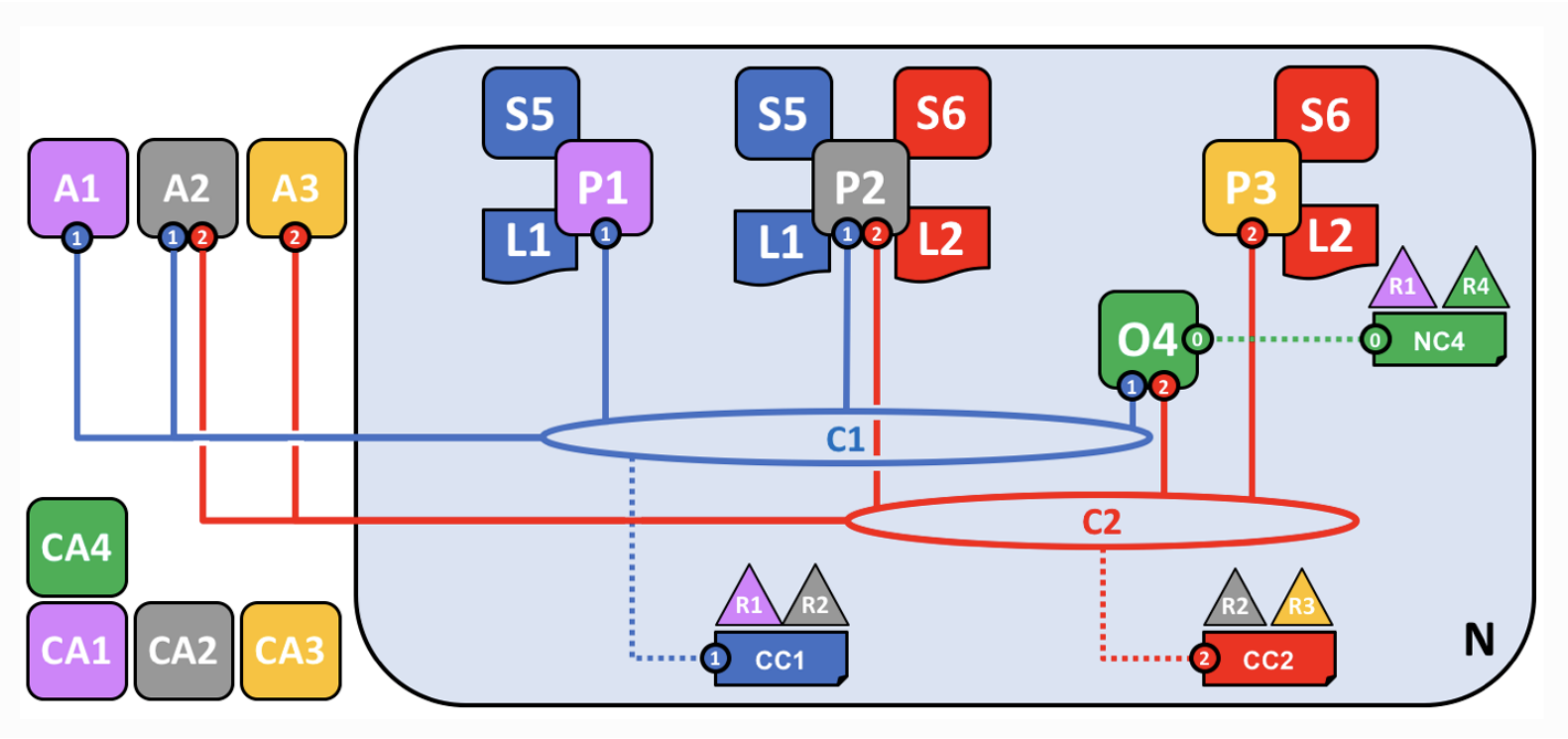
https://hyperledger-fabric.readthedocs.io/en/release-2.0/orderer/ordering_service.html

Contexto

Cada una de las organizaciones tiene su propia Autoridad Certificadora CA.



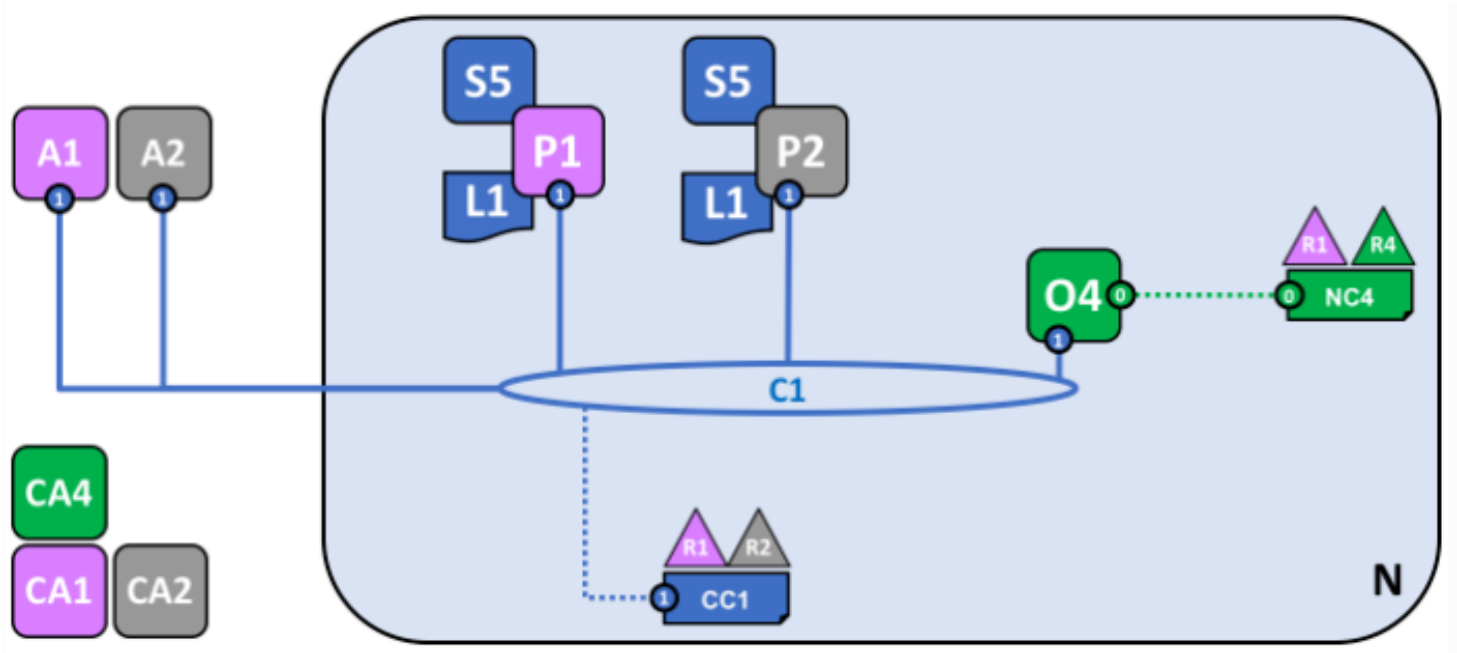
Contexto



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

Contexto

Análisis paso a paso previo a ejecución.



Índice

Ejecución

Una vez descargado Hyperledger Fabric ya es posible interactuar con alguno de los proyectos.

En realidad, se descargaron imágenes y ejemplos Docker, esto es, contenedores con aplicaciones precargadas, las cuales se guardaron en la carpeta **fabric-samples**.

Ejecución

Vamos a trabajar con el proyecto test-network que está dentro de la carpeta fabric-samples (cd fabric-samples/test-network).

En esa carpeta se encuentra el script *network.sh*, el cual levanta una red Fabric utilizando las imágenes Docker descargadas. Para ver el texto de ayuda del script se puede ejecutar *network.sh -h*

```
Usage:
  network.sh <Mode> [Flags]
    <Mode>
      - 'up' - bring up fabric orderer and peer nodes. No channel is created
      - 'up createChannel' - bring up fabric network with one channel
      - 'createChannel' - create and join a channel after the network is created
      - 'deployCC' - deploy the fabcar chaincode on the channel
      - 'down' - clear the network with docker-compose down
      - 'restart' - restart the network

    Flags:
    -ca <use CAs> -  create Certificate Authorities to generate the crypto material
    -c <channel name> - channel name to use (defaults to "mychannel")
    -s <dbtype> - the database backend to use: goleveldb (default) or couchdb
    -r <max retry> - CLI times out after certain number of attempts (defaults to 5)
    -d <delay> - delay duration in seconds (defaults to 3)
    -l <language> - the programming language of the chaincode to deploy: go (default), javascript, or java
    -v <version>  - chaincode version. Must be a round number, 1, 2, 3, etc
    -i <imagetag> - the tag to be used to launch the network (defaults to "latest")
    -verbose - verbose mode
  network.sh -h (print this message)

 Possible Mode and flags
  network.sh up -ca -c -r -d -s -i -verbose
  network.sh up createChannel -ca -c -r -d -s -i -verbose
  network.sh createChannel -c -r -d -verbose
  network.sh deployCC -l -v -r -d -verbose

 Taking all defaults:
        network.sh up

 Examples:
  network.sh up createChannel -ca -c mychannel -s couchdb -i 2.0.0
  network.sh createChannel -c channelName
  network.sh deployCC -l javascript
```

Crear la red

El siguiente comando levanta la red del proyecto **test-network**, la cual consiste en dos peer nodes y un nodo ordering service; todavía no se crea algún canal:

```
./network.sh up
```

Listar contenedores

En este momento se encuentran los contenedores Docker corriendo (los dos nodos y el nodo orden de servicio). El siguiente comando permite listar los contenedores en ejecución:

```
docker ps -a
```

Consorcio

test-network tiene un consorcio (grupo de organizaciones) con dos miembros Org1 y Org2, además de una organización que mantiene el orden de servicio (ordering service) en la red.

Los **peers nodes** son los componentes fundamentales en cualquier red Fabric. Ellos son los encargados de **almacenar** en el ledger de la **blockchain** y validar las transacciones antes de que se suban al ledger. Además, ellos **ejecutan los smart contracts** los cuales contienen la lógica del negocio.

Ordering service

Todas las redes Fabric deben incluir un nodo **ordering service**. Este nodo es el que decide el orden y las transacciones que se incluyen en un bloque.

Los **ordering nodes** hace uso de la configuración de la red (define las capacidades de la red) y los parámetros de configuración del canal. Por defecto, los **ordering nodes** utilizan el algoritmo de consenso **Raft (**`https://raft.github.io`**)**.

Canales

La red que se está ejecutando cuenta con 2 peer nodes, uno por cada organización (Org1 y Org2), y un ordering node. Ahora vamos a crear un **canal para las transacciones** entre el consorcio.

Recordemos que los canales son **enlaces privados de comunicación** entre miembros específicos de la red y cada canal tiene su propio ledger de la blockchain.

Crear canal

Para **crear un canal** entre las organizaciones Org1 y Org2 y unir los peer nodes al canal se debe ejecutar el siguiente comando:

```
./network.sh createChannel
```

Smart contract en Fabric

Para asegurar que una transacción creada a partir de un **smart contract** sea válida, ésta debe estar firmada por múltiples organizaciones.

En Fabric los smart contracts se despliegan en paquetes referidos como **chaincode**. El chaincode se **instala** en los peer nodes de la organización y después se **despliega** en el canal donde se ocupará. Los miembros del canal deben estar de acuerdo con su definición.

Iniciar chaincode

Para **crear un chaincode** en el canal desplegado se debe ejecutar el siguiente comando:

```
./network.sh deployCC
```

deployCC instala el chaincode de **fabcar** en peer0.org1.example.com y en peer0.org2.example.com y luego lo despliega en el canal.

Crear una aplicación cliente

En este momento se puede usar una **aplicación CLI** para **interactuar con la red** (ejecutar smart contracts, actualizar canales, instalar y desplegar smart contracts).

Crear una aplicación cliente

Para crear a **CLI** se va a ocupar la aplicación peer que está en la carpeta **bin** de fabric-samples (ya en el PATH). También se requiere poner en el PATH la variable FABRIC_CFG_PATH para apuntar al archivo de configuración **core.yaml**:

```
export FABRIC_CFG_PATH=$PWD/../config/
```

Crear una aplicación cliente

Después, se deben establecer las **variables de entorno** que permitiran operar a la aplicación CLI en la Org1:

Variables de entorno para Org1
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=$PWD/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=$PWD/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051

Interactuar con la red

Ahora, el nodo **CLI** puede **solicitar el ledger** del canal al que pertenece y hacer transacciones: **consultar todos los carros.**

```
peer chaincode query -C mychannel -n fabcar -c '{"Args":["queryAllCars"]}'
```

Interactuar con la red

El nodo **CLI** puede **solicitar el ledger** del canal al que pertenece y hacer transacciones: **cambiar el dueño de un auto.**

```
peer chaincode invoke -o localhost:7050 –ordererTLSHostnameOverride
orderer.example.com –tls true –cafile
$PWD/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
-C mychannel -n fabcar –peerAddresses localhost:7051 –tlsRootCertFiles
$PWD/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
–peerAddresses localhost:9051 –tlsRootCertFiles
$PWD/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
-c '{"function":"changeCarOwner","Args":["CAR9","Dave"]}'
```

Interactuar con la red

Debido a que la política de la red requiere que Org1 y Org2 aprueben la transacción, el **chaincode invoca** a ambos **peers** utilizando un **certificado TLS**.

El **chaincode** se instala en los **peer nodes** de la organización y **después se despliega** en el canal donde se ocupará. Los miembros del canal deben estar de acuerdo con su definición.

Validar cambios

Ahora, se pueden **validar los cambios** en el ledger de la blockchain. En esta ocasión se va a utilizar el nodo **CLI de Org2**, para ello hay que exportar las variables de entorno de Org2:

Variables de entorno para Org2
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=$PWD/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=$PWD/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=localhost:9051

Validar cambios

El nodo **CLI** puede **solicitar el ledger** del canal al que pertenece y hacer transacciones: **Consultar un carro**

```
peer chaincode query -C mychannel -n fabcar -c '{"Args":["queryAllCars"]}'
```

Dar de baja la red

Para **terminar** un proyecto se debe **dar de baja la red**. El comando **down** detiene y elimina los nodos, los contendores, las organizaciones, el material criptográfico, los chaincodes, los registros Docker, los canales y los volúmenes Docker previamente ejecutados.

./network.sh down

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

Índice

Logspout (monitordocker.sh)

Es una **bitácora** de cualquier evento en la red, esta herramienta **recolecta cualquier flujo de salida** de diferentes contenedores. El script está en commercial-paper/organization/digibank/configuration/cli/. Para **iniciar** el monitor:

```
./monitordocker.sh net_test
```

Para **detener** el monitor:

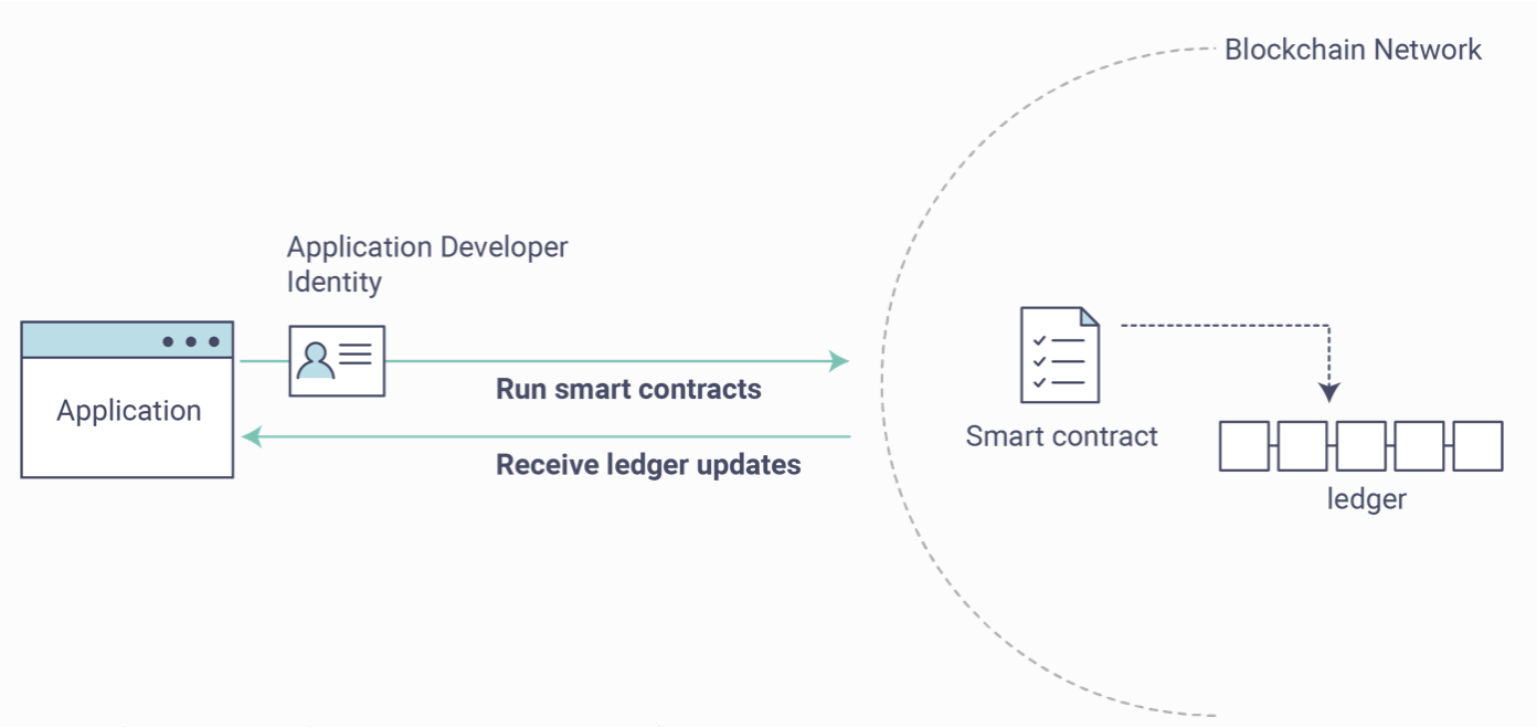
```
docker stop logspout
docker rm logspout
```

Referencia

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

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/
```

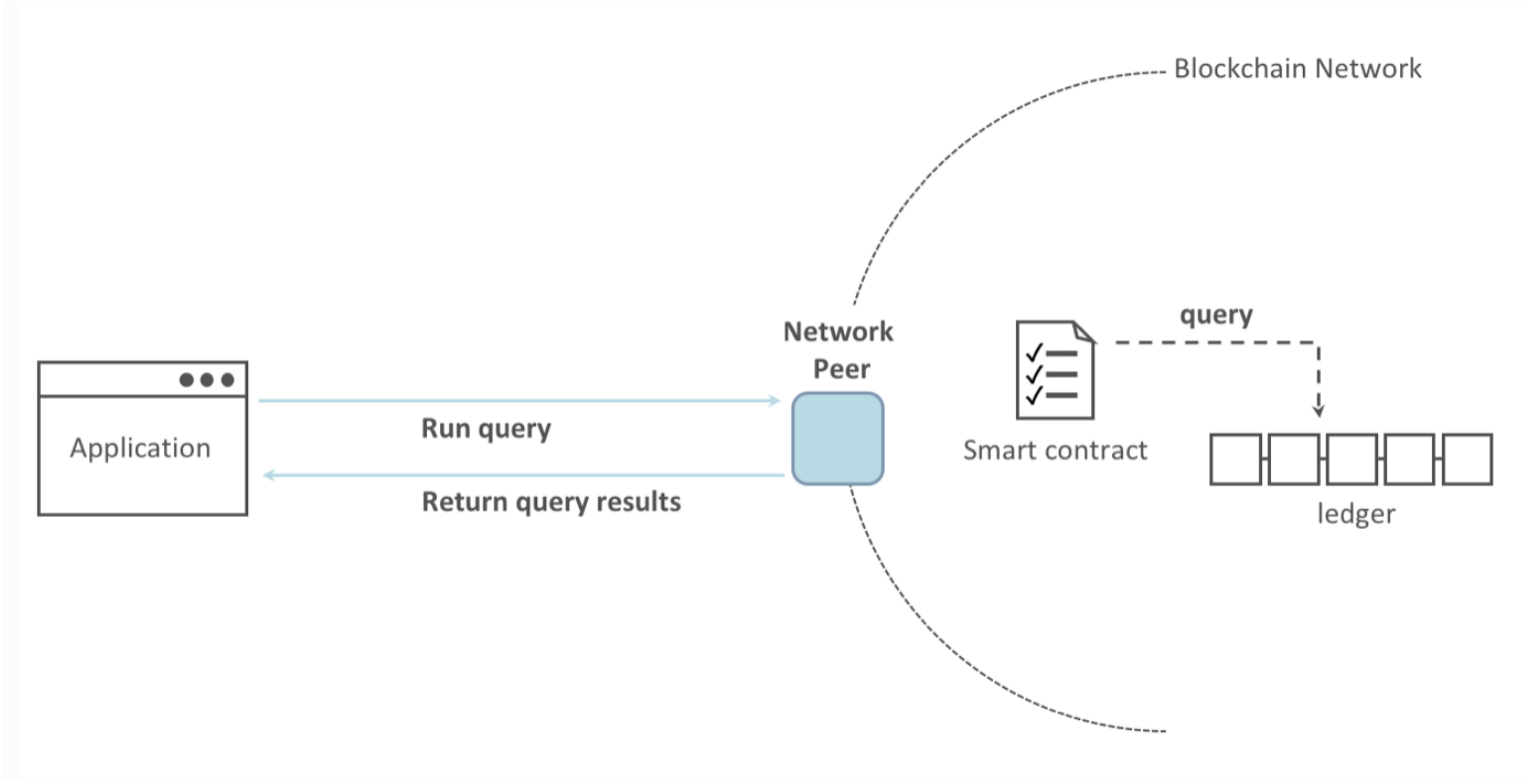
Usuario cliente

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

```
$ node registerUser.js
```

```
$ ls wallet/
```

Solicitar el ledger

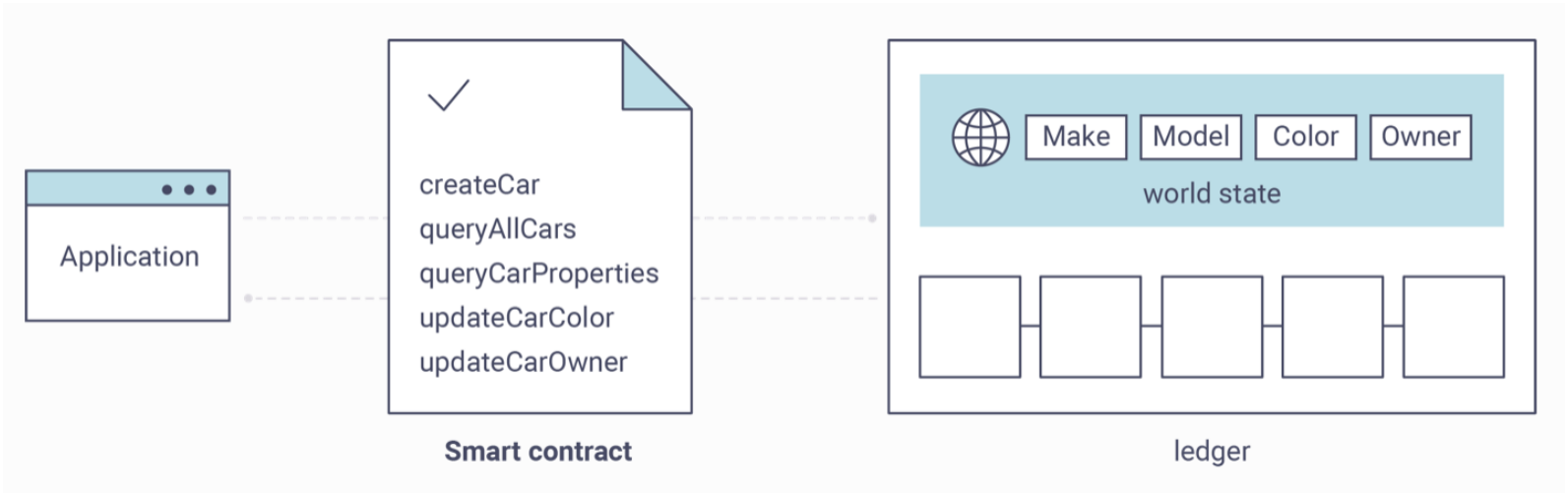


Los valores actuales del ledger están en el **world state**. El world state está representado como un conjunto llave-valor.

Aplicación cliente

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

```
$ node query.js
```

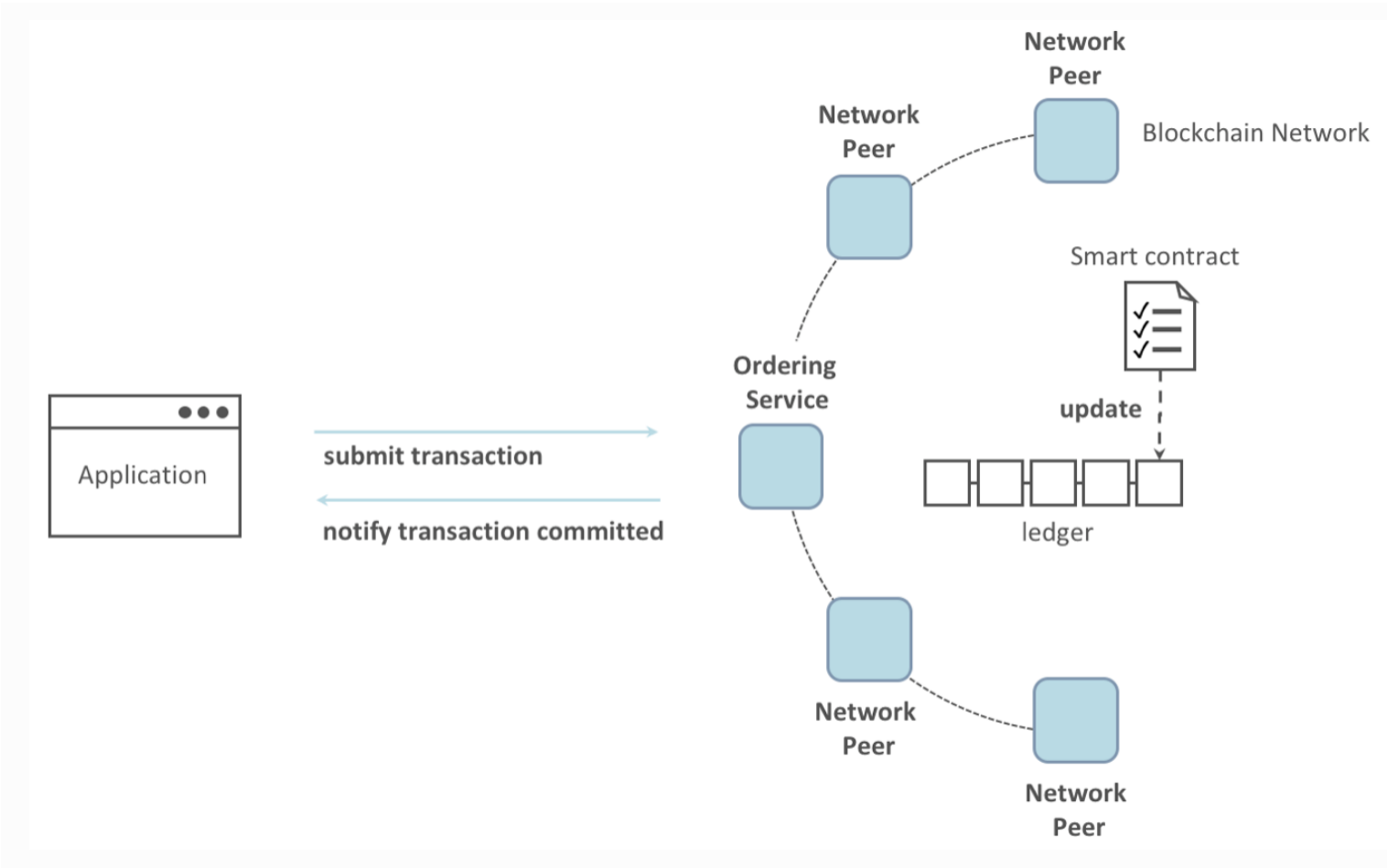


Contrato FabCar

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.

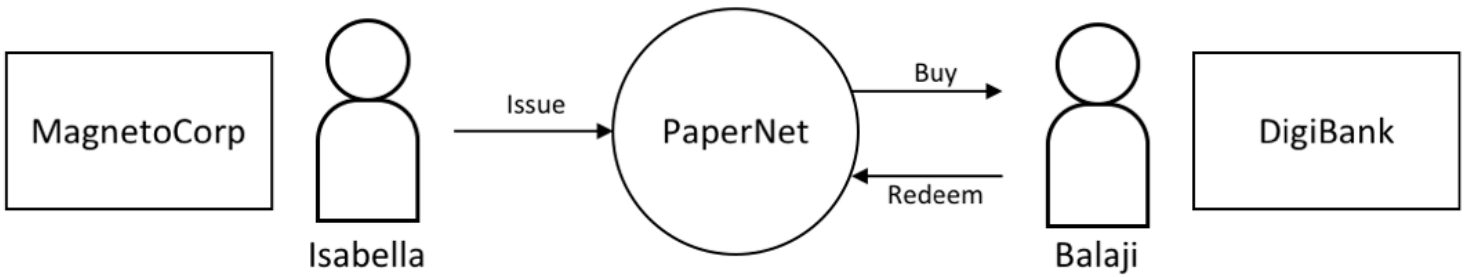
```
$ ./networkDown.sh
```

Referencia

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

Índice

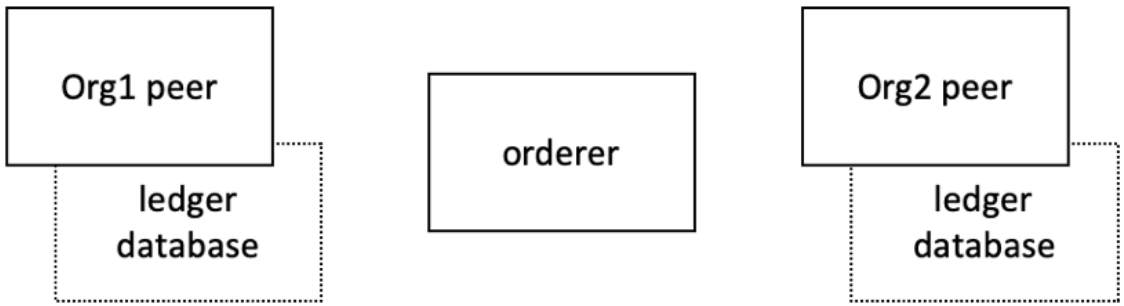
Diagrama de las organizaciones:



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

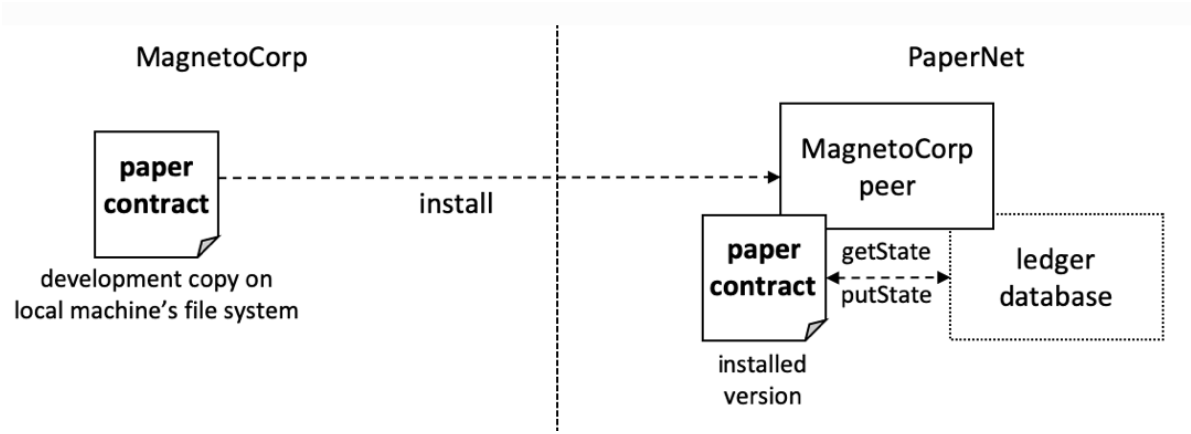
Creación del contrato

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

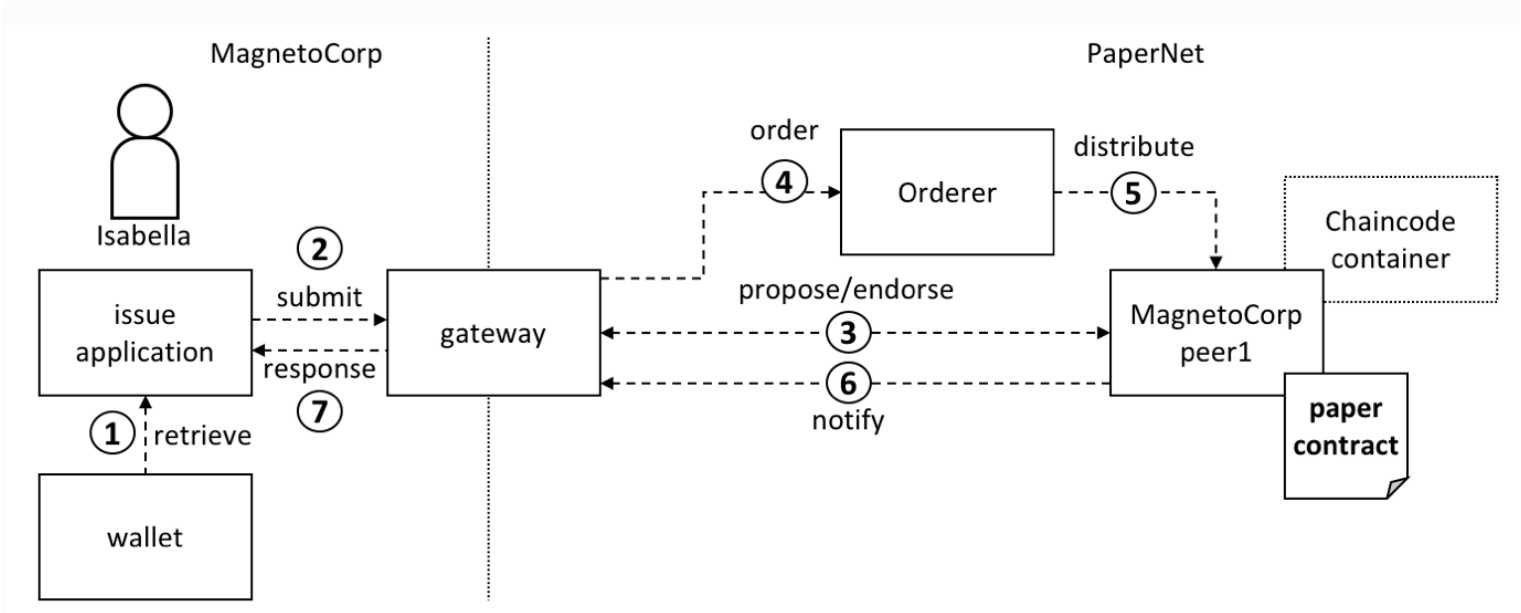
Publicar papercontract: El administrador de cualquier organización puede publicar el smart contract en el canal.

```
(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

Índice

Referencias I

[1] Raft consensus algorithm website.
The raft consensus algorithm.
URL https://raft.github.io.
[Visitada el 03-04-2020].

[2] Steven J. Vaughan-Nichols for Linux and Open Source.
What is docker and why is it so darn popular?
URL https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/.

[Visitada el 02-04-2020].

[3] Hyperledger.
Hyperledger fabric: A blockchain platform for the enterprise.
URL https://hyperledger-fabric.readthedocs.io/en/release-2.0/.
[Visitada el 29-02-2020].

Por su atención

Hyper **gracias**

René - Jorge