

Introducción al desarrollo Blockchain con Hyperledger Fabric por [Coding Bootcamps](https://coding-bootcamps.com) (coding-bootcamps.com)

Proyecto - Crear una DApp para cadena de suministro con Hyperledger Fabric

***Nota:** Este proyecto pertenece al libro “Hands-on Smart Contract Development with Hyperledger Fabric V2” escrito por Matt Zand, Brian Wu y Mark Anthony Morris por O’Reilly Media. A fecha Julio 2020, es el único libro de Hyperledger Fabric en el mercado que cubre Hyperledger Fabric V2 en profundidad. Se recomienda este libro para completar el curso.*

Este proyecto está organizado en 5 secciones:

1. [Diseño de la cadena de suministro en Blockchain](#)

- [Workflow de la cadena de suministro](#)
- [Definiendo el Consorcio](#)
- [Ciclo de vida PLN](#)
- [Atributos y valores del equipo](#)
- [Cambio del estado del equipo](#)
- [Equipo al alcance de la farmacia](#)
- [Transacciones](#)

2. [Escribir chaincode como smart contract](#)

- [Estructura del proyecto](#)
- [Clase Contract](#)
- [Lógica de transacción](#)
- [Consulta a la ledger](#)

3. [Compilando y desplegando chaincode en Fabric](#)

- [Prerequisitos](#)
- [Estructura del proyecto](#)
- [configtx](#)
- [Docker](#)
- [Cryptogen](#)
- [Instalar Binarios e imágenes de Docker](#)
- [Iniciar la red PLN](#)
- [Monitorizar la red PLN](#)

- [Creando un canal en la red PLN](#)
- 4. [Ejecutando y testeando el smart contract](#)
- 5. [Desarrollo de una aplicación con el SDK de Hyperledger Fabric](#)

En este proyecto, pondremos en práctica los conceptos teóricos y conceptuales de Hyperledger Fabric para diseñar y construir una simple cadena de suministro basada en blockchain (HF), llamada Pharma Ledger Network (PLN). Este proyecto proporcionará una visión global de como blockchain genera transparencia, ambiente de confianza, seguridad, etc en una red de negocio global. El proyecto PLN mostrará como blockchain puede ayudar a fabricantes, mayoristas y otros miembros de la cadena de suministro, como las farmacias, en la entrega de su material y equipo médico.

1. Diseño de la cadena de suministro en Blockchain

Generalmente, una cadena de suministro tradicional carece de transparencia e informes confiables. Las grandes organizaciones han construido sus propios sistemas para permitir el control global de sus operaciones diarias mientras registran las transacciones entre proveedores y distribuidores en tiempo real. Otras pequeñas empresas carecen de esos sistemas y tienen una visibilidad limitada. Esto genera que todo el flujo del proceso de productos de la cadena de suministro (desde la producción hasta el punto final de consumo) tenga transparencia muy limitada, generando informes erróneos y falta de interoperabilidad.

Por diseño, blockchain es un sistema descentralizado de contabilidad, transparente, inmutable y seguro. Se considera una buena solución para la industria de la cadena de suministro donde se registran, controla y transfieren activos. La gran adopción y popularidad de blockchain se debe a su aplicación y uso en cadenas de suministro.

Un smart contract define un conjunto de funciones comerciales que se implementan en blockchain, que puede ser accedido por múltiples partes en la red blockchain. Cada miembro de una blockchain tiene asignados identificadores únicos para firmar y verificar los bloques que se agregan a la blockchain. Durante el ciclo de vida de la cadena de suministro, cuando los miembros autorizados en una red de un consorcio invocan una función del smart contract, los datos del estado se actualizarán, después el estado de los activos actuales y los datos de la transacción se convertirán en un registro permanente en la ledger. Asimismo, los procesos relacionados con los activos se pueden mover fácil y rápidamente de un paso a otro. Todos los participantes de la cadena de suministro pueden rastrear, compartir y consultar las transacciones digitales en la ledger en tiempo real. Blockchain proporciona a las organizaciones nuevas oportunidades para corregir problemas dentro de su sistema de cadena de suministro, ya que gira en torno a una única fuente de verdad.

En esta sección, analizaremos un sistema de cadena de suministro simple y crearemos una red basada en blockchain (PLN) con su propia aplicación. Este proyecto, proporcionará una buena base sobre cómo analizar e implementar una aplicación basada en Hyperledger Fabric. Comenzaremos analizando el flujo de trabajo del proceso empresarial, identificando las

organizaciones en la red y luego diseñando la red del consorcio. También definiremos un smart contract que ejecutará cada organización.

Workflow de la cadena de suministro

Veamos las diferentes organizaciones que participan en nuestra red PLN, como se muestra en la “Figura 1”. Para fines prácticos se ha simplificado el proceso de la cadena de suministro, ya que puede ser mucho más complejo en un caso de uso del mundo real:

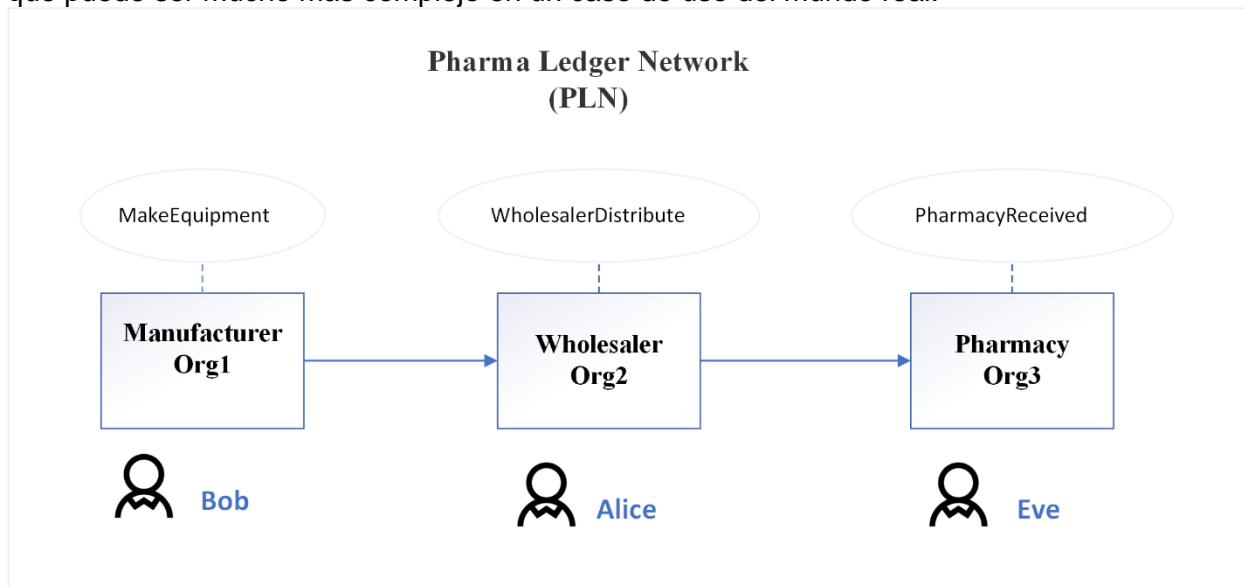


Figura 1. Organizaciones de PLN

El proceso en la red PLN se divide en los siguientes 3 pasos:

1. Un fabricante fabrica un equipo y lo envía al mayorista.
2. Un mayorista distribuye el equipo a la farmacia.
3. La farmacia, como consumidor, recibe el equipo y finaliza el flujo de trabajo (workflow) de la cadena de suministro.

Definiendo el Consorcio

Como podemos ver en el flujo de trabajo del proceso, nuestra red PLN involucra a tres organizaciones: fabricante, mayorista y farmacia. Estas tres entidades se unirán para construir un consorcio para llevar a cabo el negocio de la cadena de suministro. Los miembros del consorcio pueden crear usuarios, invocar smart contracts y consultar datos de la blockchain. La Tabla 1 muestra las organizaciones y usuarios del consorcio PLN.

Nombre de la Org.	Usuario	MSP	peer
Manufacturer	Bob	Org1MSP	peer0.org1.example.com
Wholesaler	Alice	Org2MSP	peer0.org2.example.com
Pharmacy	Eve	Org3MSP	peer0.org3.example.com

Tabla 1: Organizaciones y usuarios en el consorcio PLN

En nuestro consorcio PLN, cada una de las tres organizaciones tiene un usuario, un MSP y un peer. Para la organización del fabricante (manufacturer), tenemos a Bob como usuario de la aplicación, Org1MSP es un ID de MSP para cargar la definición de MSP. Definimos AnchorPeers con el nombre de host peer0.org1.example.com para la comunicación gossip. De manera similar, mayorista (wholesaler) es la segunda organización, Alice es su usuario de la aplicación y su ID de MSP es Org2MSP. Eve es el usuario de la farmacia (pharmacy) con Org3MSP.

Con las organizaciones identificadas, podemos definir nuestra red de Hyperledger Fabric como se muestra en la figura 2.

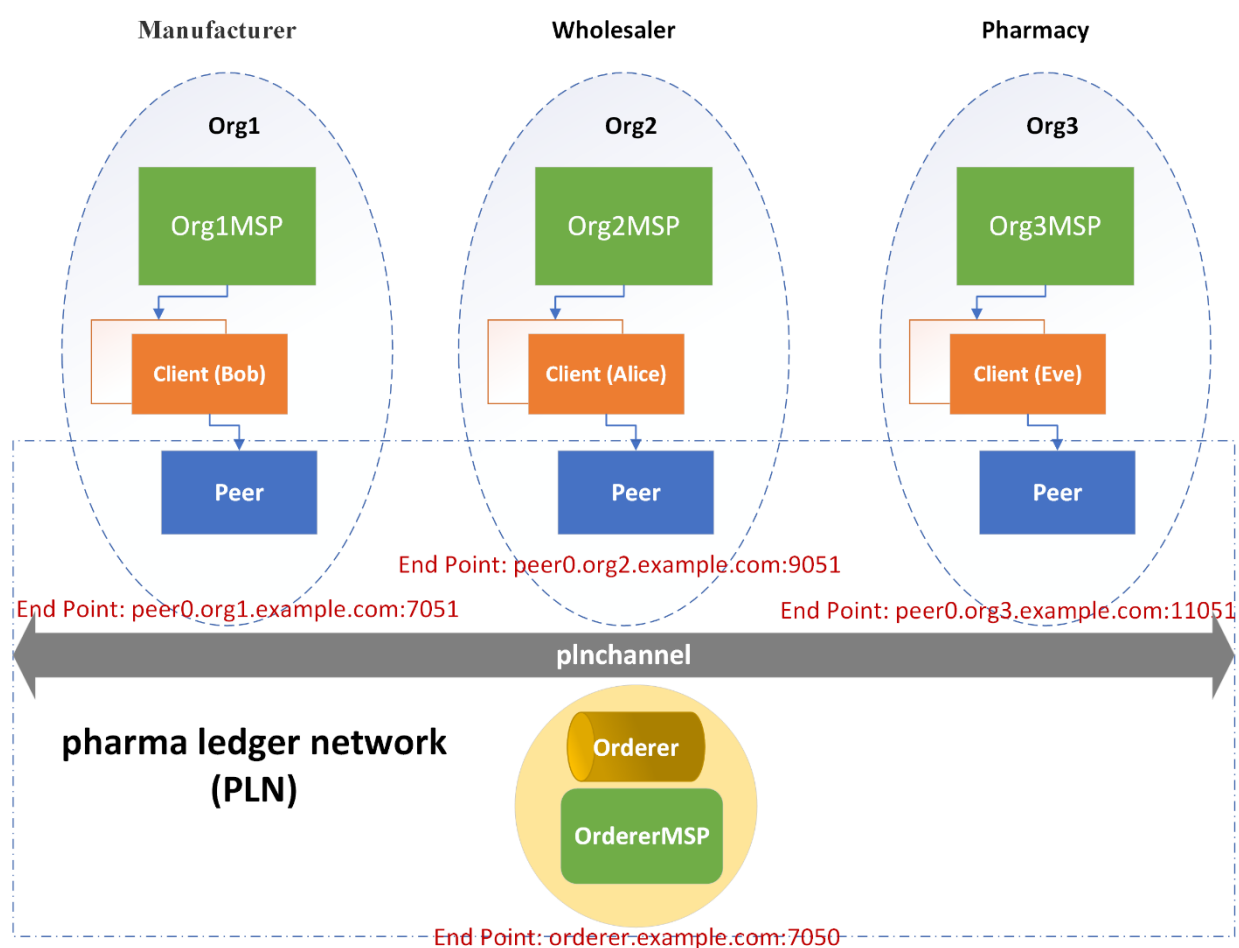


Figura 2. Red de Hyperledger Fabric

Dado que la instalación e implementación de PLN en múltiples nodos físicos puede no estar dentro del alcance de este proyecto, definimos un peer para cada una de las cuatro organizaciones, que representan el nodo del fabricante, mayorista, farmacia y orderer.

El canal plnchannel proporciona un mecanismo de comunicación privada que será utilizado por el orderer y las otras tres organizaciones para validar y ejecutar transacciones.

Ciclo de vida PLN

El 1 de enero, un fabricante fabricó una pieza con un id de identificación 2000.001, con sus atributos y valores como se muestran en la siguiente Figura 3.

```
equipmentNumber: 2000.001  
manufacturer: GlobalEquipmentCorp  
equipmentName: e360-Ventilator  
ownerName: GlobalEquipmentCorp  
previousOwnerType: MANUFACTURER,  
currentOwnerType: MANUFACTURER,  
createDateTime: Jan 1, 2021,  
lastUpdated: Jan 1, 2021, 10:01:02
```

Atributos y valores del equipo

Cada equipo tiene un número de identificación único que representa el equipo. Cada equipo tiene un propietario durante un cierto período de tiempo. En nuestro caso, tenemos tres tipos diferentes de propietarios: fabricante, mayorista y farmacia.

Cuando un fabricante fabrica un equipo y lo registra en el PLN, el resultado de la transacción muestra el equipo con un número de identificación único de 2000.001 en la ledger. El propietario actual es GlobalEquipmentCorp. El tipo de propietario actual y el anterior son el mismo: fabricante. El equipo se fabricó el 1 de enero de 2021. La última fecha de actualización es cuando se registró la transacción en PLN.

Después de un cierto tiempo, el fabricante envía el equipo al mayorista, el estado del equipo cambiará, incluida la propiedad, el tipo de propietario anterior y actual, y la última fecha de actualización. A continuación, echemos un vistazo a los estados del equipo que cambian como se muestra en la figura 4.

```
equipmentNumber: 2000.001  
manufacturer: GlobalEquipmentCorp  
equipmentName: e360-Ventilator  
ownerName: GlobalWholesalerCorp  
previousOwnerType: MANUFACTURER,  
currentOwnerType: WHOLESALER,
```

Cambio del estado del equipo

Uno de los cambios más significativos es que el equipo ahora es propiedad de GlobalWholesalerCorp. El tipo del anterior propietario es fabricante. La última fecha de actualización también ha cambiado.

Después de un mes, finalmente, la farmacia recibe el equipo. La propiedad ahora se transfiere del mayorista a la farmacia, como se muestra en la figura 5, a continuación. El flujo de la cadena de suministro puede considerarse cerrado.

```
equipmentNumber: 2000.001
manufacturer: GlobalEquipmentCorp
equipmentName: e360-Ventilator
ownerName: PharmacyCorp
previousOwnerType: WHOLESALER,
currentOwnerType: PHARMACY,
createDateTime: Jan 1, 2021,
lastUpdated: Feb 25, 2021, 11:01:08
```

Equipo al alcance de la farmacia

Gracias al número de identificación del equipo, los diferentes peers de las organizaciones pueden rastrear el historial completo de las transacciones del equipo.

Transacciones

Hemos visto que el flujo de trabajo de la red PLN tiene 3 pasos: El equipo es creado por el fabricante, se envía al mayorista y del mayorista a la farmacia. En conclusión, el fabricante fabrica el equipo, el mayorista lo distribuye y la farmacia recibe el equipo.

Una vez analizado el flujo de trabajo, podemos comenzar a escribir nuestro smart contract para PLN.

2. Escribir chaincode como smart contract

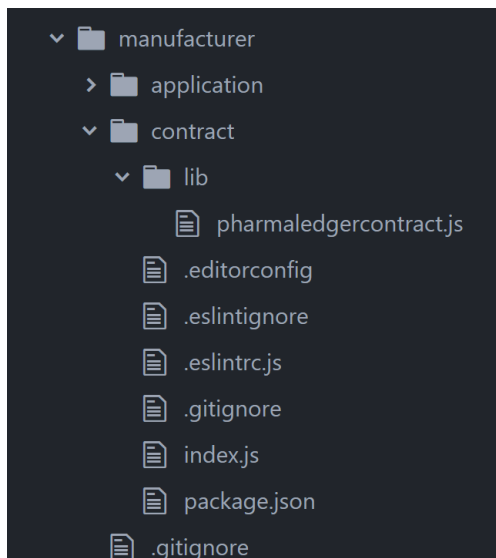
Anteriormente, hemos analizado como el estado y los atributos del equipo cambian durante todo el ciclo de vida dentro de la cadena de suministro de nuestra red PLN.

Un smart contract es un programa que implementa la lógica empresarial y gestiona el estado de un objeto empresarial durante el ciclo de vida. Durante la implementación, se empaquetará en el chaincode y se instalará en cada peer que se ejecuta en un contenedor Docker. El smart contract de Hyperledger Fabric se puede programar en Go, JavaScript, Java y Python. En esta sección, escribiremos el smart contract para nuestra red PLN usando JavaScript. Toda la documentación y código utilizado para este proyecto se pueden encontrar en la página del curso. También usaremos Fabric v2.1.0 y Fabric CA v1.4.7 durante todo el proyecto.

Estructura del proyecto

Para comenzar el desarrollo de nuestro smart contract, primero debemos crear nuestro proyecto. Dado que tenemos tres organizaciones, todos los peers deben estar de acuerdo y aprobar la nueva versión del contrato inteligente que se instalará y se implementará en la red. Para nuestra PLN, asumiremos que son todos iguales.

Definimos el smart contract llamado: `pharmalledgercontract.js`. Podemos ver en la figura 6 la estructura del proyecto.



En el archivo `package.json` se definen dos de las librerías más importantes de Fabric:

```
"dependencies": {  
  "fabric-contract-api": "^2.1.2",  
  "fabric-shim": "^2.1.2"  
},
```

fabric-contract-api proporciona la interfaz del contrato. Tiene dos clases importantes que todo smart contract tiene que utilizar - Contract y Context.

```
const { Contract, Context } = require('fabric-contract-api');
```

Contract tienen funciones “beforeTransaction”, “afterTransaction”, “unknownTransaction” y “createContext” que son opcionales. El nombre de la clase del contrato se puede especificar utilizando su superclase.

La clase Context proporciona el contexto de la transacción para cada invocación de transacción. Se puede anular para que el comportamiento adicional de la aplicación sea compatible con la ejecución del smart contract.

Clase Contract

La implementación de nuestro smart contract en la red PLN se extenderá de la clase de contrato incorporado en la librería de fabric-contract-api. Primero definamos PharmaLedgerContract con un constructor. org.pln.PharmaLedgerContract da un nombre muy descriptivo y único para nuestro contrato. Un nombre de contrato único es importante para evitar conflictos cuando hay muchos contratos de diferentes usuarios y operaciones en un sistema compartido.

```
const { Contract, Context } = require('fabric-contract-api');  
class PharmaLedgerContract extends Contract {  
  
  constructor() {  
    super('org.pln.PharmaLedgerContract');  
  }  
}
```

Lógica de transacción

Como ya sabemos, PharmaLedgerContract necesita 3 funciones comerciales para modificar el propietario del equipo del fabricante al mayorista y del mayorista a la farmacia.


```

async makeEquipment(ctx, manufacturer, equipmentNumber, equipmentName,
ownerName) {
  // makeEquipment logic
}
async wholesalerDistribute(ctx, equipmentNumber, ownerName) {
  // wholesalerDistribute logic
}
async pharmacyReceived(ctx, equipmentNumber, ownerName) {
  // pharmacyReceived logic
}

```

El fabricante se inicializará y creará la entrada de un nuevo equipo. Estas funciones aceptan como primer parámetro predeterminado un contexto. Cuando se llama a la función `makeEquipment`, la función espera cuatro atributos que los asignará al nuevo equipo.

```

async makeEquipment(ctx, manufacturer, equipmentNumber, equipmentName,
ownerName){
  let dt = new Date().toString();
  const equipment = {
    equipmentNumber,
    manufacturer,
    equipmentName,
    ownerName,
    previousOwnerType: 'MANUFACTURER',
    currentOwnerType: 'MANUFACTURER',
    createDateTime: dt,
    lastUpdated: dt
  };
  await ctx.stub.putState(equipmentNumber, Buffer.from(JSON.stringify(equipment)));
}

```

La última línea de código en la función `makeEquipment`, `ctx.stub.putState` almacenará en la ledger la información del equipo con el número de identificación del equipo. Los datos en formato JSON se convertirán en cadena y seguidamente en búfer. La API shim, requiere este formato de datos para comunicarse con el peer.

Utilizamos la función de JavaScript `new Date()` para obtener la fecha y hora actual y asignarla a los atributos del equipo que la necesiten como `lastUpdated`. Una vez los datos se han enviado, cada peer ha de validarlos y confirmar la transacción.

Una vez el fabricante ha creado el registro del equipo, el mayorista y la farmacia solo necesitan actualizar la propiedad. Ambas funciones son muy similares.

```
async wholesalerDistribute(ctx, equipmentNumber, ownerName) {
  const equipmentAsBytes = await ctx.stub.getState(equipmentNumber);
  if (!equipmentAsBytes || equipmentAsBytes.length === 0) {
    throw new Error(`${equipmentNumber} does not exist`);
  }
  let dt = new Date().toString();
  const strValue = Buffer.from(equipmentAsBytes).toString('utf8');
  let record;
  try {
    record = JSON.parse(strValue);
    if(record.currentOwnerType !== 'MANUFACTURER') {
      throw new Error(`equipment - ${equipmentNumber} owner must be MANUFACTURER`);
    }
    record.previousOwnerType = record.currentOwnerType;
    record.currentOwnerType = 'WHOLESALE';
    record.ownerName = ownerName;
    record.lastUpdated = dt;
  } catch (err) {
    throw new Error(`equipment ${equipmentNumber} data can't be processed`);
  }
  await ctx.stub.putState(equipmentNumber, Buffer.from(JSON.stringify(record)));
}
```

En la función `wholesalerDistribute`, consultamos los datos actuales de un equipo en la ledger llamando a `ctx.stub.getState (equipmentNumber)`. Una vez que los datos han sido devueltos, debemos asegurarnos de que `equipmentAsBytes` no esté vacío y que `equipmentNumber` sea un número válido. Dado que los datos en la ledger tienen un formato JSON string byte, es necesario convertir los datos codificados a un formato JSON legible mediante `Buffer.from().toString('utf8')`. Seguidamente verificamos si el tipo de propietario actual del equipo es el fabricante utilizando los datos devueltos. Una vez que se cumplen todas estas condiciones, se vuelve a llamar a `ctx.stub.putState`. El estado del propietario del equipo se actualizará al mayorista con la fecha actual. Todo el registro de transacciones inmutables de todo el histórico de cambios de estado se almacenará permanentemente en la ledger. Definiremos la función `queryHistoryByKey` para consultar todos estos datos en el siguiente paso.

La función `pharmacyReceived` es muy similar a la función `wholesalerDistribute`. Necesita validar que el propietario actual del equipo es el “wholesaler” y seguidamente transferir y actualizar el propietario del equipo a la farmacia, para acabar actualizando los datos en la ledger.

```

    if(record.currentOwnerType!=='WHOLESALE') {
        throw new Error(` equipment - ${equipmentNumber} owner must be
        WHOLESALE`);
    }
    record.previousOwnerType= record.currentOwnerType;
    record.currentOwnerType = 'PHARMACY';

```

Consulta a la ledger

Después de implementar las tres funciones básicas de nuestro negocio, crearemos una función de consulta para buscar los datos de los equipos y una función para consultar todo el historial de registros de las transacciones.

ChaincodeStub es implementado por la biblioteca fabric-shim y proporciona las funciones GetState, GetHistoryForKey. En nuestro caso, para definir una función de consulta de datos de un equipo solo necesitamos llamar a ctx.stub.getState para obtener el resultado correspondiente.

La función GetHistoryForKey devuelve todo el histórico de valores a lo largo del tiempo para un equipo. Podemos iterar a través de estos registros y convertirlos en bytes JSON y enviar los datos como respuesta. Los datos de fecha y hora nos dicen cuándo se actualizó el estado del equipo. Cada registro contiene un ID de transacción relacionado y un sello de tiempo.

```

async queryHistoryByKey(ctx, key) {
    let iterator = await ctx.stub.getHistoryForKey(key);
    let result = [];
    let res = await iterator.next();
    while (!res.done) {
        if (res.value) {
            const obj = JSON.parse(res.value.value.toString('utf8'));
            result.push(obj);
        }
        res = await iterator.next();
    }
    await iterator.close();
    console.info(result);
    return JSON.stringify(result);
}

```

Estas son todas las funciones que implementaremos en el smart contract de nuestra red PLN. Seguidamente, veremos cómo compilar y desplegar el chaincode en Fabric.

3. Compilando y desplegando chaincode en Fabric

Ahora ya tenemos nuestro smart contract escrito utilizando JavaScript. Antes de implementar nuestro contrato, primero debemos configurar la red Fabric.

Para comenzar con Hyperledger Fabric, primero debemos cumplir algunos requisitos previos. Suponemos que ya los ha instalado, si aún no lo ha hecho, instálelos primero.

Prerequisitos

Antes de seguir, necesitamos instalar las siguientes herramientas:

- Linux (Ubuntu)
- Git (<https://git-scm.com>)
- cURL (<https://curl.haxx.se/>)
- Docker y Docker Compose: Docker versión 17.06.2-ce o superior.
- Go Language: Go versión 1.14.x
- Node.js y NPM: Node.js versión 8 es soportada (desde 8.9.4 y superior). Node.js version 10 es soportada (desde 10.15.3 y superior).

Para configurar una red de HF crearemos una organización utilizando la herramienta de Fabric cryptogen, seguidamente crearemos un consorcio y desplegaremos nuestra red PLN con Docker-compose. Veamos la estructura de nuestro proyecto.

Estructura del proyecto

Hemos definido todos los scripts de instalación y archivos de configuración para nuestro proyecto PLN. El código del proyecto se puede encontrar en la página del curso. La estructura del proyecto se muestra en la figura 7.

```
Project
  -- pharma-ledger-network
    -- configtx (configtx.yaml)
    -- docker (docker-compose-pln-net.yaml)
    -- organizations
      -- manufacturer, -- pharmacy, -- wholesaler,
```

Figura 7. Estructura del proyecto

Revisemos configuraciones importantes:

configtx

El archivo configtx.yaml en la carpeta Configtx define las diferentes organizaciones de nuestra red, que se utilizarán para crear el consorcio. La herramienta configtxgen de Fabric, generará el bloque génesis de configuración del canal y los archivos relacionados, utilizando la configuración en el archivo configtx.yaml. En la sección de organizaciones definimos OrdererOrg y tres organizaciones: org1, org2 y org3; que representan el fabricante, mayorista y farmacia, respectivamente. Cada organización tiene definido su nombre, ID, MSPDir y AnchorPeers. En MSPDir se indica el directorio MSP del material generado por cryptogen. AnchorPeers se especifica el host y el puerto de cada peer para cada organización.

```
Organizations:
- &OrdererOrg
  Name: OrdererOrg
  ID: OrdererMSP
  MSPDir: ../organizations/ordererOrganizations/example.com/msp
  Policies:
    ....
  OrdererEndpoints:
    - orderer.example.com:7050
- &Org1
  Name: Org1MSP
  ID: Org1MSP
  MSPDir: ../organizations/peerOrganizations/org1.example.com/msp
  Policies:
    ...
  AnchorPeers:
    - Host: peer0.org1.example.com
      Port: 7051
- &Org2
  AnchorPeers:
    - Host: peer0.org2.example.com
      Port: 9051
- &Org3
  AnchorPeers:
    - Host: peer0.org3.example.com
      Port: 11051
```

La sección de políticas de la organización define quien deberá aprobar los recursos de la organización. En la red PLN, utilizaremos políticas de "signature" (firma). Por ejemplo,

definimos las políticas de lectura para la org2, que permite al admin, peer y client de la org2 acceder a ese recurso en el nodo y solo permite que los peers respalden las transacciones. Se pueden definir las políticas según las necesidades de la aplicación y la red.

```
Policies:
  Readers:
    Type: Signature
    Rule: "OR('Org2MSP.admin', 'Org2MSP.peer', 'Org2MSP.client')"
  Endorsement:
    Type: Signature
    Rule: "OR('Org2MSP.peer')"
```

La sección Profile, define como se generará la red, incluyendo configuraciones y organizaciones en el consorcio PLN.

```
Profiles:
  PharmaLedgerOrdererGenesis:
    <<: *ChannelDefaults
    Orderer:
      <<: *OrdererDefaults
      Organizations:
        - *OrdererOrg
      Capabilities:
        <<: *OrdererCapabilities
    Consortiums:
      PharmaLedgerConsortium:
        Organizations:
          - *Org1
          - *Org2
          - *Org3
  PharmaLedgerChannel:
    Consortium: PharmaLedgerConsortium
    <<: *ChannelDefaults
    Application:
      <<: *ApplicationDefaults
      Organizations:
        - *Org1
        - *Org2
        - *Org3
      Capabilities:
        <<: *ApplicationCapabilities
```

Docker

La carpeta docker contiene el archivo de configuración de docker-compose: docker-compose-pln-net.yaml. Docker Compose utiliza este archivo de configuración para iniciar el entorno de ejecución de Fabric. Se definen volúmenes, redes y servicios. En nuestro proyecto PLN, definimos el nombre de nuestra red como pln. Especificamos las variables de entorno para cada servicio de la organización. El contenedor utilizará las imágenes de hyperledger/fabric-peer. En la configuración de los volúmenes se asigna los directorios donde se utilizan el MPS, TLS y otras configuraciones. Working_dir establece el directorio de trabajo del peer.

```
services:
  orderer.example.com:
    container_name: orderer.example.com
    image: hyperledger/fabric-orderer:$IMAGE_TAG
    environment:..
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: orderer
    volumes:..
    ports:
      - 7050:7050
    networks:
      - pln

  peer0.org1.example.com:
    container_name: peer0.org1.example.com
    image: hyperledger/fabric-peer:$IMAGE_TAG
    environment:
      -
      CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=${COMPOSE_PROJECT_NAME}_
      pln
    ..
    - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
    - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
    - CORE_PEER_CHAINCODEADDRESS=peer0.org1.example.com:7052
    - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.example.com:7051
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051
    - CORE_PEER_LOCALMSPID=Org1MSP
    volumes:
    ...
    working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
    command: peer node start
    ports:
      - 7051:7051
    networks:
```

```
- pln
```

Cryptogen

Tenemos 4 configuraciones criptográficas: para el orderer y para cada una de las organizaciones. En el apartado OrdererOrgs se definen los nodos del servicio de ordenamiento, creando una definición de organización. En el apartado PeerOrgs se define los peers de la organización.

Como ya sabemos, para crear los componentes de la red, se requiere de una autoridad de certificación (CA). La herramienta de Fabric, cryptogen utiliza los archivos de configuración crypto para generar los certificados X.509 para todas las organizaciones.

Para el ordererOrgs, definimos la siguiente configuración crypto:

```
OrdererOrgs:
- Name: Orderer
  Domain: example.com
  EnableNodeOUs: true
  Specs:
  - Hostname: orderer
    SANS:
    - localhost
```

Para los peerOrgs, definimos la siguiente configuración crypto. Similar para las otras organizaciones.

```
PeerOrgs:
- Name: Org1
  Domain: org1.example.com
  EnableNodeOUs: true
  Template:
    Count: 1
    SANS:
    - localhost
  Users:
    Count: 1
```

Establecemos EnableNodeOUs como true, habilitando la clasificación de identificación.

Instalar Binarios e imágenes de Docker

Hasta ahora hemos revisado los archivos de configuración para ejecutar nuestra red PLN. net-pln.sh es un script para arrancar la red PLN. Para poder realizar este paso necesitamos descargar e instalar los binarios de Fabric. En los archivos del proyecto, encontramos el archivo loadFabric.sh, que nos permite descargar e instalar todo lo necesario.

```
./loadFabric.sh
```

Con el comando anterior se instalarán los archivos binarios y de configuración para Hyperledger Fabric en los directorios /bin y /config del proyecto. Podemos ejecutar el comando “docker images -a” para comprobar las imágenes de Fabric instaladas.

Note:

make sure all of the scripts in the project are executable. For example, You can run `chmod +x laodFabric.sh` to make it executable.

Ya podemos empezar a construir nuestra red PLN.

Iniciar la red PLN

Como hemos comentado antes, para iniciar la red PLN, necesitamos:

1. Generar las identidades de la organización y los peers con la herramienta cryptogen. Aquí está el comando para org1:

```
cryptogen generate --config=./organizations/cryptogen/crypto-config-org1.yaml --  
output="organizations"
```

La salida generada se almacenará en la carpeta organizations.

2. Generar las identidades para la organización orderer con la herramienta cryptogen.

```
cryptogen generate --config=./organizations/cryptogen/crypto-config-orderer.yaml --  
output="organizations"
```

3. Generar un perfil de conexión común (CCP) para Org1, Org2, Org3.

```
./organizations/ccp-generate.sh
```

El archivo ccp-generate.sh está en la carpeta de organizaciones. Este archivo utiliza los archivos ccp-template.json y ccp-template.yaml como plantilla, pasando el nombre de la organización, el puerto del mismo, el puerto CA y los certificados CA PEM para generar archivos de conexión de las organizaciones. El archivo ccp-generate.sh copia los archivos de conexión generados a la carpeta de organizaciones.

```
echo "$(json_ccp $ORG $P0PORT $CAPORT $PEERPEM $CAPEM)" >  
organizations/peerOrganizations/org1.example.com/connection-org1.json  
echo "$(yaml_ccp $ORG $P0PORT $CAPORT $PEERPEM $CAPEM)" >  
organizations/peerOrganizations/org1.example.com/connection-org1.yaml
```

Estos archivos de conexión serán utilizados por cada cliente web del peer para conectarse a la red de Fabric.

4. Crear un consorcio y generar un bloque génesis.

```
configtxgen -profile PharmaLedgerOrdererGenesis -channelID system-channel -outputBlock  
./system-genesis-block/genesis.block
```

La herramienta configtxgen utiliza el archivo configtx.yaml y genera el archivo genesis.block en la carpeta system-genesis-block.

5. Iniciar los nodos peer y orderer.

El archivo de docker compose se define en docker/docker-compose-pln-net.yaml. El comando compilará las imágenes del orderer y peers e iniciará los servicios que definimos en el archivo. yaml.

```
IMAGE_TAG=$IMAGETAG docker-compose ${COMPOSE_FILES} up -d 2>&1
```

A continuación, podemos iniciar la red PLN, ejecutando el archivo net-pln.sh en la carpeta pharma-ledger-network.

```
cd pharma-ledger-network
./net-pln.sh up
```

Creating network "net_pln" with the default driver				
Creating volume "net_orderer.example.com" with default driver				
Creating volume "net_peer0.org1.example.com" with default driver				
Creating volume "net_peer0.org2.example.com" with default driver				
Creating volume "net_peer0.org3.example.com" with default driver				
Creating orderer.example.com ... done				
Creating peer0.org2.example.com ... done				
Creating peer0.org1.example.com ... done				
Creating peer0.org3.example.com ... done				
CONTAINER ID	IMAGE	COMMAND	CREATED	
STATUS	PORTS	NAMES		
5a1fb5778a94	hyperledger/fabric-peer:latest	"peer node start"	4 seconds ago	
Up Less than a second	7051/tcp, 0.0.0.0:11051->11051/tcp	peer0.org3.example.com		
969a5a9f5a85	hyperledger/fabric-peer:latest	"peer node start"	4 seconds ago	
Up Less than a second	0.0.0.0:7051->7051/tcp	peer0.org1.example.com		
2f2cf2b0463d	hyperledger/fabric-peer:latest	"peer node start"	4 seconds ago	
Up 1 second	7051/tcp, 0.0.0.0:9051->9051/tcp	peer0.org2.example.com		

Deberíamos ver el siguiente registro (se muestra en la figura 8):

Figura 8. Registro

Tenemos 4 organizaciones, 3 peers y 1 orderer, en nuestra red net_pln. En el siguiente paso utilizaremos un script para crear un canal de comunicación para todas las organizaciones.

Monitorizar la red PLN

Las imágenes de Fabric en la red PLN están basadas en Docker. Durante el desarrollo del proyecto o ciclo de vida de producción, pueden aparecer varios errores. Es importante realizar una supervisión de registros, esto nos ayudará a solucionar problemas y encontrar la causa mucho más fácil y rápido. La herramienta logspout nos permite monitorizar los eventos de docker. En nuestro proyecto, utilizaremos logspout para monitorear la creación de canales, smart contracts y otras acciones. Accedemos a la carpeta pharma-ledger-network:

```
cd pharma-ledger-network
```

Ejecutamos el siguiente comando que iniciará la herramienta logspout en los contenedores que se ejecutan en la red PLN net_pln:

```
./net-pln.sh monitor-up

...
Starting docker log monitoring on network 'net_pln'
Starting monitoring on all containers on the network net_pln
Unable to find image 'gliderlabs/logspout:latest' locally
latest: Pulling from gliderlabs/logspout
cbdbe7a5bc2a: Pull complete
956fa3cf18b6: Pull complete
94f24e0675e0: Pull complete
Digest: sha256:872555b51b73d7f50726baeae8d8c138b6b48b550fc71d733df7ffcadc9072e1
Status: Downloaded newer image for gliderlabs/logspout:latest
e8a8ad1787b69cfb7387264ee6ff63fd5a805aabe50ca6af6356d4cd8b27e052
```

Aquí podemos ver la lógica del script para activar la herramienta logspout.

```
docker run -d --name="logspout" \
  --volume=/var/run/docker.sock:/var/run/docker.sock \
  --publish=127.0.0.1:${PORT}:80 \
  --network ${DOCKER_NETWORK} gliderlabs/logspout
```

Esta ventana del terminal mostrará la salida de los contenedores de la red PLN.

Consejo:

Si durante el proceso tiene problemas, consulte este terminal para ver los errores.

Creando un canal en la red PLN

En el proceso de creación del canal, utilizaremos la herramienta CLI configtxgen para generar un bloque génesis, seguidamente utilizaremos los comandos del peer para añadir al canal cada peer. Toda la lógica de creación del canal se puede encontrar en scripts/createChannel.sh.

El primer paso, generamos la transacción de configuración del canal.

En el script createChannel.sh, tenemos la función createChannelTxn, que ejecuta el siguiente comando:

```
configtxgen -profile PharmaLedgerChannel -outputCreateChannelTx ./channel-  
artifacts/${CHANNEL_NAME}.tx -channelID $CHANNEL_NAME
```

La herramienta configtxgen selecciona el perfil PharmaLedgerChannel del archivo configtx.yaml que define la configuración relacionada con el canal para generar la transacción y el bloque génesis.

En segundo lugar, creamos el archivo de transacción de configuración del AnchorPeer.

Tenemos definida la función createAnchorPeerTxn. La herramienta configtxgen lee la configuración organizativa de PharmaLedgerChannel y genera los archivos de transacción de configuración del peer.

```
configtxgen -profile PharmaLedgerChannel -outputAnchorPeersUpdate ./channel-  
artifacts/${orgmsp}anchors.tx -channelID $CHANNEL_NAME -asOrg ${orgmsp}
```

Después de que se ejecute la función createAnchorPeerTxn, deberíamos ver los archivos de transacción Org1MSPanchors.tx, Org2MSPanchors.tx y Org3MSPanchors.tx generados.

En tercer lugar, crearemos un canal utilizando el comando peer channel.

La función createChannel utiliza el comando "peer channel create" para crear el canal PLN. Cuando se ejecuta el comando, enviará la transacción de creación de canal al ordering service. El ordering service comprobará los permisos de la política de creación de canales definidos en configtx.yaml. Solo los usuarios administradores pueden crear un canal.

La función setGlobalVars () en scripts / utils.sh nos permitirá establecer cierta organización como el usuario administrador. Usamos org1 como administrador para crear nuestro canal.

Los comandos para crear un canal son los siguientes:

```
setGlobalVars 1
peer channel create -o localhost:7050 -c $CHANNEL_NAME --ordererTLSHostnameOverride
orderer.example.com -f ./channel-artifacts/${CHANNEL_NAME}.tx --outputBlock ./channel-
```

La función `setGlobalVars()` in `scripts/utls.sh` tienen la siguiente lógica para configurar la `org1` como usuario administrador. También podemos usar esta función para configurar otras organizaciones como usuario administrador.

```
setGlobalVars() {
..
if [ $USING_ORG -eq 1 ]; then
    export CORE_PEER_LOCALMSPID="Org1MSP"
    export CORE_PEER_TLS_ROOTCERT_FILE=$PEER0_ORG1_CA
    export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.c
om/users/Admin@org1.example.com/msp
    export CORE_PEER_ADDRESS=localhost:7051
..
}
```

Una vez el canal esta creado, añadiremos los peers al canal.

En cuarto lugar, unir los peers al canal.

Una vez creado nuestro canal PLN, podemos unir los peers al canal. La función `joinMultiPeersToChannel` en el script `createChannel.sh` unirá los peers de nuestras organizaciones en nuestro canal ejecutando el comando “peer channel join”.

```
for org in $(seq 1 $TOTAL_ORGS); do
    setGlobalVars $ORG
    peer channel join -b ./channel-artifacts/$CHANNEL_NAME.block >&log.txt
done
```

Cuando los peers de las organizaciones se unen al canal, deben asignarse como usuarios administradores llamando a la función `setGlobalVars` pasando el parámetro `$ORG`. El comando “peer channel join” utiliza el `genesis.block` para unir los peers al canal. Una vez el peer esta unido al canal, puede participar en la creación de bloques al recibir el envío por parte del ordering service.

En quinto lugar, actualizar `AnchorPeer` para los peers.

Como último paso en el proceso de creación del canal, debemos seleccionar al menos un peer como un anchor peer. La función principal de un anchor peer es el descubrimiento de servicios

y datos privados. Los endpoint de un anchor peer son fijos, otros peers pertenecientes a miembros diferentes pueden comunicarse con el anchor peer para descubrir todos los peers existentes en un canal. Para actualizar un anchor peer, seleccionamos un peer como usuario administrador y emitimos el comando de actualización del canal del peer.

```
setGlobalVars $ORG
peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -
c $CHANNEL_NAME -f ./channel-artifacts/${CORE_PEER_LOCALMSPID}anchors.tx --tls
$CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA >&log.txt
```

Podemos utilizar el script net-pln.sh para crear el canal PLN, ejecutando el comando:

```
./net-pln.sh createChannel
```

Una vez completada la creación del canal, deberíamos ver el siguiente registro:

```
***** [Step: 5]: start call updateAnchorPeers 3 on peer: peer0.org3, channelID: plnchannel,
smartcontract: , version , sequence *****
Using organization 3
2020-06-06 03:24:36.333 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and
orderer connections initialized
2020-06-06 03:24:36.392 UTC [channelCmd] update -> INFO 002 Successfully submitted
channel update
***** completed call updateAnchorPeers, updated peer0.org3 on anchorPeers on channelID:
plnchannel, smartcontract: , version , sequence *****

***** completed call updateOrgsOnAnchorPeers, anchorPeers updated on channelID:
plnchannel, smartcontract: , version , sequence *****

===== Pharma Ledger Network (PLN) Channel plnchannel successfully joined
=====
```

Para ver la información relacionada con el contenedor, podemos consultar la ventana de terminal de logspout que abrimos anteriormente.

4. Ejecutando y testeando el smart contract

Necesitamos empaquetar un smart contract antes de poder instalarlo en el canal. Accedemos al directorio de la carpeta del contrato del fabricante y ejecutamos el comando npm install:

```
cd pharma-ledger-network/organizations/manufacturer/contract
npm install
```

Esto instalará las dependencias del contrato en la carpeta node_modules.

Ahora podemos comenzar a instalar nuestro smart contract ejecutando el siguiente script deploySmartContract.sh.

El primer paso es empaquetar el smart contract pharmaledger.

El comando peer lifecycle chaincode package empaquetará nuestro smart contract. Asignamos al fabricante como usuario administrador para ejecutar el comando.

```
setGlobalVars 1
peer lifecycle chaincode package ${CHINCODE_NAME}.tar.gz --path ${CC_SRC_PATH} --
lang ${CC_RUNTIME_LANGUAGE} --label ${CHINCODE_NAME}_${VERSION}
```

A continuación, podemos instalar el chaincode en todos los peers de las organizaciones como administrador con el comando peer lifecycle chaincode install:

```
for org in $(seq 1 $CHAINCODE_ORGS); do
setGlobalVars $ORG
peer lifecycle chaincode install ${CHINCODE_NAME}.tar.gz >&log.txt
done
```

Cuando el paquete de chaincode esté instalado, aparecerán mensajes similares en el terminal:

```
2020-06-06 03:30:50.025 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 001
Installed remotely: response:<status:200
payload:"\nWpharmaLedgerContract_1:1940852a477d7697bb3a12d032268ff48c741c585db1
66403dd35f5e0b5c4e74\022\026pharmaLedgerContract_1" >
2020-06-06 03:30:50.025 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 002
Chaincode code package identifier:
pharmaLedgerContract_1:1940852a477d7697bb3a12d032268ff48c741c585db166403dd35f5
e0b5c4e74
***** completed call installChaincode, Chaincode is installed on peer0.org1 on channelID:
plnchannel, smartcontract: pharmaLedgerContract, version 1, sequence 1 *****
```


Después de instalar el smart contract, debemos consultar si el chaincode está instalado. Podemos consultar el ID del paquete usando el comando `peer lifecycle chaincode queryinstalled`.

```
peer lifecycle chaincode queryinstalled >&log.txt
```

Si el comando se ha completado correctamente, podremos ver un mensaje similar al siguiente:

```
Installed chaincodes on peer:
Package ID:
pharmaLedgerContract_1:1940852a477d7697bb3a12d032268ff48c741c585db166403dd35f5
e0b5c4e74, Label: pharmaLedgerContract_1
***** completed call queryInstalled, Query installed successful with PackageID is
pharmaLedgerContract_1:1940852a477d7697bb3a12d032268ff48c741c585db166403dd35f5
e0b5c4e74 on channelID: plnchannel, smartcontract: pharmaLedgerContract, version 1,
sequence 1 *****
```

Con el ID de paquete devuelto, ahora podemos aprobar el chaincode para el fabricante mediante el uso de `ApproveForMyOrg`, que llama al comando `peer lifecycle chaincode approveformyorg`.

```
peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA --
channelID $CHANNEL_NAME --name ${CHAINCODE_NAME} --version ${VERSION} --
package-id ${PACKAGE_ID} --sequence ${VERSION} >&log.txt
```

Podemos comprobar si los miembros del canal han aprobado la misma definición chaincode utilizando `checkOrgsCommitReadiness`, que ejecuta el comando `peer lifecycle chaincode checkcommitreadiness`.

```
peer lifecycle chaincode checkcommitreadiness --channelID $CHANNEL_NAME --name
${CHAINCODE_NAME} --version ${VERSION} --sequence ${VERSION} --output json >&log.txt
```

Como era de esperar, deberíamos ver que las aprobaciones de `Org1MSP` son verdaderas y las otras dos organizaciones son falsas.

```

***** [Step: 5]: start call checkCommitReadiness org1 on peer: peer0.org1, channelID:
plnchannel, smartcontract: pharmaLedgerContract, version 1, sequence 1 *****
Attempting to check the commit readiness of the chaincode definition on peer0.org1, Retry
after 3 seconds.
+ peer lifecycle chaincode checkcommitreadiness --channelID plnchannel --name
pharmaLedgerContract --version 1 --sequence 1 --output json
{
  "approvals": {
    "Org1MSP": true, "Org2MSP": false, "Org3MSP": false
  }
}

```

Las políticas de endorsamiento requieren que un conjunto de organizaciones mayoritarias respalden una transacción antes de que se valide el chaincode.

Continuamos ejecutando el comando `peer lifecycle chaincode approveformyorg` para Org2 y Org3, las tres organizaciones aprobarán la instalación del chaincode.

```

## approve org2
approveForMyOrg 2
## check whether the chaincode definition is ready to be committed, two orgs should be
approved
checkOrgsCommitReadiness 3 1 1 0
## approve org3
approveForMyOrg 3
## check whether the chaincode definition is ready to be committed, all 3 orgs should be
approved
checkOrgsCommitReadiness 3 1 1 1

```

Si todos los comandos se ejecutan correctamente, las tres organizaciones obtendrán la aprobación de la instalación del chaincode.

```

{
  "approvals": {
    "Org1MSP": true, "Org2MSP": true, "Org3MSP": true
  }
}
***** completed call checkCommitReadiness, Checking the commit readiness of the chaincode
definition successful on peer0.org3 on channel 'plnchannel' on channelID: plnchannel,

```

Ahora que sabemos que el fabricante, el mayorista y la farmacia han aprobado el pharmaledgercontract chaincode, podemos validar el chaincode. Tenemos las organizaciones necesarias (3 de 3) para asignar la definición del chaincode al canal. Cualquiera de las tres organizaciones puede enviar el chaincode al canal mediante el comando peer lifecycle chaincode commit.

```
peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA --
channelID $CHANNEL_NAME --name ${CHAINCODE_NAME} $PEER_CONN_PARMS --
version ${VERSION} --sequence ${VERSION} >&log.txt
```

Usaremos el comando peer lifecycle chaincode querycommitted para verificar el estado del chaincode

```
peer lifecycle chaincode querycommitted --channelID $CHANNEL_NAME --name
${CHAINCODE_NAME} >&log.txt
```

Hemos visto los pasos necesarios para desplegar chaincode el canal, por lo que ahora podemos ejecutar siguiente comando para desplegar el pharmaledgercontract chaincode en la red PLN.

```
./net-pln.sh deploySmartContract
```

Si el comando ha funcionado correctamente, deberíamos ver la siguiente respuesta:

```
Committed chaincode definition for chaincode 'pharmaLedgerContract' on channel
'plnchannel':
Version: 1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals:
[Org1MSP: true, Org2MSP: true, Org3MSP: true]
***** completed call queryCommitted, Query committed on channel 'plnchannel' on channelID:
plnchannel, smartcontract: pharmaLedgerContract, version 1, sequence 1 *****
***** completed call queryAllCommitted, Chaincode installed on channelID: plnchannel,
smartcontract: pharmaLedgerContract, version 1, sequence 1 *****
=== Pharma Ledger Network (PLN) contract successfully deployed on channel plnchannel
====
```

Una vez instalado el chaincode, podemos comenzar a invocar y probar las funciones del chaincode pharmaledgercontract.

Testeando el smart contract

Hemos creado el script `invokeContract.sh` para este proyecto. El script ejecuta las funciones `makeEquipment`, `wholesalerDistribute`, `pharmacyReceived` y la función de consulta.

Primero, ejecutamos la función del chaincode `makeEquipment`:

```
./net-pln.sh invoke equipment GlobalEquipmentCorp 2000.001 e360-Ventilator  
GlobalEquipmentCorp
```

Pasamos los argumentos `manufacturer`, `equipmentNumber`, `equipmentName` y `ownerName`. El script básicamente llama a los comandos `peer chaincode invoke` pasando los argumentos necesarios para cada función.

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride  
orderer.example.com --tls $CORE_PEER_TLS_ENABLED --cafile $ORDERER_CA -C  
$CHANNEL_NAME -n ${CHAINCODE_NAME} $PEER_CONN_PARMS -c  
'{"function":"makeEquipment","Args":["$manufacturer","$equipmentNumber",  
"$equipmentName","$ownerName"]}' >&log.txt
```

En los registros podemos ver lo siguiente:

```
invokeMakeEquipment--> manufacturer:GlobalEquipmentCorp, equipmentNumber:2000.001,  
equipmentName: e360-Ventilator,ownerName:GlobalEquipmentCorp  
+ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride  
orderer.example.com --tls true --cafile /home/ubuntu/Hyperledger-Fabric-V2/project7-  
supplychain/pharma-ledger-  
network/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/  
tlscacerts/tlsca.example.com-cert.pem -C plnchannel -n pharmaLedgerContract --  
peerAddresses localhost:7051 --tlsRootCertFiles /home/ubuntu/Hyperledger-Fabric-  
V2/project7-supplychain/pharma-ledger-  
network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tl  
s/ca.crt --peerAddresses localhost:9051 --tlsRootCertFiles /home/ubuntu/Hyperledger-Fabric-  
V2/project7-supplychain/pharma-ledger-  
network/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tl  
s/ca.crt --peerAddresses localhost:11051 --tlsRootCertFiles /home/ubuntu/Hyperledger-Fabric-  
V2/project7-supplychain/pharma-ledger-  
network/organizations/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tl  
s/ca.crt -c '{"function":"makeEquipment","Args":["GlobalEquipmentCorp","2000.001", "e360-  
Ventilator", "GlobalEquipmentCorp"]}'  
2020-06-06 03:43:09.186 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001  
Chaincode invoke successful. result: status:200
```

Después de invocar la función `makeEquipment`, podemos ejecutar una función de consulta para verificar el resultado de la ledger. La función de consulta utiliza el comando `peer chaincode query`.

```
peer chaincode query -C $CHANNEL_NAME -n ${CHAINCODE_NAME} -c  
'{"function":"queryByKey","Args":["$QUERY_KEY"]}' >&log.txt
```

Podemos utilizar el siguiente script para consultar la información de un equipo.

```
./net-pln.sh invoke query 2000.001
```

La consulta (query) nos devolverá los datos de estado actuales del equipo.

```
{"Key":"2000.001","Record":{"equipmentNumber":"2000.001","manufacturer":"GlobalEquipmen  
tCorp","equipmentName":"e360-  
Ventilator","ownerName":"GlobalEquipmentCorp","previousOwnerType":"MANUFACTURER"  
,"currentOwnerType":"MANUFACTURER","createDateTime":"Sat Jun 06 2020 03:43:09  
GMT+0000 (Coordinated Universal Time)","lastUpdated":"Sat Jun 06 2020 03:43:09  
GMT+0000 (Coordinated Universal Time)"}}
```

Podemos continuar invocando el resto de funciones.

```
./net-pln.sh invoke wholesaler 2000.001 GlobalWholesalerCorp  
./net-pln.sh invoke pharmacy 2000.001 PharmacyCorp
```

Una vez que el propietario del equipo es la farmacia, la cadena de suministro alcanza su estado final. Podemos ejecutar la función `queryHistoryByKey`, utilizando el comando `peer chaincode query command`. Podemos revisar el histórico de datos del equipo.

```
./net-pln.sh invoke queryHistory 2000.001
```

Podemos ver la siguiente salida en el terminal.

```
***** start call chaincodeQueryHistory on peer: peer0.org1, channelID: plnchannel,
smartcontract: pharmaLedgerContract, version 1, sequence 1 *****
+ peer chaincode query -C plnchannel -n pharmaLedgerContract -c
'{"function":"queryHistoryByKey","Args":["2000.001"]}'
[{"equipmentNumber":"2000.001","manufacturer":"GlobalEquipmentCorp","equipmentName":"
e360-
Ventilator","ownerName":"PharmacyCorp","previousOwnerType":"WHOLESALER","currentOw
nerType":"PHARMACY","createDateTime":"Sat Jun 06 2020 03:43:09 GMT+0000
(Coordinated Universal Time)","lastUpdated":"Sat Jun 06 2020 03:48:48 GMT+0000
(Coordinated Universal
Time)"},"{"equipmentNumber":"2000.001","manufacturer":"GlobalEquipmentCorp","equipmentN
ame":"e360-
Ventilator","ownerName":"GlobalWholesalerCorp","previousOwnerType":"MANUFACTURER
","currentOwnerType":"WHOLESALER","createDateTime":"Sat Jun 06 2020 03:43:09
GMT+0000 (Coordinated Universal Time)","lastUpdated":"Sat Jun 06 2020 03:46:41
GMT+0000 (Coordinated Universal
Time)"},"{"equipmentNumber":"2000.001","manufacturer":"GlobalEquipmentCorp","equipmentN
ame":"e360-
Ventilator","ownerName":"GlobalEquipmentCorp","previousOwnerType":"MANUFACTURER"
,"currentOwnerType":"MANUFACTURER","createDateTime":"Sat Jun 06 2020 03:43:09
GMT+0000 (Coordinated Universal Time)","lastUpdated":"Sat Jun 06 2020 03:43:09
GMT+0000 (Coordinated Universal Time)"}]
***** completed call chaincodeQuery, Query History successful on channelID: plnchannel,
smartcontract: pharmaLedgerContract, version 1, sequence 1 *****
```

Todo el registro de las transacciones se muestra en el terminal. Hemos podido comprobar que el smart contract funciona como se esperaba.

5. Desarrollo de una aplicación con el SDK de Hyperledger Fabric

Acabamos de implementar la red de Fabric pharma-ledger-network. El siguiente paso es crear una aplicación cliente para interactuar con las funciones del smart contract desplegado en la red. Dediquemos un momento a examinar la arquitectura de la aplicación.

Al comienzo de la creación de la red PLN, generamos un perfil de conexión común (CCP) para Org1, Org2 y Org3. Usaremos estos archivos de conexión para conectarnos a nuestra red PLN para cada organización. Cuando el usuario de la aplicación del fabricante, Bob, envía una transacción makeEquipment a la ledger, se inicia el flujo del proceso de la red. Examinemos rápidamente cómo funciona nuestra aplicación, como se muestra en la figura 9.

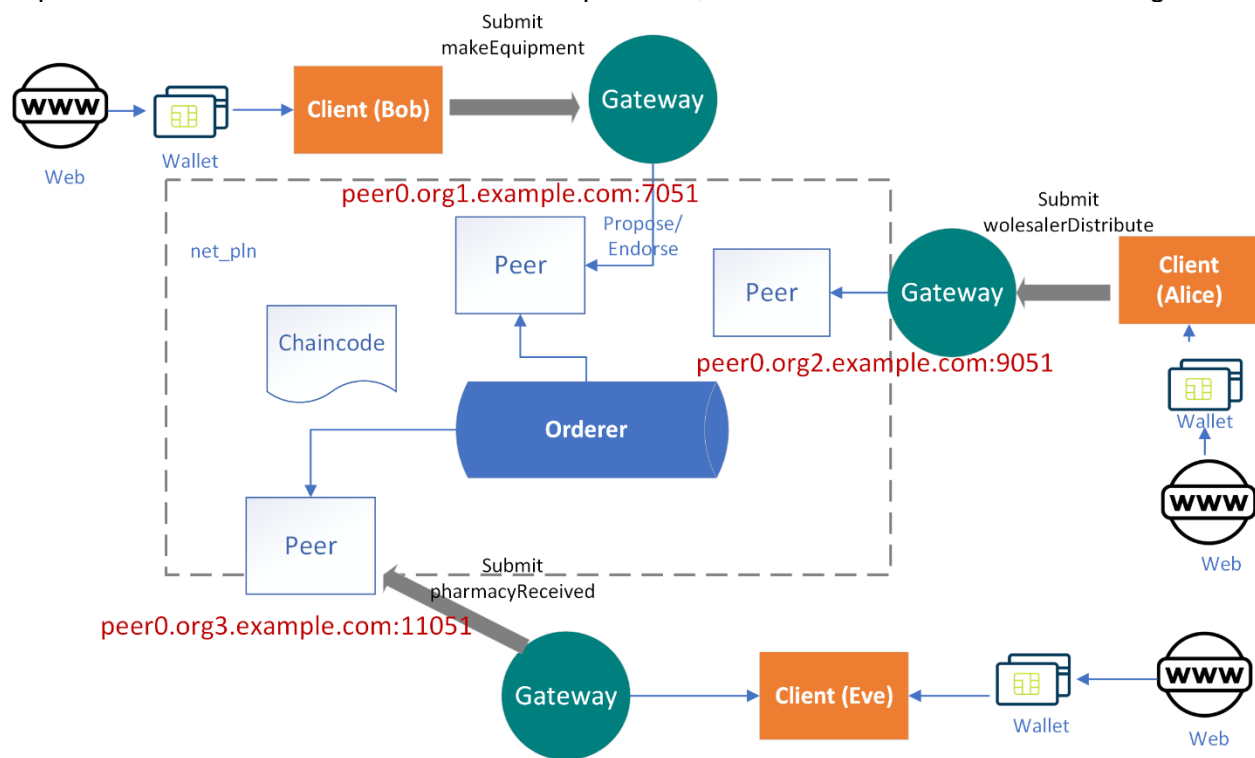


Figura 9. Como funciona nuestra aplicación PLN

Bob, el usuario web del fabricante, se conecta a la red Fabric a través de una billetera (wallet). Wallet proporciona al usuario una identidad autorizada que será verificada por la red blockchain para garantizar la seguridad del acceso. Luego, Fabric SDK envía una propuesta de transacción `makeEquipment` a `peer0.org1.example.com`. El endorsing peer verifica la firma, simula la propuesta e invoca la función `makeEquipment` del chaincode con los argumentos necesarios. La transacción se inicia después de que la "respuesta a la propuesta" se envíe de vuelta al SDK. La aplicación recopila y verifica el endorsamiento con las políticas de endorsamiento del chaincode que se satisface produciendo el mismo resultado. Luego, el cliente "transmite" la propuesta de transacción y la "respuesta de la propuesta" al ordering

service. El ordering service ordena cronológicamente las transacciones por canal y crea bloques y entrega los bloques de transacciones a todos los peers del canal. Los peers validan las transacciones para garantizar que se cumplan las políticas de endosamiento y para asegurarse de que no ha habido cambios en el estado de la ledger desde que la ejecución de la transacción generó la respuesta a la propuesta. Después de una validación exitosa, el bloque se confirma en la ledger y los estados se actualizan.

Ahora entendemos el flujo de trabajo de la transacción de un extremo a otro. Es hora de comenzar a construir nuestra aplicación cliente. A continuación, se muestra la estructura del proyecto.

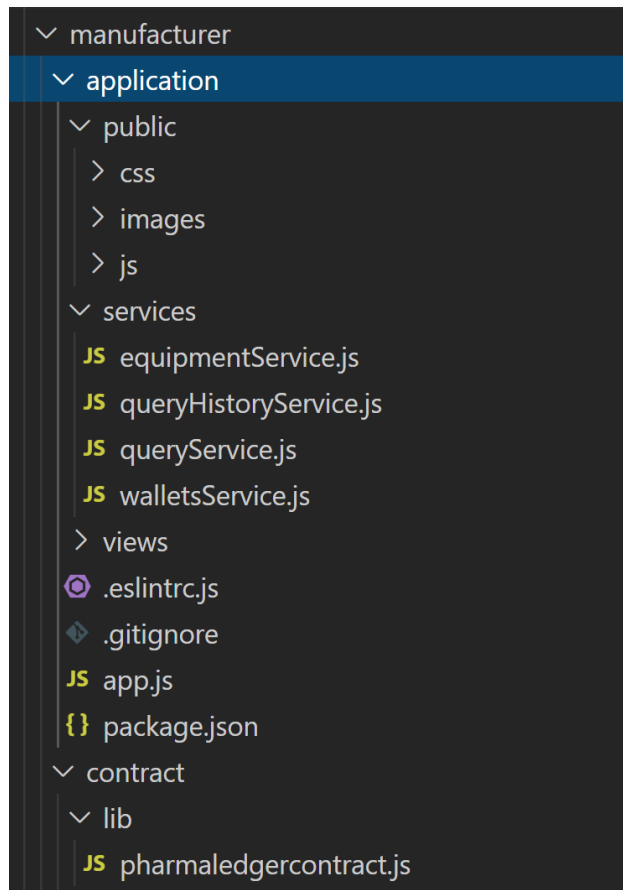


Figura 10. Estructura del proyecto de aplicación cliente

Usamos express.js para crear nuestra aplicación basada en node. Vamos a revisar algunos archivos importantes.

package.json se definen dos dependencias relacionadas con Fabric.


```
"dependencies": {
  "fabric-contract-api": "^2.1.2",
  "fabric-shim": "^2.1.2"
}
```

app.js define todos los puntos de entrada para el fabricante y addUser agregará un usuario cliente para el fabricante, que en nuestro caso es Bob. makeEquipment creará los registros de equipos cuando el fabricante sea propietario. queryByKey y queryHistoryByKey son funciones comunes para las tres organizaciones. Mayorista y Farmacia tendrán funciones similares.

```
app.post('/addUser', async (req, res, next) => {
});
app.post('/makeEquipment', async (req, res, next) => {
})
app.get('/queryHistoryByKey', async (req, res, next) => {
})
app.get('/queryByKey', async (req, res, next) => {
})
```

addUser utilizará walletsService para agregar un usuario. Echemos un vistazo a addToWallet (usuario) en walletsService.

```
const wallet = await Wallets.newFileSystemWallet('../identity/user/'+user+'/wallet');
```

newFileSystemWallet creará una billetera para un usuario de entrada (Bob) en el directorio del sistema de archivos proporcionado. A continuación, encontramos el certificado de usuario y la clave privada y generamos el certificado X.509 para almacenarlo en la billetera.

```
const credPath = path.join(fixture,
'/peerOrganizations/org1.example.com/users/User1@org1.example.com');
const certificate = fs.readFileSync(path.join(credPath,
'/msp/signcerts/User1@org1.example.com-cert.pem')).toString();
const privateKey = fs.readFileSync(path.join(credPath, '/msp/keystore/priv_sk')).toString();
```

La billetera utiliza funciones de la clase key para administrar X509WalletMixin.createIdentity que se usa para crear una identidad Org1MSP usando credenciales X.509. La función necesita tres entradas: mspid, el certificado y la clave privada.

```
const identityLabel = user;
const identity = {
  credentials: {
    certificate,
    privateKey
  },
  mspId: 'Org1MSP',
  type: 'X.509'
}
const response = await wallet.put(identityLabel, identity);
```

El usuario del fabricante llamará a la función equipmentService makeEquipment.

Primero, busque la billetera de usuario creada agregando la función de usuario.

```
const wallet = await Wallets.newFileSystemWallet('../identity/user/'+userName+'/wallet');
```

A continuación, cargue el perfil de conexión asociado con el usuario, luego la billetera se usará para ubicar una puerta de enlace (Gateway) y conectarse a ella.

```
const gateway = new Gateway();
let connectionProfile =
yaml.safeLoad(fs.readFileSync('../organizations/peerOrganizations/org1.example.com/connection-org1.json', 'utf8'));
// Set connection options; identity and wallet
let connectionOptions = {
  identity: userName,
  wallet: wallet,
  discovery: { enabled:true, asLocalhost: true }
};
await gateway.connect(connectionProfile, connectionOptions);
```

Una vez que una puerta de enlace (Gateway) está conectada a un canal, podemos encontrar nuestro PharmaLedgerContract con su nombre único cuando creamos el contrato.

```
const network = await gateway.getNetwork('plnchannel');
const contract = await network.getContract('pharmaLedgerContract',
'org.pln.PharmaLedgerContract');
```

Ya podemos enviar la invocación makeEquipment del chaincode.

```
const response = await contract.submitTransaction('makeEquipment', manufacturer,
equipmentNumber, equipmentName, ownerName);
```

Para verificar que el registro de los equipos esta almacenado en la Blockchain podemos usar funciones de consulta (query). El siguiente código muestra cómo podemos enviar una consulta o una función queryHistory para obtener los resultados del equipo.

```
const response = await contract.submitTransaction('queryByKey', key);
const response = await contract.submitTransaction('queryHistoryByKey', key);
```

A continuación, buscaremos un fabricante y crearemos el usuario de Bob, luego enviaremos una transacción a nuestra blockchain PLN.

Accedemos a la carpeta pharma-ledger-network/organizations/manufacturer/application, y ejecutamos el comando npm install. Importante asegurarse de actualizar la dirección IP del cliente en plnClient.js en public / js

```
var urlBase = " http://your-machine-public-ip:30000";
```

```
npm install
pharma-ledger-network/organizations/manufacturer/application$ node app.js
App listening at http://:::30000
```

Como fabricante, definimos el puerto de aplicación 30000.

Nota:

Asegúrese de que este puerto esté abierto o puede cambiarlo a otro puerto disponible en la línea app.js

```
var port = process.env.PORT || 30000;
```

Abra un navegador y accede a [http:// your-machine-public-ip: 30000](http://your-machine-public-ip:30000)

Veremos la pantalla que se muestra en la figura 11.

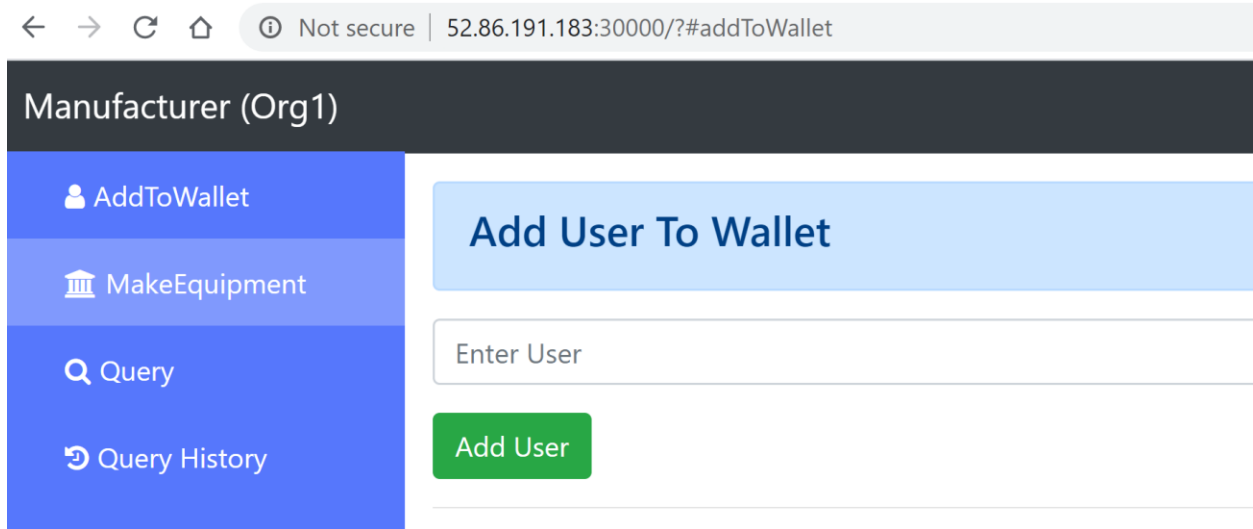


Figura 11. Añadir un usuario a la wallet del fabricante

La página predeterminada es addToWallet. Dado que no hemos agregado ningún usuario a la billetera hasta ahora, no podemos enviar makeEquipment ni consultar las transacciones del historial en este momento. Tenemos que agregar un usuario en la billetera. Agreguemos a Bob como usuario fabricante, como se muestra en la figura 12.

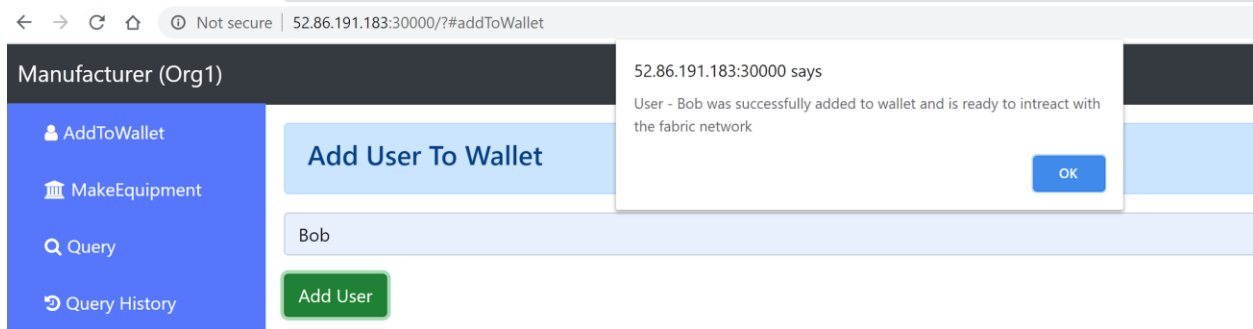


Figura 12. Nuevo usuario añadido (Bob)

Con la billetera del usuario configurada, la aplicación ahora puede conectarse a nuestra red PLN e interactuar con el chaincode. Haga clic en MakeEquipment en el menú de la izquierda, ingrese toda la información necesaria del equipo y envíe la solicitud (Figura 13). La respuesta

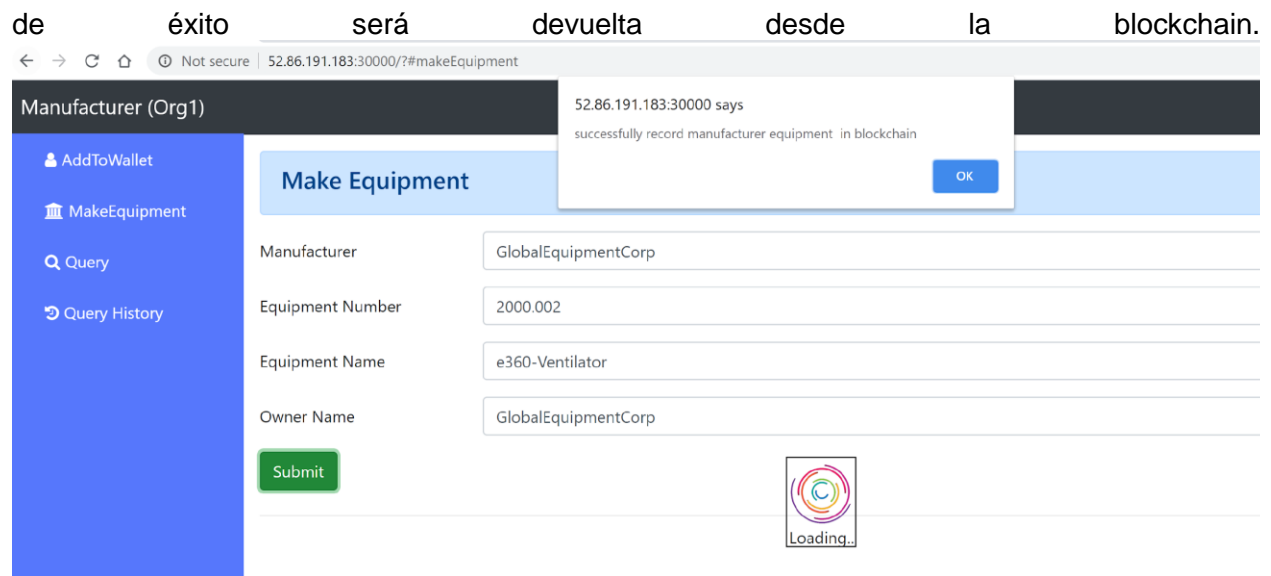


Figura 13. Añadiendo un equipo a la red PLN

Ahora podemos consultar los datos del equipo en la red PLN utilizando el número identificador del equipo, el resultado será como en la Figura 14.

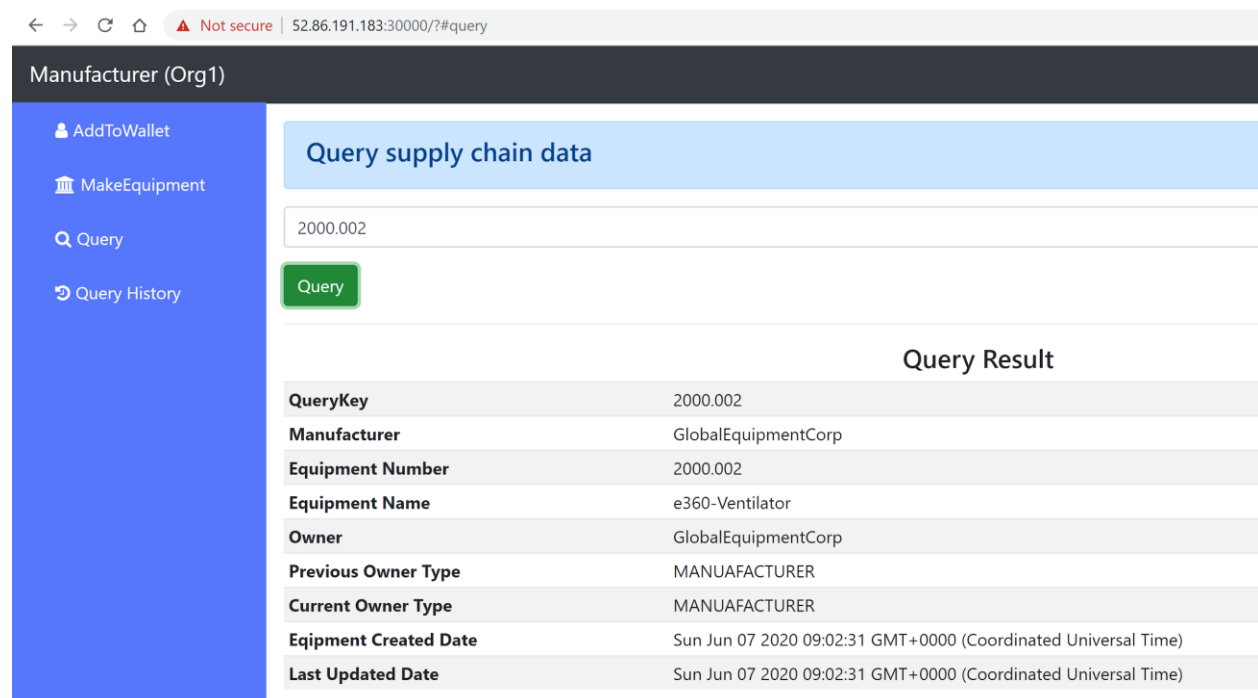


Figura 14. Consulta de datos de un equipo en la red PLN

A continuación, abrimos dos nuevas terminales para iniciar las aplicaciones para mayorista y farmacia.

Accedemos a:

/pharma-ledger-network/organizations/wholesaler/contract

Ejecutamos 'npm install' para instalar las dependencias de la aplicación.

Actualizamos la URL `http: // your-machine-public-ip: 30001` en `plnClient.js`. Seguidamente, volvemos a la carpeta `pharma-ledger-network/organizations/wholesaler/application` y ejecutamos:

```
npm install
```

```
node app.js
```

Esto inicia la aplicación web para mayoristas. Abra un navegador e ingrese: `http: // your-machine-public-ip: 30001`

Agregamos Alice como usuario mayorista (Figura 15) y ejecutamos `wholesalerDisctribute`.

The screenshot shows a web browser window with the address bar displaying `52.86.191.183:30001/?#wholesalerDistribute`. The page title is "Wholesaler (Org2)". A dark blue sidebar on the left contains the following menu items: "AddToWallet", "WholesalerDistribute", "Query", and "Query History". The main content area has a light blue header with the text "Wholesaler Distribute" and an "OK" button. Below this, there is a form with two input fields: "Equipment Number" with the value "2000.002" and "Owner Name" with the value "GlobalWholesalerInc". A green "Submit" button is located below the "Owner Name" field. A white modal box is open in the center of the screen, displaying the message: "52.86.191.183:30001 says successfully record wholesaler Distribute in blockchain".

Figura 15. Nuevo usuario mayorista añadido

Seguimos los mismos pasos para iniciar la aplicación para la farmacia y añadimos un nuevo usuario. Finalmente ejecutamos PharmacyReceived. (Figura 16).

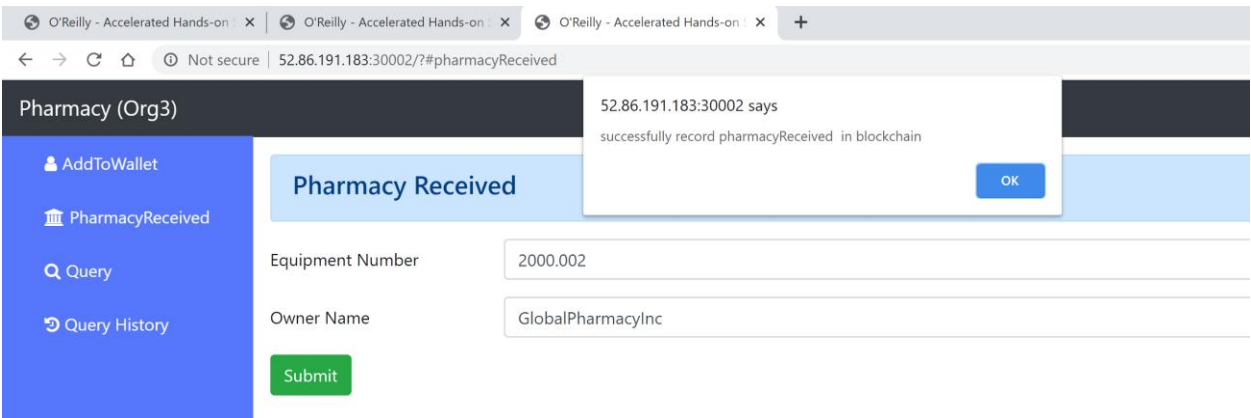


Figura 16. Ejecutamos Pharmacy Received.

Una vez finalizado el flujo de trabajo de la cadena de suministro farmacéutica, Bob, Alice y Eve pueden consultar datos de los equipos y rastrear todo el proceso de la cadena de suministro consultando datos del histórico. Simplemente podemos utilizar cualquier usuario y en la página "Query History", buscamos el equipo 2000.002, deberíamos ver todo el historial de transacciones como se muestra en la Figura 17.

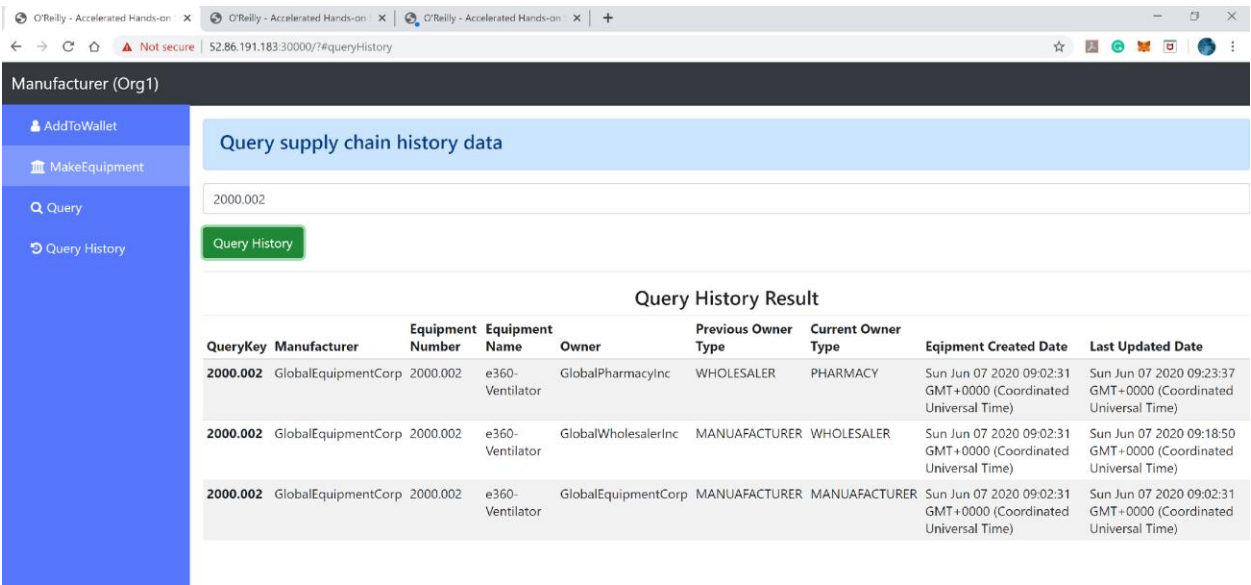


Figura 17. Historial de transacciones de un equipo

Resumen

En este proyecto, hemos aprendido cómo construir DApps de cadena de suministro con Hyperledger Fabric. Hemos introducido, entre otras cosas, cómo definir un consorcio, analizar el ciclo de vida de la cadena de suministro y comprender cómo rastrear todo el historial de transacciones de los equipos. Hemos escrito chaincode como un contrato inteligente, incluidas las lógicas de contrato inteligente de fabricante, mayorista y farmacia. Después de configurar la red pharmaledger en Fabric, instalamos e implementamos nuestro contrato inteligente en blockchain.

También probamos las funciones del smart contract a través de un script para asegurarnos de que todas las funciones que definimos funcionaban como se esperaba. Finalmente, empezamos a trabajar en la parte de la aplicación cliente, donde aprendimos cómo agregar usuarios a una billetera, cómo conectar con nuestra red de Fabric a través del SDK. También creamos las páginas de interfaz de usuario para el fabricante, el mayorista y la farmacia que permiten a los usuarios de estas organizaciones interactuar con el smart contract de la blockchain PLN.

Puede ver que es bastante trabajo construir la aplicación Hyperledger Fabric completa. Así que esperamos que no esté cansado, ya que, en el próximo curso, exploraremos otro tema interesante: Implementar Hyperledger Fabric en la nube.