Objetivo

- El objetivo es aprender a realizar las siguientes operaciones
  - Desplegar sobre nuestra red Hyperledger Fabric el chaincode creado previamente usando el SDK de Fabric
  - Consumir las funciones de nuestro contrato, tanto de lectura como de escritura

- Pasos:
  - Partiremos de la red privada Hyperledger Fabric desplegada previamente
  - App de gestión usando SDK de NodeJS que nos permitirá:
    - Instalar contratos en los peers
    - Instanciar contratos en el canal
  - Crear una app cliente usando SDK de NodeJS
    - Acceder como un miembro de una organización
    - Consumir las funciones definidas en el Smart Contract

TENEMOS
MUCHO
QUE HACER
JUNTOS

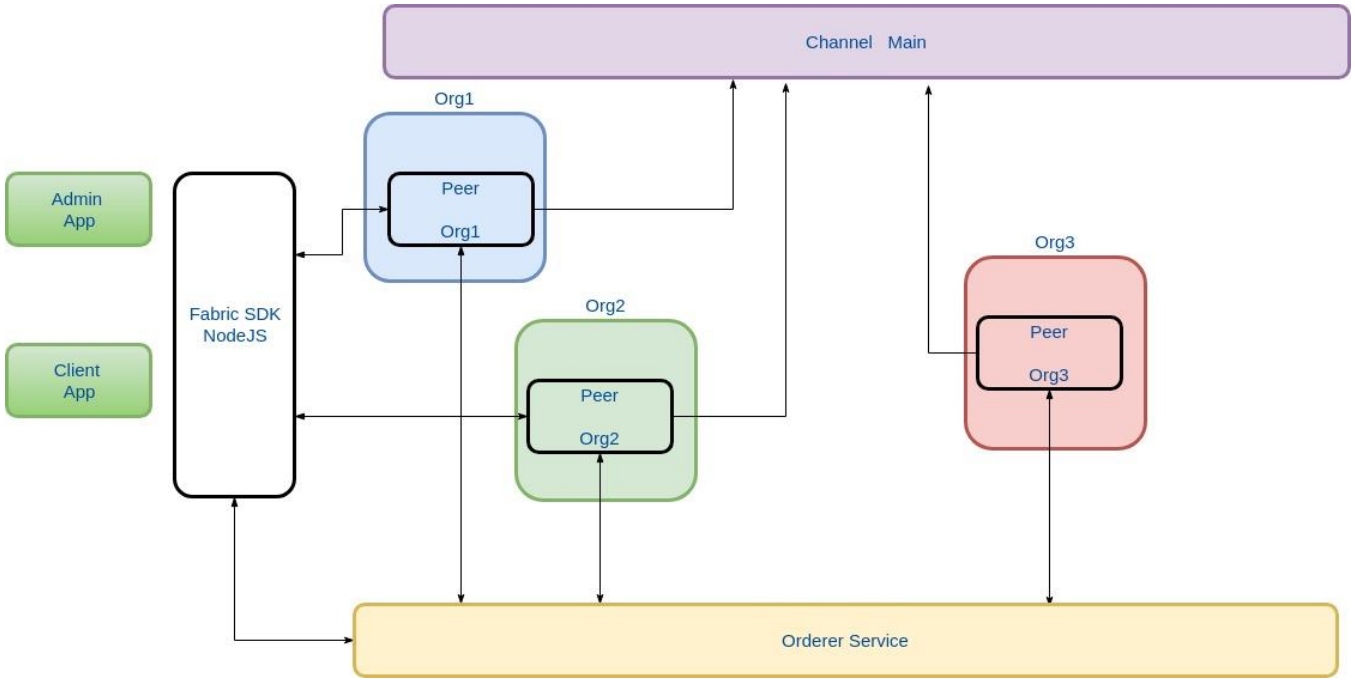
4 - Interacción con la Blockchain y Smart Contracts

tecnalia Inspiring Business

Práctica – Requisitos previos

- Requisitos
  - Hyperledger Fabric Network arrancada
  - Canal Main con dos organizaciones
  - NodeJS v8.9.0

Arquitectura



Componentes

- **Orderer:** acepta transacciones respaldadas, las ordena en un bloque, y entrega los bloques a los peers commiters
- **Peer**: mantienen el estado de la red así como una copia de la cadena.
  - Endosers: simulan y respaldan las transacciones
  - Committers: verifican las transacciones respaldadas y validan los resultados de la transacción antes de comprometer las transacciones a la cadena
- **Certification Authority:** entidad que permite gestionar los certificados de usuario
- **Client**: representa la entidad que actúa en nombre del usuario

Transaction Flow

- Es el proceso de llegar a un acuerdo sobre el siguiente conjunto de operaciones que se añadirán al ledger

- En Hyperledger Fabric se compone de tres grandes pasos:
  - Endorse transaction
  - Ordering
  - Commit transaction

Transaction Flow

- Es importante tener en cuenta que el estado de la red es mantenido por los peers, y no por el ordering service o el cliente
- Los peers endosers deben tener acceso a los chaincodes, además de cumplir el papel de un peer commiter. Sin embargo los peers commiters no es necesario que tengan acceso al chaincode
- Los pares endosers verifican la firma del cliente y ejecutan una función de código de cadena para simular la transacción
- La respuesta de la propuesta se envía de vuelta al cliente, junto con una firma del endorsment
- Estas respuestas a la propuesta se envían al ordering service, ordena las transacciones en un bloque, que reenvía a los peers del canal
- Los peers verifican las transacciones y actualizan el estado
- Finalmente, los pares notifican asincrónicamente a la aplicación cliente del éxito o fracaso de la transacción

Interacción blockchain: Comando *peer*

- Información acerca del comando:
  - **peer –help**

- Instalar chaincode en cada peer:
  - **peer chaincode install** -n myChaincode -v 1.0 -p **/**chaincode_example02

- Instanciar un chaincode
  - **peer chaincode instantiate** -C mychannel -n mycc -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P "OR ('Org1MSP.peer','Org2MSP.peer')"

Interacción blockchain: Fabric SDK

- Stable
  - NodeJS - https://github.com/hyperledger/fabric-sdk-node
  - Java - https://github.com/hyperledger/fabric-sdk-java

- In Incubation
  - Go - https://github.com/hyperledger/fabric-sdk-go
  - Python - https://github.com/hyperledger/fabric-sdk-py

- WIP
  - REST - https://github.com/hyperledger/fabric-sdk-rest

Fabric SDK NodeJS

- Usa gRPC para comunicarse con la red de Blockchain
  - Principal problema para usar el sdk en el navegador

- Las principales responsabilidades:
  - Comunicarse con la CA para obtener los certificados de cliente
  - Consumir funciones de los chaincodes
  - Realizar labores administrativas (instalar e instanciar chaincodes)

- Lo componen:
  - **fabric-ca**: interacciona con la CA para la gestión de los usuarios y los certificados
  - **fabric-client:** implementa el "transaction flow"

Fabric SDK NodeJS – Fabric CA

- Hyperledger Fabric es permisionada y por lo tanto es necesario disponer de un usuario para el acceso a la red

- Mediante el SDK podemos implementar el ciclo de vida de un usuario
  - **Register**: registrar un nuevo usuario
  - **Enroll**: obtener de un usuario que existe los certificados firmados de la CA
  - **Revoke**: revocar un usuario existente o un certificado en concreto

- Las claves (pública y privada) de un usuario del que se ha hecho enroll son almacenados por el SDK en el State Store

Fabric SDK NodeJS – Fabric Client

- Operaciones de interacción con la red Blockchain
  - crear canales
  - pedir a los peers que se añadan al canal
  - instalar chaincodes en los peers
  - instanciar chaincodes en el canal
  - información sobre la red Blockchain
    - información de bloques
    - información de transacciones
    - chaincodes instalados en un peer
    - ….

- Operaciones de interacción con el chaincode desplegado
  - invocar transacciones llamando al chaincode
  - realizar peticiones al chaincode sobre la situación actual

Fabric SDK NodeJS – State Store

- Se encarga de persistir datos por parte del SDK
  - Datos de usuario
  - KeyStore

- Permite escalar entre varias instancias
  - NodeJS en modo cluster
  - Varias máquinas "físicas"

- Existen dos implementaciones del State Store
  - Implementación con escritura en ficheros (por defecto)
  - Implementación en CouchDB

- Sencillo crear e usar nuestro propio State Store

Fabric SDK NodeJS – Connection Profile

- En la versión 1.1 se introdujo el concepto de connection profile
- Para que el SDK funcione correctamente es necesario que disponga de la información de la composición de red (organizaciones, canales, peer por canal, orderer,..) para ello se usan el Network connection profile que es un espejo de la red
- Permite mucha flexibilidad a la hora de parametrizar el SDK
- Varios niveles de configuración:
  - uno global de la red donde se configura toda la red
  - particular por organización

Fabric SDK NodeJS – Class Client

- Permite interaccionar con los orderes y peers de la red

- Es necesario configurar el contexto de usuario antes del envío del cualquier comando a la red Blockchain para poder firmar todas las peticiones
  - **setUserContext**: asigna al cliente la instancia de usuario que recibe como parámetro y almacena la información de usuario en el "State Store" configurado
  - **createUser**: permite crear una instancia de la clase User usando private keys y certificados pre-creados como alternativa a hacer un enroll a una CA. Realiza automáticamente un setUserContext con la instancia de usuario

- Operaciones de cliente Chaincode
  - **installChaincode**: permite instalar chaincodes en los peers. Esta operación debe ser realizada "a mano" en cada de los peers (independientemente del canal) que vayan a invocar algun metodo del chaincode

Fabric SDK NodeJS – Class Channel

- Los canales nos permiten generan aislar los datos y mantener la privacidad estableciendo un canal para que las organizaciones puedan compartir información

- Operaciones de admin de red Blockchain:
  - **sendInstantiateProposal**: permite inicializar un chaincode en los peer endosers del canal

- Operaciones de cliente Chaincode
  - **queryByChaincode**: realizar operaciones de lectura a través de los chaincode instanciados
  - **sendTransaccionProposal**: envía una "transaction proposal" a uno o mas endorsing peers
  - **sendTransaccion**: enviar el "transaction message" al canal que se lo entregará a los orderers configurados para que sean commiteados por los peers committers

Fabric SDK NodeJS – Class EventHub

- La clase EventHub nos permite conectarnos a los event's stream de cada uno de los peers

- Los principales métodos son:
  - **connect**: se conecta a un peer
  - **registerTxEvent**: a partir de un txId se recibe un evento cuando su estado cambia (falla una verificación, se ha incluido en la cadena,...)
  - **registerBlockEvent**: se recibe un evento cuando se añade un bloque a la cadena. Si un peer pertenece a varios canales el bloque puede venir de cualquiera de ellos
  - **registerChaincodeEvent**: permite subscribirnos a eventos emitidos por un chaincode. Nuestro peer tiene que pertenecer al canal donde el chaincode ha sido instanciado

Práctica - Estructura de ficheros

- Ficheros:
  - **chaincode**: chaincode a desplegar
  - **despliegue**: ficheros del despliegue realizado
  - **network.yaml**: connection profile de la red desplegada
  - **org1.yaml**: fichero de configuración del cliente que se conectara a la organización1
  - **org2.yaml**: fichero de configuración del cliente que se conectara a la organización2
  - **package.json**: descriptor de la app
  - **requirements.sh**: script de resolución de dependencias
  - **sdk**: librería que encapsula las funciones de query, invoke y enroll
  - **users.json**: json con las credenciales de los usuarios de las organizaciones ya que no disponemos de CA

- Ejercicios:
  - admin.js
  - client.js

Práctica - Admin App

- Vamos a generar una app que como administrador de la red instale e instancie el chaincode generado
- Paso previo: Modificar users.json para indicar las private keys del despliegue ya que no disponemos de una CA
- Pasos:
  1. Genera un instancia de cliente SDK por cada una de las organizaciones
  2. Hace enroll por cada de los usuarios
  3. Instala en cada uno de los peers endosers el chaincode
  4. Instancia el contrato en el canal
- Probar la aplicación
  - node admin.js

Práctica – Client APP

- Vamos a generar una app que actúe como cliente de nuestro chaincode y haga invoke y query de nuestro chaincode

- Paso previo: indicar el nombre del chaincode desplegado a través de la Admin app

- Pasos
  1. Genera un instancia de cliente SDK de cada una de las organizaciones
  2. Hacer enroll por cada usuario (no admin) generado en el despliegue
  3. Realizamos un invoke desde el cliente de la org1 para generar un certificado
  4. Realizamos un query desde el cliente de la org2 para obtener el certificado que hemos creado previamente

- Probar la aplicación
  - node admin.js