
JORNADA:
Oportunidades de negocio
a través de Internet of Things (IoT) y Blockchain
Tecnologías Block Chain aplicadas a la industria del siglo XXI
Dr. Ing. Ind. Javier Sedano
ITCL/Grupo EAIA
"Este proyecto está cofinanciado por la Unión Europea a través del Fondo Europeo de Desarrollo Regional (FEDER) a través del Programa Interreg V-A España-Portugal (POCTEP) 2014-2020"
Interreg
España - Portugal
IOTEC
competitividad empresarial
ice
Junta de Castilla y León

¿Quienes somos?

Dr. Javier Sedano

Doctor en Sistemas Inteligentes para la Ingeniería Industrial, experto en el desarrollo de sistemas electrónicos y desarrollo de modelos conexionistas. Experto en el desarrollo de soluciones y resolución de problemas industriales.

UNIVERSIDAD DE BURGOS

UNED

universidad de león

GRUPO DE INVESTIGACION EN ELECTRONICA APLICADA E INTELIGECIA ARTIFICIAL EAIA

Contenido.

- Tecnologías
  - Bitcoin
  - Ethererum
  - Redes laterales
  - Lighting
  - Raiden
  - Ripple
  - Hyperledger
- Hyperledger fabric.

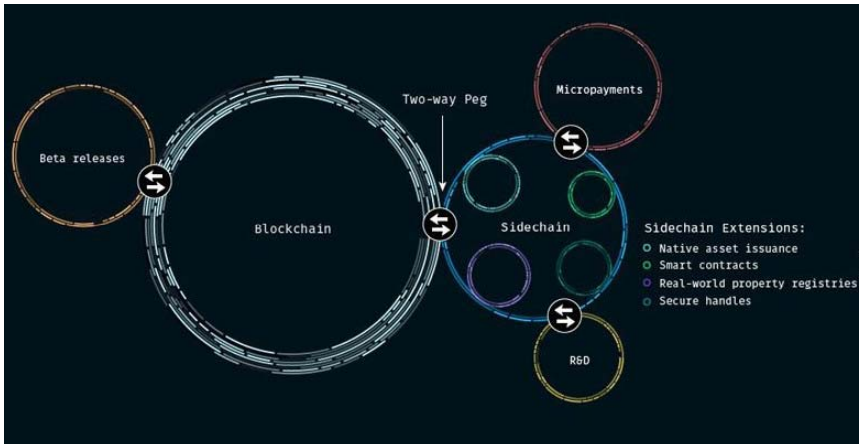
BitCoin

- Aplicación descentralizada creada por Satoshi Nakamoto.
- Primera **red blockchain.**
- Bitcoin es tanto la red como la moneda digital.

Ethereum

- Aplicación descentralizada creada por Vitalik Buterin.
- Sirve para realizar **transacciones** con contratos inteligentes.
- Se usan valores de gas para cuantificar pasos computacionales y cuotas a pagar.

Redes laterales



- Red que realiza transacciones **apoyándose** en la red principal.
- Ayuda a **mejorar el rendimiento** de la red principal. Incrementa enormemente la cantidad de transacciones que se realizan por segundo.
- Reduce la sobrecarga en las redes principales, liberándolas de realizar micro transacciones.

Redes laterales: lighting

- Capa creada sobre Bitcoin con el objetivo de realizar transacciones mas rápidas sin la repercusión de la ralentización de la red principal.
- Permite la interacción directa entre dos usuarios.
- Mantiene la descentralización de la red Bitcoin.

Redes laterales: lighting

- Las transacciones son publicadas en la blockchain principal, por lo que hay total transparencia.
- Permite realizar transacciones con intermediarios. No necesitas conocer al destinatario, solo a alguien que le conozca.

Redes laterales: raiden

- Proyecto open-source cuyo objetivo es hacer mas escalable la red de Ethereum.
- Red de transferencia fuera de cadena para los tokens de ERC20.
- Reduce la tarifa(gas) que se necesita para realizar una transacción.

Redes laterales: raiden

- Las transacciones son privadas entre los usuarios y los nodos que formen parte de la transacción.
- Una vez que termina la transacción, el estado final será visible para todos.

Ripple

- Software libre.
- Sistema descentralizado que conecta las diferentes redes con el objetivo de que una transacción sea mas directa.
- Red de pagos e intercambio de divisas.
- La transacción se realiza a través de redes de confianza.
- Permite la aparición de estructuras jerárquicas de pago.

HYPERLEDGER

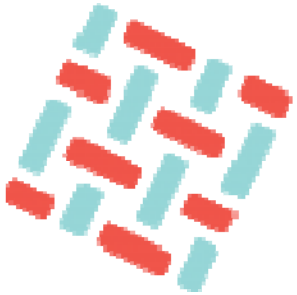
- Red de desarrollo de smartcontracts desarrollada por la fundación Linux.
- Completamente modularizada.
- Existen diferentes proyectos basados en este estilo de desarrollo de aplicaciones blockchain.

Hyperledger

- Hyperledger Borrow: proporciona un cliente modular de blockchain con un interprete de smartcontracts.
- Hyperledger Sawtooth: utiliza un algoritmo desarrollado por Intel llamado Prueba de tiempo transcurrido(PoeT)
- Hyperledger fabric: implementación plug&play de la tecnología blockchain implementado por IMB.
- Hyperledger cello: implementación bajo demanda con el objetivo de simplificar la creación de apliaciones blockchain.
- Hyperledger composer: conjunto de herramientas que simplifican y agilizan la construcción de una red en hyperledger con JavaScript.

Índice

- ¿Qué es hyperledger fabric?
- Requisitos básicos.
- Red básica
  - Funcionamiento.
  - Elementos:
    - Peer
    - Chaincode/SmartContract
    - Ledger
    - Orderer
    - Certificados de identidad
    - Miembros u organización
    - Consenso
    - Canal
    - Anchor peer
- Dockers.

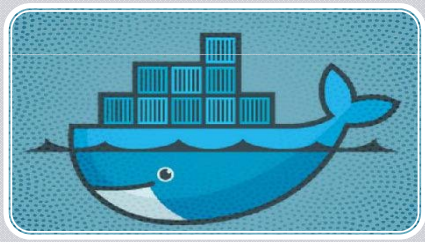


HYPERLEDGER FABRIC

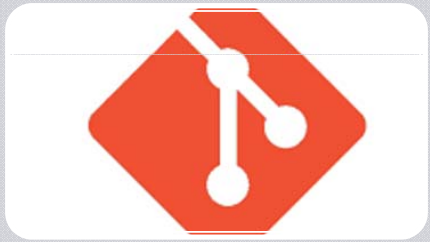
¿Qué es Hyperledger Fabric?

- Proyecto de código abierto creado por la fundación Linux.
- Su objetivo es crear ecosistemas de la distribución del ledger, pero en la práctica se centra en el desarrollo de aplicaciones BlockChain.
- Es la infraestructura que provee el ledger y el chainCode a las aplicaciones. Contiene el resto de elementos que forman la red.

Requisitos básicos: HyperledgerFabric 1.3



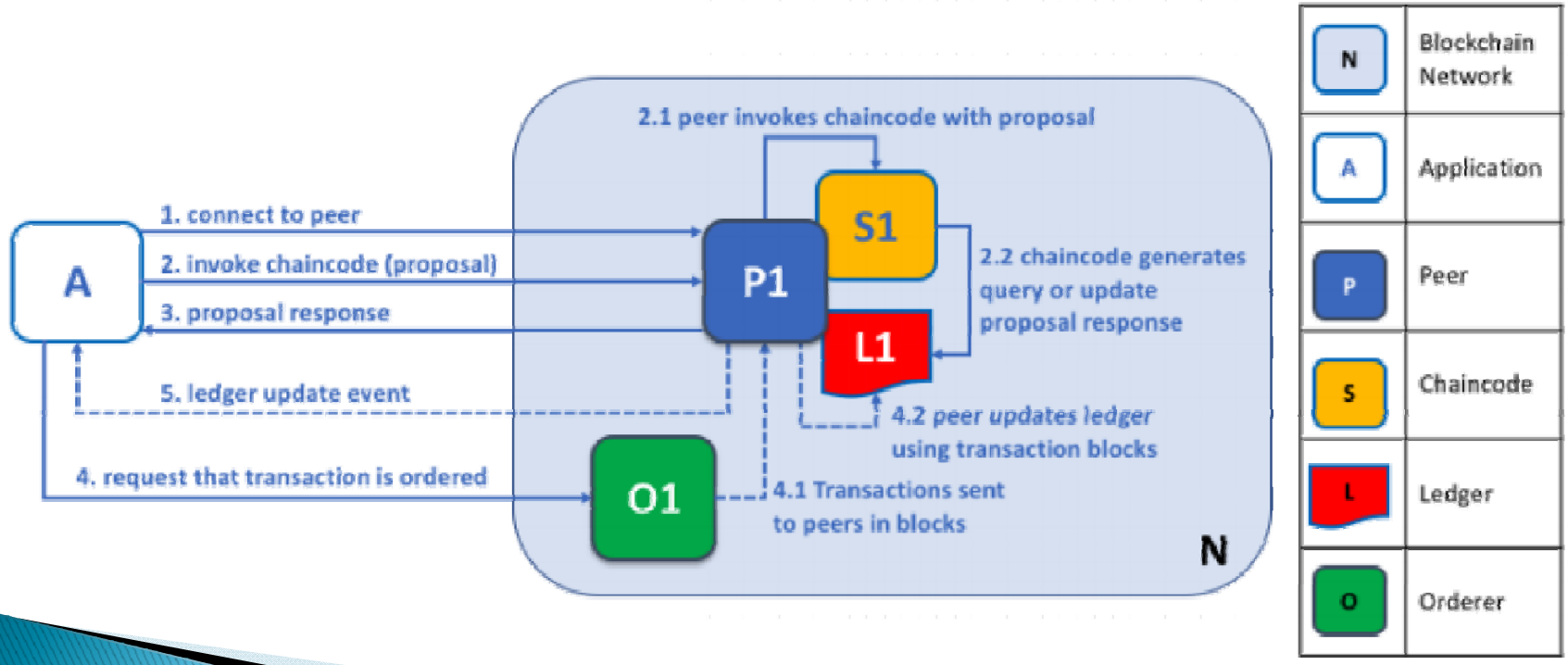
- Docker >=v17.06
- Docker-compose >=v1.14



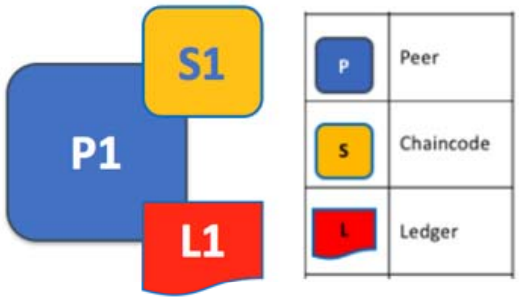
- cURL
- Windows-build-tools
- GRPC
- Git

- Go >=v1.11
- Java >=8
- Node.js >=v8.9
- Python >=v3.5

Red básica y su funcionamiento.



Elementos:Peer.



- **Elemento principal de una red.**
- Contienen los ledger y los chaincode.
- El peer se relacionará con la aplicación y los usuarios para realizar las transacciones.
- Una Peer puede contener mas de un ledger y de un chaincode.

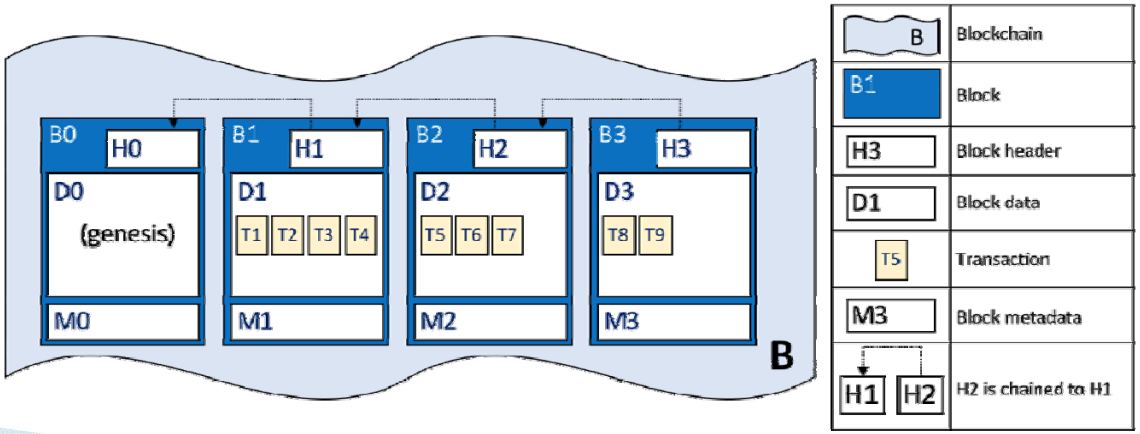
Elementos: ChainCode o SmartContract.

- Es la lógica de la red, interactúa con el ledger y los usuarios para realizar las transacciones. Si una transacción es valida se agregara al ledger.

Elementos: Ledger.

- El ledger es la base de datos donde se guardan todas las transacciones validadas.
- Su funcionamiento es similar a las estructuras diccionario de Python o java→ (Clave,Valor).
- Su característica principal es que es inmutable. Todos los Ledger empiezan por un bloque, llamado bloque Génesis.

- Existen dos implementaciones del ledger:
  - **CouchDB** que permite consultas enriquecidas y guarda objetos tipo JSON
  - **LevelDB** es la implementación predeterminada y mas simple. Se usa cuando los datos a guardar son simples



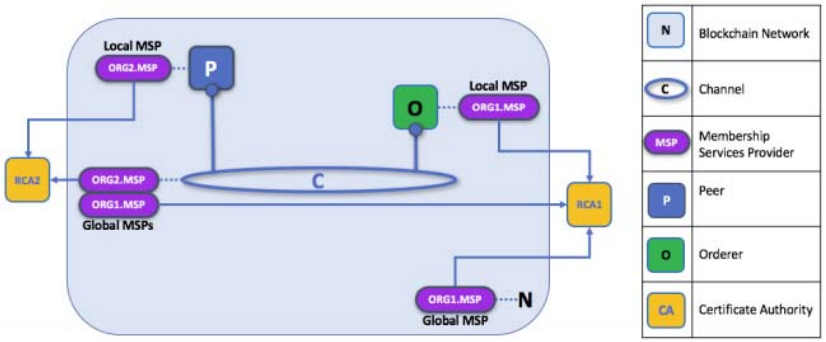
Elementos: Orderer.

- Nodo cuyo objetivo es mantener la coherencia del ledger gracias a los diferentes consensos con los que funcione la red.
- Al estar presentes en todas las transacciones, implementan una garantía de entrega entre los dos usuarios involucrados.

Elementos: Certificados de identidad.

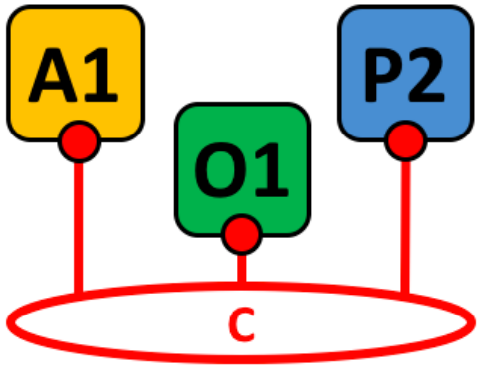
- Con la herramienta cryptogen se generan los certificados de identidad para autentificar la autoría de los canales, los bloques génesis y los usuarios.
- Sin esta característica no se podría autenticar la identidad de los actores en una transacción.

Elementos: Miembros u organización.



- El objetivo de los miembros u organizaciones es verificar las membresías de los diferentes participantes de una transacción.

Elementos: Canales.

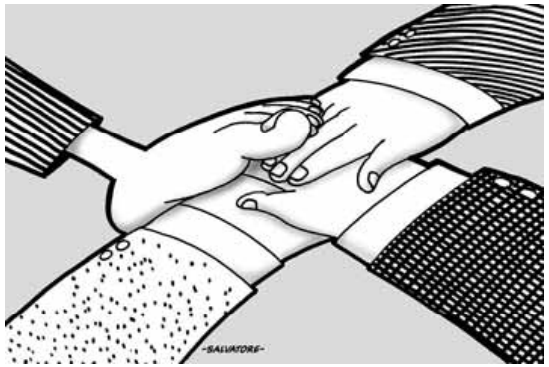


- Un canal es una visión reducida y privada de la red de hyperledger fabric. Un canal implementa un, un peer (con o sin leger) y un orderer y el ususario (A).
- Gracias a los canales se puede mantener la confidencialidad y el aislamiento de lo datos.

Elementos: Anchor Peer

- La función de los anchor peer es dar a conocer a los peer, dentro de un canal y organización, la existencia de otros peer que no conocen.
- Sin la existencia de los anchor peer, un peer solo podría conocer al resto de peers dentro del mismo canal y tendría que navegar para conocer al resto de peers.

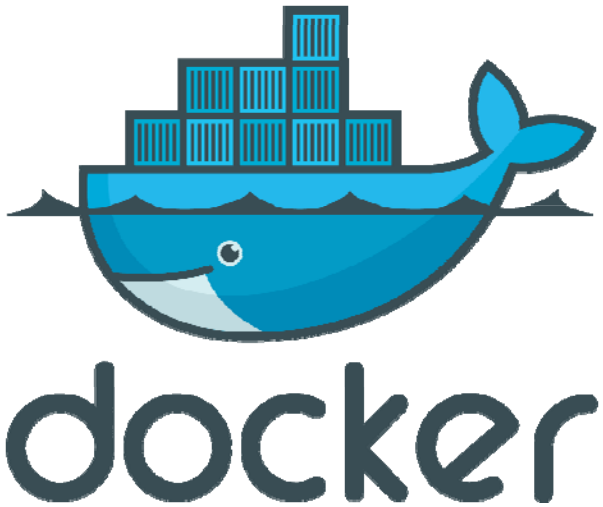
Elementos: Consenso



En la red de Hyperledger fabric hay 3 formas ya implementadas para realizar un consenso de si una transacción es valida o no.

- **Solo**, estas solo en el sistema, todas las transacciones serán correctas.
- **Kafka**, es el predeterminado. Sistema de reenvió de un mensaje a todos los peers del canal. Los peers contestan y se realiza el consenso.
- **BFT**, el sistema elige a unos "validadores" y son ellos los encargados de realizar el consenso.

Docker.



- La instalación de la red de hyperledger fabric funciona sobre los Docker.
- Hay una serie de dockers básicos:
  - Fabric-ca, es el emisor de los certificados de identidad.
  - Fabric-peer: Peers
  - Fabric-couchDB, es la base de datos donde se guarda el ledger.
  - Fabric-orderer: Orders

Ejemplo chainCode en Java.

- Método obligatorio.
- INIT
- Se usa para realizar una instancia o actualización.

```java
public Response init(ChaincodeStub stub) {
    try {
        _logger.info("Init java simple chaincode");
        String func = stub.getFunction();

        if (!func.equals("init")) {
            return newErrorResponse("function other than init is not supported");
        }
        List<String> args = stub.getParameters();
        if (args.size() != 4) {
            newErrorResponse("Incorrect number of arguments. Expecting 4");
        }
        // Initialize the chaincode
        String account1Key = args.get(0);
        int account1Value = Integer.parseInt(args.get(1));
        String account2Key = args.get(2);
        int account2Value = Integer.parseInt(args.get(3));

        _logger.info(String.format("account %s, value = %s; account %s, value %s", accou
        stub.putStringState(account1Key, args.get(1));
        stub.putStringState(account2Key, args.get(3));

        return newSuccessResponse();
    } catch (Throwable e) {
        return newErrorResponse(e);
    }
}
```

Ejemplo chainCode en Java.

- **Método obligatorio.**
- **Invoke**
- Se usa cuando se quiere interactuar con el Ledger al realizar una transacción.
- Ejemplos de transacción:
  - {“Arg”:[“init”,”a”,”100”,”b”,”200”]}
  - {“Arg”:[“move”,”a”,”b”,”10”]}

```
public Response invoke(ChaincodeStub stub) {
    try {
        _logger.info("Invoke java simple chaincode");
        String func = stub.getFunction();
        List<String> params = stub.getParameters();
        if (func.equals("move")) {
            return move(stub, params);
        }
        if (func.equals("delete")) {
            return delete(stub, params);
        }
        if (func.equals("query")) {
            return query(stub, params);
        }
        return newErrorResponse("Invalid invoke function nar
    } catch (Throwable e) {
        return newErrorResponse(e);
    }
}
```

Ejemplo chainCode en Java.

- Método principal que implementa la lógica del SmartContract. **Cuerpo del chainCode**
- Este SmartContract realiza transacciones “monetarias” entre dos usuario, comprobando que el usuario que envía el dinero en realidad tiene como mínimo esa cantidad.
- Al realizar el envió de dinero guarda los dos nuevos estados en el Ledger .

```
private Response move(ChaincodeStub stub, List<String> args) {
    if (args.size() != 3) {
        return newErrorResponse("Incorrect number of arguments. Expecting 3");
    }
    String accountFromKey = args.get(0);
    String accountToKey = args.get(1);
    String accountFromValueStr = stub.getStringState(accountFromKey);
    if (accountFromValueStr == null) {
        return newErrorResponse(String.format("Entity %s not found", accountFromKey));
    }
    int accountFromValue = Integer.parseInt(accountFromValueStr);

    String accountToValueStr = stub.getStringState(accountToKey);
    if (accountToValueStr == null) {
        return newErrorResponse(String.format("Entity %s not found", accountToKey));
    }
    int accountToValue = Integer.parseInt(accountToValueStr);

    int amount = Integer.parseInt(args.get(2));

    if (amount > accountFromValue) {
        return newErrorResponse(String.format("not enough money in account %s", accountFromKey));
    }
    accountFromValue -= amount;
    accountToValue += amount;
    _logger.info(String.format("new value of A: %s", accountFromValue));
    _logger.info(String.format("new value of B: %s", accountToValue));

    stub.putStringState(accountFromKey, Integer.toString(accountFromValue));
    stub.putStringState(accountToKey, Integer.toString(accountToValue));
    _logger.info("Transfer complete");
    Map<String, byte[]> transientMap = stub.getTransient();
    if (null != transientMap) {
        if (transientMap.containsKey("event") && transientMap.get("event") != null) {
            stub.setEvent("event", transientMap.get("event"));
        }
        if (transientMap.containsKey("result") && transientMap.get("result") != null) {
            return newSuccessResponse(transientMap.get("result"));
        }
    }
    return newSuccessResponse();
                toByteArray());
}
```

Tomamos el estado actual del Ledger.

Comprobamos que los datos son correctos.

Realizamos la transacción.

Guardamos el nuevo estado.

Ejemplo chainCode en Java.

- Métodos propios que complementan la lógica del chainCode.
- Por ejemplo:
- Borrado de canales en Peers, estado de usuari etc.

```java
// Deletes an entity from state
private Response delete(ChaincodeStub stub, List<String> args) {
    if (args.size() != 1) {
        return newErrorResponse("Incorrect number of arguments. Expecting 1");
    }
    String key = args.get(0);
    // Delete the key from the state in ledger
    stub.delState(key);
    return newSuccessResponse();
}

// query callback representing the query of a chaincode
private Response query(ChaincodeStub stub, List<String> args) {
    if (args.size() != 1) {
        return newErrorResponse("Incorrect number of arguments. Expecting name of the person to query");
    }
    String key = args.get(0);
    //byte[] stateBytes
    String val = stub.getStringState(key);
    if (val == null) {
        return newErrorResponse(String.format("Error: state for %s is null", key));
    }
    _logger.info(String.format("Query Response:\nName: %s, Amount: %s\n", key, val));
    return newSuccessResponse(val, ByteString.copyFrom(val, UTF_8).toByteArray());
}

public static void main(String[] args) {
    System.out.println("OpenSSL avaliable: " + OpenSsl.isAvailable());
    new SimpleChaincode().start(args);
}
```

Dr. Javier Sedano
Andrés Martinéz

Teléfonos: 947 29 84 71 **Fax:** 947 29 80 91 **E-mail:** info@itcl.es **Web site:** www.itcl.es/