UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Sistema de trazabilidad basado en DLT (Distributed Ledger Technology)

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jorge Justo Giménez Duro

Tutor: Lenin Guillermo Lemus Zuniga

Curso 2020-2021

Resum

Les economies d'escala que sustenten els béns i serveis que consumim en el nostre dia a dia depenen de forma irremeiable de les cadenes de subministrament. Els problemes associats a aquestes i com solucionar-los són un dels punts més crítics per assegurar el bé futur del consumidor en un món digitalitzat.

En aquest treball s'analitzaran els problemes que presenten les cadenes de subministrament en el sector agroalimentari, i la possible solució tècnica mitjançant la implantació d'un llibre major distribuït que asseguri la integritat, disponibilitat i traçabilitat de les dades.

Paraules clau: DLT, Blockchain, Hyperledger, Cadenas de suministro

Resumen

Las economías de escala que sustentan los bienes y servicios que consumimos en nuestro día a día dependen de forma irremediable de las cadenas de suministro. Los problemas asociados a estas y como solucionarlos son uno de los puntos más críticos para asegurar el bien futuro del consumidor en un mundo digitalizado.

En este trabajo se analizarán los problemas que presentan las cadenas de suministro en el sector agroalimentario, y la posible solución técnica mediante la implantación de un libro mayor distribuido que asegure la integridad, disponibilidad y trazabilidad de los datos.

Palabras clave: DLT, Blockchain, Hyperledger, Cadenes de subministrament

Abstract

Economies of scale that sustain the goods and services we consume in our day-to-day lives are irretrievably dependent on supply chains. The problems associated with these and how to solve them are one of the most critical points to ensure the future good of the consumer in a digitized world.

This project will analyze the problems that supply chains present in the food sector, and the possible technical solution through the implementation of a distributed ledger that ensures the integrity, availability and traceability of the data.

Key words: DLT, Blockchain, Hyperledger, Supply Chain

Índice general

Índice general v
Índice de figuras vii

1 Introducción 1
1.1 Motivación . . . 2
1.2 Objetivos . . . 3
1.3 Estructura de la memoria . . . 3
2 Contexto tecnológico 5
2.1 Distributed Ledger Technology (DLT) . . . 5
2.2 Bitcoin y Blockchain . . . 6
2.2.1 Tipos de Blockchain . . . 9
2.3 Ethereum . . . 9
2.3.1 Aplicaciones descentralizadas . . . 10
2.4 Hyperledger . . . 12
2.5 Estado del arte . . . 13
3 Análisis del problema 15
3.1 Problemática en el sector agroalimentario . . . 15
3.2 Análisis de la seguridad . . . 16
4 Análisis de la solución 19
5 Diseño de la solución 21
5.1 Herramientas utilizadas . . . 21
5.1.1 VMWare Workstation Pro . . . 21
5.1.2 Docker . . . 21
5.1.3 Hyperledger Fabric . . . 22
5.1.4 NodeJS . . . 24
5.1.5 React . . . 24
5.1.6 Visual Studio Code . . . 25
5.1.7 Git . . . 25
5.2 Configuración del entorno . . . 25
5.3 Creación de la red . . . 26
5.4 Lógica de la red . . . 29
5.5 Creación de usuarios . . . 30
5.6 Despliegue de la API . . . 32
6 Pruebas 35
7 Conclusiones 39
Bibliografía 41

Apéndices
A Ficheros de configuración 43
A.1 crypto-config.yaml . . . 43
A.2 configtx.yaml . . . 43

B Lógica aplicación **47**
B.1 Chaincode 47
B.2 API 48

Índice de figuras

1.1 Esquema de una cadena de suministro 1
1.2 Tendencia de crecimiento en la cuota de mercado de distintos servicios logísticos 2

2.1 Diferencias entre un libro centralizado y un libro distribuido 6
2.2 Transacciones confirmadas por día 6
2.3 Relación entre los bloques de la Blockchain 7
2.4 Clave pública y clave privada 8
2.5 Cantidad de transacciones por día en la red Ethereum 9
2.6 Venta de propiedad automatizada con un contrato inteligente en la Blockchain de Ethereum 10
2.7 Diferencias entre aplicaciones web centralizadas y descentralizadas 11
2.8 Diferencias fundamentales entre los Tokens y Ethereum 12
2.9 Diferencias entre los tiempos de trazado gracias a Blockchain 13

3.1 Ciberataque en naviera Maersk 17

5.1 VMWare ejecutando varias máquinas virtuales 22
5.2 Estructura de Hyperledger Fabric 23
5.3 Funcionamiento asíncrono de NodeJS 24
5.4 Organizaciones de la red 26
5.5 Creación de certificados 28
5.6 Creación de bloque génesis y canal de comunicación 28
5.7 Despliegue de contenedores 29
5.8 Despliegue del contrato 31

6.1 El Proveedor añade carne a su stock 35
6.2 El Proveedor entrega la carne al repartidor 36
6.3 El repartidor indica que el producto ya está en sus almacenes 36
6.4 El repartidor consulta el estado del producto 36
6.5 El fabricante procesa las hamburguesas 37

CAPÍTULO 1
Introducción

El mundo hoy ofrece a sus habitantes infinidad de opciones para acceder a una gran cantidad de artículos. Uno puede, simplemente, andar hasta su supermercado más cercano y obtener aquello que desea consumir para el mismo día a un precio accesible. Somos capaces de hacer pedidos a países remotos de piezas y componentes que de otra forma serían inaccesibles. Podemos disponer de un amplio armario con multitud de piezas de ropa comprada a un precio ínfimo.

Esto es aun más impresionante cuando se tiene en cuenta la cantidad de procesos por la que estos artículos pasan para llegar al cliente final. Pongamos como ejemplo una camiseta, para su fabricación, una empresa debe adquirir los materiales a un tercero que distribuya los mismos. Seguidamente, la empresa debe disponer de los materiales adecuados para coser, dar forma e integrar el diseño en la correspondiente camiseta. Una vez ha sido fabricada, debe ser enviada a un centro de distribución que haga las veces de almacén para la misma, siendo distribuida de nuevo ante los pedidos de la tienda final.

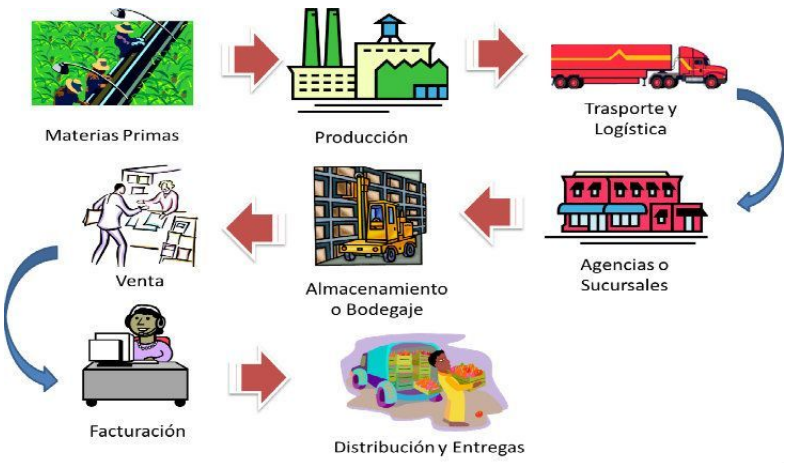

Figura 1.1: Esquema de una cadena de suministro

Solo en esta cadena, para una simple camiseta, podemos ver una gran cantidad de procesos y de actores que juegan un papel fundamental para que este producto llegue al consumidor final. Este entramado es conocido como cadena de suministro, y es el proceso más crítico que enfrentan las economías de escala.

El crecimiento y las necesidades logísticas tienen previsto un crecimiento lineal en los próximos años. La tendencia indica que cada vez más y más los propios usuarios realizarán todo tipo de pedidos directamente a su casa en mercados no tan explotados hoy día como el de los consumibles. Ello requiere construir cadenas de suministro resilientes para garantizar la integridad de los productos y la satisfacción de los clientes. Requiere

una profunda remodelación de como entendemos las cadenas de suministro y como se comunican los actores entre sí.

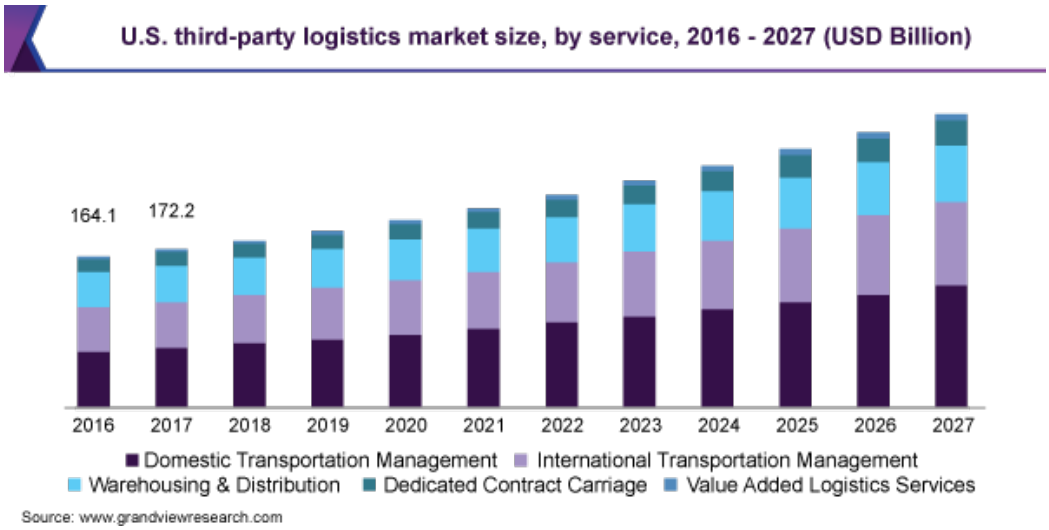

Figura 1.2: Tendencia de crecimiento en la cuota de mercado de distintos servicios logísticos

Las cadenas de suministro enfrentan un problema en su concepción, y es la cantidad de actores que juegan un papel y la confianza que se debe depositar en ellos. Una cadena de producción basa gran parte de su funcionamiento en la confianza que cada componente deposita en el resto de componentes de la cadena. Cuando un producto sale de tu almacén, un error humano puede provocar que este no se registre o no figure como entregado. Incluso un ataque externo podría inferir en la integridad de todos los datos que se almacenan en la plataforma, haciendo que estos pierdan valor y provocando un desastre en toda la cadena.

También se puede encontrar la confianza que hoy depositan ciertos agentes en un sistema centralizado que monitoriza el estado de los productos en una cadena de producción. Uno no solo tiene que confiar en el agente inmediatamente posterior, sino en el sistema que registra que el producto está en el punto que dice estar.

Todo esto provoca que, salvo en casos concretos, no exista un ente común en el que poder analizar la trayectoria del producto a lo largo de la cadena de producción. Hoy en día, las cadenas de producción son entidades completamente opacas para el exterior y, en muchos casos, no existe transparencia ni siquiera para los agentes inmediatamente posteriores y anteriores de la misma. Esto es especialmente crítico, debido a que estos agentes deben conocer la procedencia de su producto y el destino del mismo para asegurar la integridad del mismo frente al consumidor.

1.1 Motivación

Durante gran parte de mi etapa en la universidad he estado vinculado en mayor o menor medida a Blockchain. Desarrollé junto a un compañero una casa de cambio de criptomonedas durante las primeras etapas de mi formación académica.

Me fascinaba la capacidad que tenía esta tecnología para democratizar el acceso a procesos que hoy entendíamos como axiomas. El hecho de que una persona fuese capaz de realizar una transacción internacional sin la dependencia de un banco gracias a Bitcoin, o la capacidad que ofrecían algunos protocolos para comprar porciones de un activo (como una casa), disponiendo de un mercado secundario para la obtención de liquidez en

cualquier momento, me fascinó hasta tal punto que sabía que gran parte de mi carrera sería dedicada a ahondar y desarrollar para la expansión de servicios que hiciesen uso de esta tecnología.

Por eso, cuando se me planteó la realización de este trabajo en la oferta pública, tenía la certeza de que era aquello que más se asemejaba a la idea de trabajo académico que tenía en mente.

Quería tener la oportunidad de contribuir, analizar e investigar la realidad y la implantación que tenían actualmente los sistemas de trazabilidad basados en Blockchain en entornos reales, intentando crear una solución que fuese capaz de ser implantada en cualquier empresa para mejorar sus procesos.

1.2 Objetivos

Durante este proyecto, se tiene el objetivo de desarrollar un sistema de trazabilidad basado en DLT, que pueda servir de estándar como implementación de otras cadenas de suministro con necesidades estructurales más complejas.

En este caso concreto, se busca que:

- Existan distintas organizaciones dentro de la red descentralizada, cada una contando con permisos distintos y acceso a funciones únicas en función de las necesidades de su rol en la cadena de suministro.
- Se puedan añadir activos y que estos puedan ser convertidos en otros productos, representando estados concretos en una cadena de suministro.
- Cualquier agente de la cadena sea capaz de analizar y obtener un historial de estados por los que ha pasado el activo, incluyendo quién ha producido el activo, quien lo ha transformado en un bien distinto y en qué localizaciones ha residido el mismo.

1.3 Estructura de la memoria

A continuación, se introducirá como va a estructurarse la memoria, indicando los temas a tratar en cada uno de los puntos.

En el contexto tecnológico se tratará de adecuar al lector con los temas y tecnologías que se tratarán durante el desarrollo del trabajo. Se creará una visión global de la tecnología DLT, pasando por algunas de sus implementaciones más relevantes y se investigará su uso en entornos reales.

Tras conocer el entorno en el que se plantea el desarrollo de este trabajo, se analizará el problema principal que busca desarrollar este trabajo académico, incidiendo en los problemas de seguridad que pueden conllevar otras implementaciones en la actualidad bajo la perspectiva de un usuario y un regulador externo.

Una vez planteado el problema, se especificará por qué la solución que se propone durante el trabajo es la más adecuada, y que ventajas tiene con respecto a implementaciones actuales en el sector.

Se diseñará en detalle la solución, pasando por el necesario despliegue del entorno, las herramientas que se han usado y los pasos seguidos para su desarrollo.

Por último, se realizarán las pruebas necesarias para asegurar su funcionamiento y se expondrán las conclusiones finales del trabajo.

CAPÍTULO 2
Contexto tecnológico

En este capítulo se tratarán las tecnologías de obligado entendimiento para el completo análisis del trabajo. Es un punto especialmente relevante por lo complejo y alejado que puede encontrarse el lector de muchos de los términos que se usarán durante el desarrollo de la solución.

Durante el capítulo también se analizarán las diferentes implementaciones de la tecnología DLT, haciendo especial hincapié en Blockchain, por su relevancia, y diferentes protocolos que lo usan como base para su construcción.

2.1 Distributed Ledger Technology (DLT)

Para analizar y profundizar en la tecnología que fundamentará este trabajo académico es necesario entender las bases de su funcionamiento.

Cuando se abarca la tecnología DLT, existe un fallo generalizado de concepción. Muchos de los artículos que se pueden encontrar en Internet clasifican de igual forma la tecnología Blockchain (que se abarcará en el siguiente punto de este trabajo) y DLT. Ello es porque Blockchain, con sus diferentes iteraciones e implementaciones, ha resultado ser la punta de lanza de la tecnología DLT, asentándose como la referencia cuando se habla de la misma. Si bien es cierto que no es erróneo afirmar que Blockchain es una DLT, Blockchain solo es una implementación de la tecnología.

La definición de DLT bajo la que profundizaremos en este trabajo es "DLT se refiere a los procesos y tecnologías relacionadas que permiten a los nodos en una red proponer, validar y registrar cambios de estado en un libro mayor sincronizado, distribuido a través de los nodos de la red." [1]

El libro mayor distribuido es el lugar en el que se perfilan los registros que cada nodo propone en la red. Este está replicado en cada uno de los nodos participantes, evitando así que los datos se vean comprometidos en caso de fallo en alguno de los nodos.

Un nodo es la unidad de cómputo básica en una red. En un libro mayor distribuido, todos los nodos registran cambios mediante consenso. Gracias a este planteamiento distribuido y a la capacidad de decisión de los nodos, se consigue "democratizar" el acceso a ciertos servicios y aplicaciones, ya que las transacciones en la red no necesitan del permiso o justificación de un ente central. Otorga seguridad a los registros en el libro distribuido debido al poder de decisión de los nodos participantes, eliminando la centralización y los puntos de fallo que se encuentran en un sistema centralizado. Esta seguridad, evidentemente, aumenta cuanto mayor sea el número de nodos presente en la

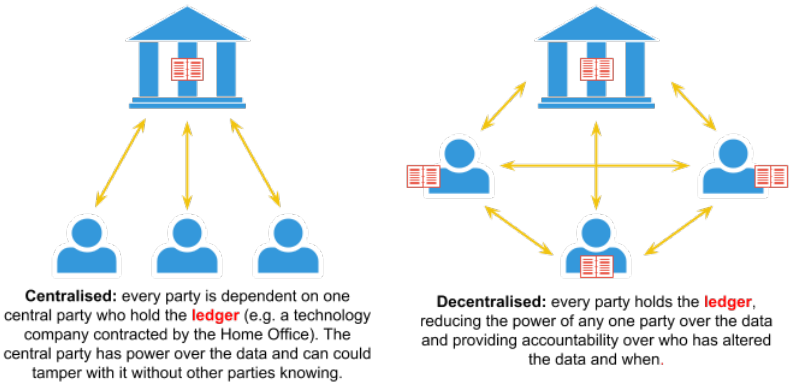

Figura 2.1: Diferencias entre un libro centralizado y un libro distribuido

red, evitando así que un actor o la suma de unos pocos actores puedan tomar decisiones en contra del resto.

Al ser DLT un término general, es complicado entender y ahondar en los diferentes mecanismos que se usan para implementar estos estándares. En los siguientes apartados se conseguirá tener una visión global de la tecnología mediante las distintas implementaciones que tienen relevancia en la actualidad.

2.2 Bitcoin y Blockchain

Bitcoin es, además del primer tipo de Blokchain (y DLT) funcional, la implantación que ha cosechado mayores logros y repercusión. Es una tecnología que puede resultar compleja la primera vez que uno se acerca a ella por lo paradójico de muchos de sus planteamientos. Creo que un acercamiento exitoso a la tecnología Blockchain pasa por entender qué aporta a su entorno y por qué la base de su funcionamiento ha sido tan disruptiva. El ejemplo más evidente de su auge es el número de transacciones confirmadas por día, cuya tendencia es claramente ascendente.

Figura 2.2: Transacciones confirmadas por día

Muchas de las bases de su necesidad, también aplicables a casi todos los sectores, pasan por la confianza. Esto es ampliamente mencionado en el texto original de Bitcoin,

escrito por su fundador, Satoshi Nakamoto (una figura de la cual conocemos pocos detalles más allá de su nombre, obviando especulaciones):

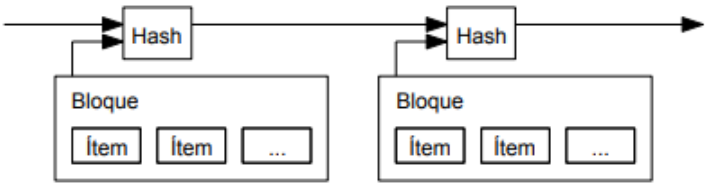

Figura 2.3: Relación entre los bloques de la Blockchain

"El comercio en Internet ha llegado a depender casi exclusivamente de las instituciones financieras como terceros de confianza en el proceso de los pagos electrónicos. A pesar de que el sistema funciona suficientemente bien en la mayor parte de las transacciones, sufre la debilidad inherente al modelo basado en confianza.(...) La solución que proponemos comienza con un servidor de sellado de tiempo. Un servidor de sellado de tiempo trabaja tomando el *hash* de un bloque de ítems para sellarlos en el tiempo y notificar públicamente su *hash* [1], como un periódico o un post Usenet [2-5]. El sellado de tiempo prueba que los datos han existido en el tiempo, obviamente, para entrar en el *hash*. Cada sellado de tiempo incluye el sellado de tiempo previo en su *hash*, formando una cadena, con cada sellado de tiempo adicional reforzando al que estaba antes." [2]

Como se puede observar, se propone como solución a la confianza inherente que se deposita sobre los agentes, el crear un sistema que permita acreditar la autoría de cada moneda de forma en que no se deposite confianza sobre un tercero, sino que la confianza se deposite sobre la propia tecnología. Considero que es un concepto clave para entender la necesidad de Bitcoin y por qué las alternativas que propone son tan disruptivas.

El concepto base sobre el que se organiza Bitcoin es un libro gigante distribuido y replicado entre una gran cantidad de nodos. Este libro está dividido en bloques, de forma que en cada bloque quedan registradas las transferencias que se han realizado en un margen de tiempo. Estos bloques, a su vez, están conectados entre sí con un identificador único llamado *hash*. Cuando un bloque se genera, el *hash* del bloque está unido al *hash* del bloque anterior, de forma que si un nodo plantease un cambio en un bloque anterior de la cadena, todos los bloques posteriores deberían recalcular su *hash*, suponiendo un coste computacional tan grande que hace que se trate a la información registrada en la cadena como inmutable.

Cada transferencia necesita de la aprobación de los nodos de la red para ser introducida en un bloque. Es bastante sencillo entender como funciona una transacción con este texto extraído uno de los sitios web más importantes de Bitcoin:

"Hay un retraso de 10 minutos de media antes de que la red empiece a confirmar una transacción al incluirla en un bloque. Una confirmación significa que hay un consenso en la red en que los bitcoins recibidos no han sido enviados a alguien más y son ahora de tu propiedad. Una vez que tu transacción ha sido incluida en un bloque, esta irá siendo "enterrada" con más confirmaciones por los siguientes bloques que van añadiéndose a la cadena, lo que hará consolidarse este consenso y disminuir el riesgo de una revocar la transacción" [3]

Los monederos usan criptografía asimétrica para su funcionamiento. Constan de una clave pública y una clave privada.

[1] https://es.wikipedia.org/wiki/Funcion_hash

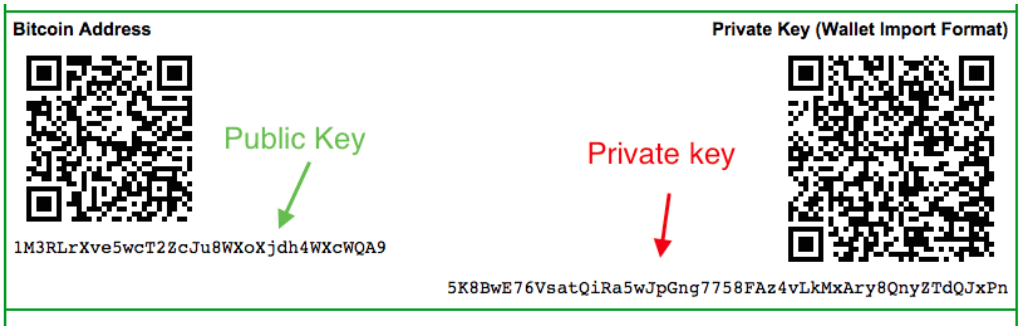

Figura 2.4: Clave pública y clave privada

La clave privada es la encargada de firmar las transacciones y tener posesión de la misma te autoriza a usar los Bitcoin contenidos en una dirección. Asimismo, la clave pública asociada a esa clave privada puede ser expuesta al mundo sin que esta pueda comprometer de forma alguna el contenido de dicho monedero. Para hacer un símil, la clave pública sería la cuenta bancaria de una persona; una persona puede enviar a cualquier parte su cuenta bancaria si desea recibir un pago en dicha cuenta, pero de ningún modo se podría acceder a esta cuenta bancaria sin la contraseña. La contraseña, en este caso, sería la clave privada, que firmaría las transacciones y permitiría acceder al monedero.

Existe una cantidad de Bitcoin limitada a 21 millones que aún no ha sido alcanzada. Cuando se genera un bloque, un nodo minero trata de resolver un complejo algoritmo criptográfico con el objetivo de obtener el *hash* del bloque. Cuando este es obtenido, se añade a la cadena y el nodo que ha conseguido resolverlo es recompensado con una cantidad de Bitcoin variable. Es la forma que tiene la red de poner más Bitcoin en circulación

Tras esta introducción, hemos repasado las tres características principales de Bitcoin:

- Es pseudoanónimo. Una cantidad de bitcoin solo es asociada a una dirección. Al estar todas las transacciones disponibles de forma pública en los distintos exploradores de la cadena de bloques, se consigue transparencia, pero nadie podrá relacionar una cantidad de bitcoin con su dueño mientras que el dueño no reconozca que esa es su dirección.

- Es inmutable. Una vez una transacción ha quedado registrado en un bloque, se puede considerar que esta transacción es inmutable en el tiempo por la relación que guardan los bloques entre sí. Esto hace que sea fácilmente auditable y uno sea capaz de encontrar en qué momento se ha realizado una transacción.

- No requiere de confianza en un tercero. Todas las operaciones que se realizan en la red dependen únicamente de la tecnología y los mecanismos de consenso que se usan para salvaguardar la fiabilidad de los bloques introducidos. No hay ningún ente dominante ni existe un agente que pueda influir en el desarrollo de la moneda.

Pero sin duda, la característica más disruptiva de Bitcoin fue la creación de la tecnología Blockchain, la base de su funcionamiento y sin la cual no sería posible entender Bitcoin. Han surgido gran cantidad de alternativas e integraciones distintas a esta Blockchain a raíz de su creación, pero si algo se le debe a la moneda digital es poner en valor el papel fundamental que juega esta tecnología en la sociedad del futuro.

2.2.1. Tipos de Blockchain

Entendido el concepto de Blockchain, existen en la actualidad bastantes acercamientos [5] a la implantación de la misma. Si intentamos tipificarlos, se puede llegar al consenso de que existen 3 grandes grupos:

- Blockchains públicas. Es la Blockchain que hemos presentado en el punto anterior, en ella no existe ningún participante con mayor poder que el resto. Es el acercamiento más puro a la Blockchain original de Bitcoin y, para muchos, el único modelo de Blockchain que merece ser llamado así por ser la única que salvaguarda sus tres principios básicos: pseudo-anonimato, inmutabilidad y la no confianza en un tercero.
- Blokchains privadas. Es el concepto más alejado a la Blockchain de Bitcoin de las tres. En ella el poder reside sobre un único ente central que se encarga de dar poder a los distintos usuarios. Algunos ejemplos pueden ser monedas como Ripple o Quorum, siendo eminentemente usada en el sector financiero.
- Blockchains híbridas o permisionadas. Es un concepto híbrido, usualmente asociado a consorcios de distintas empresas que se unen para crear una infraestructura común. En ella el poder no recae sobre un solo actor ya que todos ellos colaboran para facilitar el funcionamiento de la red, otorgando ciertos permisos a ciertos agentes o usuarios. Uno de los ejemplos más destacados es el caso de Hyperledger, el cual trataremos en siguientes apartados.

2.3 Ethereum

Gracias al auge de Bitcoin como protocolo y el nacimiento del término Blockchain, empiezan a surgir otras alternativas que permiten aprovechar la tecnología que sustenta la moneda digital para darle profundidad y nuevas opciones de uso. Una de las más disruptivas en su momento, y aún relevante a día de hoy, es Ethereum. De nuevo, como en el caso anterior, basta con echar un vistazo al número de transacciones por día (que es una medida mucho más efectiva que el precio para analizar su crecimiento) para observar que tiene una tendencia claramente alzista desde su lanzamiento en 2015:

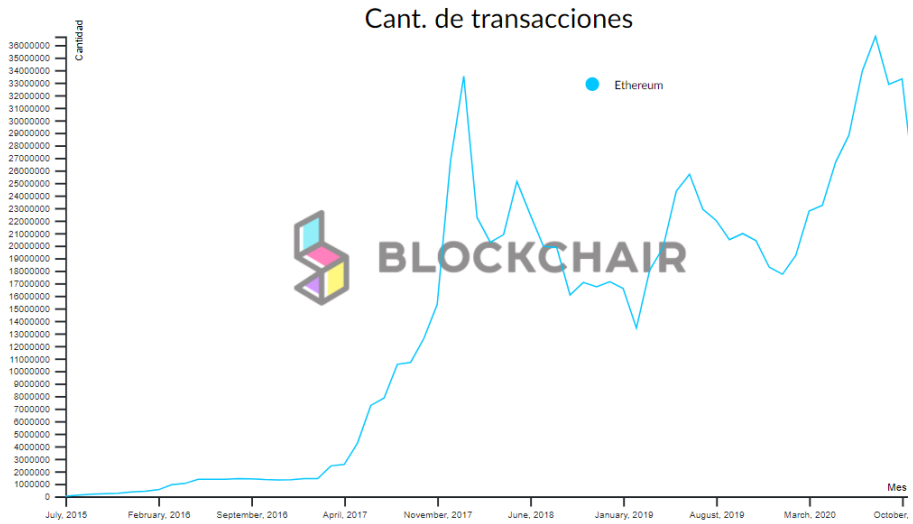

Figura 2.5: Cantidad de transacciones por día en la red Ethereum

Ethereum aprovecha y reusa gran parte de la estructura de una cadena de bloques pero introduce un nuevo concepto llamado ”contrato inteligente”.

En el propio sitio web de Ethereum definen un contrato inteligente como ün fragmento de código de programación que se ejecuta en Ethereum. Se llama çontrato"porque el código que se ejecuta en Ethereum puede controlar cosas valiosas como ETH u otros activos digitales"[4]. Es decir, se trata de un programa que puede ser interactuado mediante el uso de *Ethers*, la criptomoneda de la Blockchain de Ethereum, y que se ejecuta cuando se dan ciertas condiciones.

En el ejemplo que podemos ver en la Figura 2.6 se puede ver reflejado el funcionamiento de un contrato inteligente. En él, un vendedor quiere deshacerse de una casa, mientras que un comprador está buscando una propiedad. El vendedor hace una transacción de su bien al contrato inteligente y el comprador envía la cantidad acordada al contrato. Una vez se cumplen las dos condiciones, el dinero es recibido por el vendedor y la propiedad cambia de dueño al comprador. El contrato inteligente permite automatizar esta operación y permitir que vendedor y comprador sepan de antemano que condiciones se deben dar sin necesidad de confiar el uno en el otro para llevar a cabo la operación.

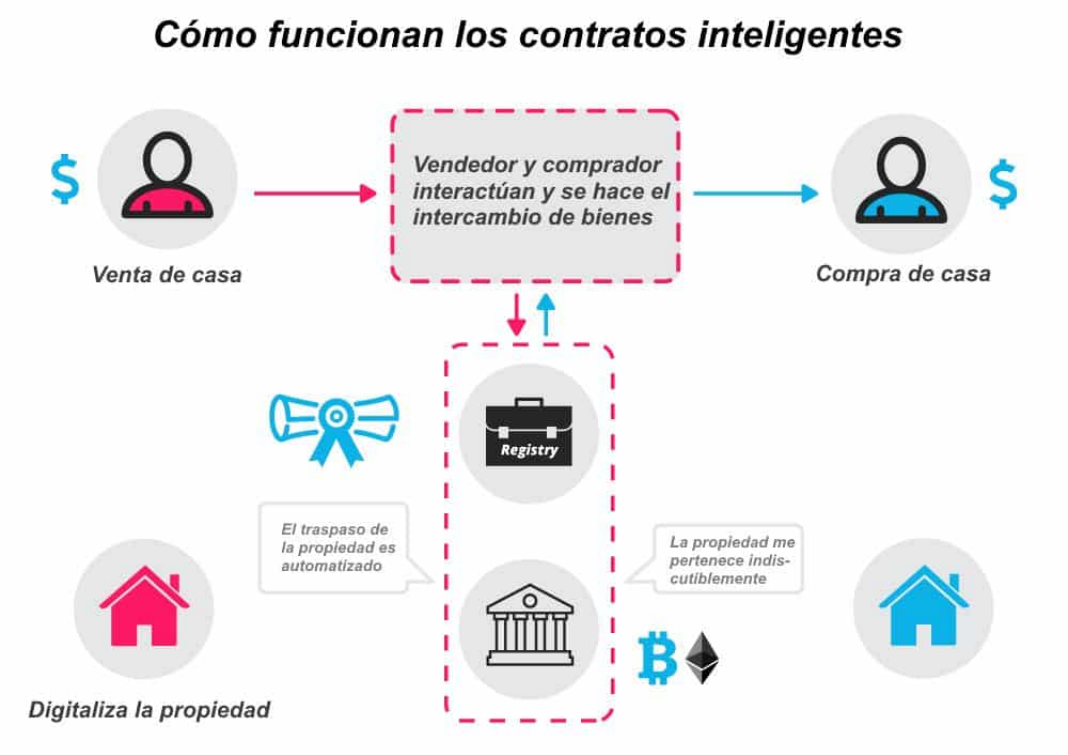

Figura 2.6: Venta de propiedad automatizada con un contrato inteligente en la Blockchain de Ethereum
2

2.3.1. Aplicaciones descentralizadas

Gracias al concepto de contrato inteligente recién introducido se genera un nuevo elemento clave en la red Ethereum. Las aplicaciones descentralizadas.

Los contratos inteligentes, al ser piezas de código que se ejecutan en la red Ethereum bajo ciertas condiciones, otorgan un entorno perfecto para actuar como base de un nuevo tipo de aplicación.

[2]https://ethereum.org/en/

Las aplicaciones tradicionales, en muchos casos, guardan un problema fundamental en el punto en que en muchas de ellas se han establecido como un punto fundamental de la sociedad actual. Herramientas tan críticas y populares como una red social, las cuales tienen inferencia directa en la opinión pública, están en manos de grandes organizaciones que tienen en su poder el control del mensaje que llega al mundo. La capacidad que tienen empresas de tal calibre deja en una posición endeble a sus usuarios, ya que los procesos que se siguen para moderar la aplicación son en muchos casos opacos para el exterior.

Una aplicación web tradicional tiene una estructura clásica, en la que una capa se renderiza del lado del cliente, teniendo cierta lógica y todo el diseño, mientras que el groso de la aplicación se encuentra del lado del servidor, en el *back-end*, siendo una incógnita como trabaja el mismo para el usuario.

Una de las partes fundamentales de la red Ethereum, y que a su vez permiten la existencia de muchas aplicaciones descentralizadas, son los Tokens. Los tokens actúan como moneda de cambio en muchas de estas aplicaciones, mientras que en otros casos representan una fuente de ingresos para los equipos tras el desarrollo de estas aplicaciones.

A fin de cuentas, el principal valor que aporta un Token es el de digitalizar un activo que existe en el exterior. Para entender el concepto, una empresa intenta recaudar fondos para llevar adelante un proyecto. Esta empresa no quiere pasar por los habituales canales para buscar financiación, como pueden ser los "Business Angels" o los fondos de capital riesgo, debido a que ellos les obligan a ceder una parte importante de la participación de la empresa. Ethereum plantea una alternativa a esta problemática con un concepto conocido como ICO (Initial Coin Offering), en que se ofrece una cantidad limitada de los Tokens que habrá en circulación. Al acceder a la oferta inicial de la moneda, uno puede invertir en la idea o concepto que la empresa intenta desarrollar. Al adquirir el Token, un "smart contract" se encarga automáticamente de hacer la conversión y depositar los tokens en la dirección del comprador, mientras que los Ether con los que son adquiridos pasan a la empresa que emite la moneda.

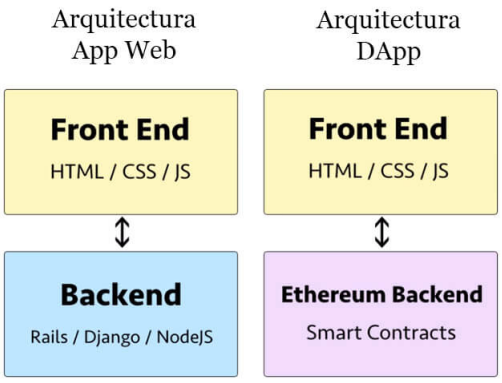

Figura 2.7: Diferencias entre aplicaciones web centralizadas y descentralizadas


	ethereum	ethereum tokens
concept	smart contracts platform	digital assets on top of ethereum
market cap (as of may 2017)	~$17 billion	~$1.5 billion
native currency	ether	augur (rep), golem (gnt), aragon (ant), & many more
founder	vitalik buterin and team	varies by project
release method	presale raised $18M in bitcoin	typically through crowd sales

Figura 2.8: Diferencias fundamentales entre los Tokens y Ethereum

Esto plantea dos escenarios para clasificar los Tokens que una empresa emite:

- **Utility Tokens.** Son aquellos Tokens que tienen una utilidad marcada en la plataforma, por tanto su interés para el comprador es el servir de puerta de entrada a los servicios que ofrecerá la empresa a un precio presumiblemente inferior del que alcanzará cuando sea lanzada.
- **Security Tokens.** Son Tokens que son clasificados como inversión, no guardan una utilidad real en la plataforma más que reportar unos dividendos por las operaciones en la misma. Suelen ser adquiridos como mera inversión. [6]

2.4 Hyperledger

Hyperledger se separa de forma bastante contundente de las implementaciones de Blockchain tratadas hasta ahora.

Se trata de un proyecto orientado a la creación de herramientas, librerías y aplicaciones para el despliegue de Blockchains orientadas al mundo empresarial. [7]

Para desarrollar este concepto, mantiene en activo distintos proyectos, entre los cuales, se pueden destacar los siguientes:

- **Hyperledger Fabric**: Es la implementación que fundamentará este trabajo, permite el despliegue de blockhains permisionadas mediante una serie de herramientas. Se tratará en detalle en el Diseño de la solución.
- **Hyperledger Explorer**: Aplicación que dota de una capa externa a los despliegues de Blockchains permisionadas. Presenta una interfaz web que permite analizar y extraer los datos que se encuentran en la cadena de bloques de forma gráfica.
- **Hyperledger Grid**: Plantea herramientas que ayudan a fundamentar la implementación de todo aquello relacionado con las cadenas de suministro en una red descentralizada. No se ha usado en este trabajo debido a lo poco maduro que se encuentra el proyecto actualmente.
- **Hyperledger Indy**: Otorga todo tipo de componentes que permiten la creación de un sistema de identidad digital usando la estructura de la cadena de bloques.
- **Hyperledger Indy**: Herramienta que diseñada para permitir la exitosa integración de diferentes Blockchains.

2.5 Estado del arte

En este punto trataremos algunos de los casos reales en los que se pueden ver implantaciones de redes DLT para sistemas de trazabilidad.

La reconocida cadena de hipermecados estadounidenses Walmart, trata la cadena de suministro no como una cadena al uso, sino como una red mucho más compleja. Tanto es así que Frank Yiannas, el antiguo vice-presidente de la compañía, reconocía que a veces podía "tardar días, incluso semanas, hallar la fuente de un brote de productos defectuosos"[8]. Esto provoca que los gobiernos acaben por recomendar incluso que se deje de consumir un producto concreto, lo que afecta negativamente a su reputación y puede hacer que hasta se deje de consumir durante un largo período de tiempo, mas allá del brote. Walmart, debido a su posición en el mercado, buscaba añadir transparencia en los datos a fin de mejorar la trazabilidad de sus productos.

Para analizar el tiempo que se necesitaba para trazar de donde provenían los productos, se hizo un modelo de análisis con los mangos vendidos por las tiendas Walmart estadounidenses. Se compraron unos mangos y se pidió al equipo que analizará de que granja provenían estos. Al no existir conexiones, se tardó 7 días en obtener una respuesta.

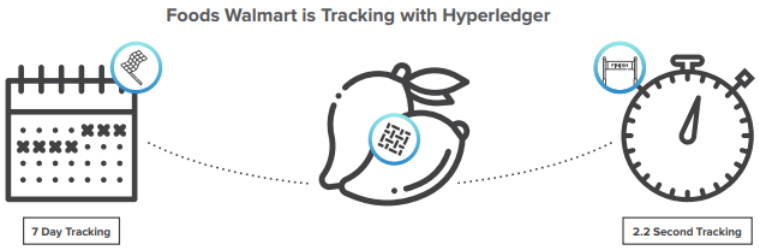

Figura 2.9: Diferencias entre los tiempos de trazado gracias a Blockchain
[3]

Para la prueba de concepto en la implantación, se desarrolló de forma exitosa una implantación de Hyperledger Fabric, una Blockchain permisionada, en la infraestructura de la empresa. Gracias a la colaboración con IBM, que desarrolló la lógica interna de la red, se consiguió pasar de un tiempo de análisis de 7 días a uno de tan solo 2.2 segundos.

Debido a la exitosa implantación de este modelo, Walmart decidió expandirlo a más productos llegando hasta un total de 25. Se ha convertido en un punto tan crítico para la compañía que ahora cualquier distribuidor de alimentos frescos debe ser parte de esta red si quiere suministrar alimentos a Walmart.

Supone una profunda transformación de uno de los grandes exponentes de economías de escala, adaptando por completo su modelo para entregar más transparencia al consumidor y a los analistas internos.

Otro de los ejemplos más exitosos de implementaciones de DLT en un entorno real es el de la empresa *Change Healthcare*[9]. Es una empresa que ha procesado más de 14 mil millones de transacciones de datos de salud.

Su misión principal es la de modernizar el sistema de salud Americano, que, según sus palabras "se encuentra plagado de ineficiencia, fraude y desperdicio". Considera que un sistema unificado permitiría ahorrar al estado 450 mil millones de dólares que se invierten en adaptar todas las tecnologías que conviven las unas con las otras.

[3] https://www.hyperledger.org/wp-content/uploads/2019/02/HyperledgerCaseStudyWalmartPrintableV4.pdf

Tomaron la decisión así de construir un consorcio de empresas basado en Hyperledger Fabric, que buscaba ser un estándar para que el resto de empresas se unieran y colaboraran con el fin de abaratar costes globales y mejorar la economía del país. Sabían que era muy difícil encontrar una solución privada que convenciese al resto de empresas, por eso Hyperledger Fabric, software libre, se consolidaba como la mejor opción.

Se construyó una solución desplegada en un entorno de producción en tan solo unos meses, permitiendo cerca de 550 transacciones por segundo. Se trata del primer consorcio empresarial de gran escala para el sector de la salud.

Representa, junto al caso de Walmart, implementaciones de una tecnología que representa una oportunidad real y, en muchos casos, la mejor de las alternativas para el despliegue de infraestructuras que necesitan de datos inmutables, trazables e íntegros.

CAPÍTULO 3
Análisis del problema

Una vez repasado el contexto tecnológico y las implicaciones que este ha tenido sobre varios sectores, vamos a introducir el problema principal en un ámbito concreto, las cadenas de suministro en el sector agroalimentario.

3.1 Problemática en el sector agroalimentario

La trazabilidad de alimentos a lo largo de una cadena de suministro es un requisito legal que es exigido a todos los actores de la cadena. Es crítico y necesario para asegurar el origen de los productos, así como el estado y la procedencia de los mismos. A pesar de ello, la información no está directamente conectada entre los distintos agentes, lo que provoca y dificulta la transparencia tanto a la hora de asegurar el origen de un producto por parte de un consumidor, como de asegurar el origen de un producto obtenido de un proveedor por parte de un fabricante.

Actualmente, los estándares en la trazabilidad de alimentos están definidos por el "JOINT FAO/WHO Food Standards Programme", e indican que un agente de la cadena de suministro debe poder conocer con exactitud la procedencia y el destino de un producto con un nivel de diferencia. Es decir, el agente en cuestión debe conocer la procedencia del producto, identificando cuál es el agente anterior a él, y el destino de un producto, conociendo el agente inmediatamente posterior al mismo. [10]

Esto resulta un problema en casos en los que se plantean productos muy complejos, como cereales que están fabricados con distintos ingredientes y complejas cadenas de suministro para proveer los mismos, o productos que son fusionados con materias primas provenientes de diferentes granjas, como en el caso de la leche. Ante esta complejidad, en cientos de casos que cuentan con tecnologías y recursos ciertamente arcaicos, es prácticamente imposible asegurar la procedencia y el destino de los productos debido a la poca transparencia y a la falta de unión de los distintos agentes de la cadena.

El problema entonces, radica en la falta de un estándar o gran consorcio que permita analizar a gran escala, y con transparencia total, la procedencia y los actores que han intervenido en la manipulación de los distintos productos.

Además del problema base, existen otros problemas que derivan de la problemática inicial. Los actores deben confiar los unos en los otros cuando establecen ciertas normas de cumplimiento. Esto, que es un problema intrínseco a casi cualquier relación humana, supone una vulnerabilidad crítica de cara a la fiabilidad de los propios datos.

Los actores, debido a la nula conexión global que existe entre ellos, deben confiar en que la palabra del agente anterior sobre la procedencia de un producto es acertada. Esto

es, como en cualquier relación basada en la confianza, un punto claro sobre el que se pueden fundamentar estafas y fraude. Eliminar la confianza entre actores de la ecuación puede permitir que los participantes en la cadena sepan que los datos que reciben son exactos.

Existen distintos acercamientos hacia cual es el tiempo que deben mantener los datos relativos a los productos vendidos o generados en las distintas empresas. Uno de las métricas más cercanas a lo que se aplica en el mundo real oscila entre los tres años. Supone crítico para una empresa tener que mantener estos datos durante 3 años, porque obliga a mantener una infraestructura que asegure que esos datos no se perderán ni serán corruptos en ningún momento. Es un problema que trataremos en detalle en el siguiente punto, el análisis de la seguridad, pero que no se puede omitir debido a que afecta de forma directa a la hora de plantear una cadena de suministro. [11]

3.2 Análisis de la seguridad

Los datos que se representan en los sistemas de control de cada agente son críticos. La dependencia entre unos y otros agentes hacen que mantener estos datos sea necesario no solo para salvaguardar el origen de los productos de un agente, sino también para permitir que agentes externos como un auditor puedan asegurar la integridad de estos datos en un momento dado.

Muchos de los agentes de la cadena, como se ha mencionado, carecen de recursos para asegurar la seguridad de estos datos, siendo especialmente vulnerables a ataques que puedan comprometer su integridad. Que un atacante externo pueda cambiar la procedencia de un producto o la fecha en que este fue distribuido puede cambiar por completo la versión que un investigador externo pueda tener sobre un fraude cometido en un momento dado.

Los ataques que puede recibir el software de una cadena pueden tener mayor impacto por la incapacidad de detección con la que cuentan empresas menos digitalizadas. Si una empresa está usando un software de terceros, sin actualizar ni mantener, es posible que esté expuesto a vulnerabilidades conocidas que puedan suponer un peligro para la integridad de los datos; este hecho se agrava cuando una organización no tiene mecanismos de detección adecuados que permitan conocer que existe un atacante dento de la organización accediendo y modificando a su antojo datos e información crítica para el consumidor y el resto de agentes.´

Un ciberataque puede resultar más devastador en una cadena de suministro porque, por su concepción hoy en día, los datos suelen estar centralizados en un solo agente, suponiendo así un punto débil para toda la cadena. Si un grupo ciberterrorista es capaz de comprometer uno de los agentes y eliminar todos sus datos, el sistema se queda cojo y todos los demás agentes que dependen de él tienen una incapacidad operativa real.

Para ejemplificar con un ataque real, se puede relatar el caso del ciberataque sobre la naviera Maersk. Resulta ser uno de los conglomerados marítimos más grandes del mundo, distribuyendo una gran cantidad de productos y siendo una pieza base para millones de cadenas de suministro. Un grupo ciberterrorista fue capaz de introducir un ransomware en el sistema conocido como Petya, el cual fue capaz de replicarse usando vulnerabilidades de día cero por todo el espectro de equipos de la organización. Una vez este fue ejecutado, no hubo vuelta atrás para Maersk, toda su información, los equipos con los que operaba y todos los pedidos en curso quedaron comprometidos. "Vi una gran cantidad de pantallas volviéndose a negro. Negro. Negro. Negro. Negro en todas partes.- elata uno de sus empleados que presenció el ataque.

Este ataque conmocionó al mundo entero y puso en jaque casi todas las cadenas de suministro del mundo. Se estima que FedEx necesitó bastantes meses para recuperar los datos perdidos, los inversores perdieron cerca de 870 millones de dólares y otros agentes, incluso de sectores como la construcción, llegaron a perder casi 188 millones de dólares. Maersk como entidad estima en 250 millones de dólares las pérdidas por el ciberataque. [12]

Figura 3.1: Ciberataque en naviera Maersk

Si esto ha ocurrido sobre un conglomerado tan grande como el de Maersk, ¿qué no podría ocurrir sobre empresas de escala mucho menor?

Resulta clave para el futuro digital de las cadenas de suministro estandarizar protocolos que aseguren la integridad de los datos, su disponibilidad a largo plazo y la autoría certificada de que ha sido una organización concreta la que ha introducido estos datos. Solo construyendo unos pilares resistentes se podrá asegurar que los productos puedan cumplir ciertos requisitos en un mundo digital y global.

CAPÍTULO 4
Análisis de la solución

Se han planteado los problemas que afectan a las cadenas de suministro en el sector agroalimentario. Se ha podido analizar que se trata de un problema complejo; no es sencillo construir un sistema que asegure la trazabilidad de los datos en todos los puntos de la cadena, asegurando además la integridad de estos mismos datos. Es un problema que, de base, no plantea soluciones a medias, puesto que se interfiere sobre sectores muy críticos.

Durante este trabajo, se va a plantear como solución una tecnología alejada de los medios tradicionales, el uso de una Blockchain permisionada (concepto que se introdujo en el contexto tecnológico).

Existen muchos detractores de las Blockchains permisionadas, de hecho, si se busca información en internet, existen muchos usuarios que realizan críticas voraces al concepto porque creen que se aleja demasiado de la idea original de Blockchain en sí. Esta tecnología permite la construcción de redes descentralizadas en las que existe un libro mayor replicado en todos los nodos de la red y cuya información está distribuida en bloques. La diferencia principal, y por la que surgen muchos detractores, es que se pueden otorgar permisos a los participantes de la cadena. Esto, que puede suponer en otros casos una desventaja, permite en esta problemática una oportunidad para la construcción de redes que requieren de complejidades estructurales y condiciones especiales. Permite asignar permisos y poderes solo a los agentes que deben hacer ciertas funciones. De esta forma, se pueden cubrir los requisitos logísticos que muchas de estas cadenas plantean.

Supone una solución a muchos de los problemas de la cadena, porque al replicar los datos entre todos los integrantes de la cadena, cualquiera de los participantes puede realizar consultas para ver el historial de un producto concreto en un momento determinado. Gracias a ser capaces de analizar el historial a lo largo de los miembros de este consorcio se puede obtener una visión real del origen de los productos trabajados. Soluciona el problema al que se enfrentaban aquellos productos que por concepción se hacían bajo la suma de materias primas que provenían de gran cantidad de organizaciones. Se consigue unificar todos los agentes que trabajan de forma conjunta en una sola red sobre la que cada empresa puede construir o basar sus aplicaciones de forma independiente.

La red permite implementar una lógica interna, similar a lo visto en los "contratos inteligentes" de Ethereum, por lo que se pueden establecer de forma explícita las condiciones que tienen que darse para casos concretos del mundo real, como una compra al proveedor estableciendo un precio concreto, bajo el que no caben negociaciones. Esto permite evitar que los miembros tengan que confiar los unos en los otros ya que las condiciones para que se den los cambios de estado en los productos de la cadena están definidas de forma intrínseca en el propio contrato que los actores aceptan al entrar a la red. Si los datos son íntegros y trazables, y las condiciones son explícitas y autoverificadas

por la red, no cabe duda a la hora de plantear operaciones entre empresas y se elimina gran parte del fraude y la estafa asociados a muchas cadenas de suministro.

Evita que existan puntos críticos en el despliegue, ya que la retirada de una organización del consorcio no supone la pérdida de los datos asociadas a ella, esto ayuda a que los auditores externos sean capaces de analizar con certeza cada movimiento de un producto. Además, permite a las empresas no tener que preocuparse por un ataque crítico en cualquiera de los miembros, ya que su información está salvaguardada en el resto.

Por su carácter de estándar, permite añadir de forma regular nuevas empresas al consorcio en función de las necesidades que tenga la cadena, obteniendo así una forma fácil de que los nuevos miembros se adapten y empiecen a trabajar. Esto es crítico, ya que elimina el tener que convivir con ciertos de protocolos e implementaciones distintas, en los que cada agente está aislado del resto porque no existe forma viable de unirlos a todos.

Para la implementación de un modelo que cumpla con las condiciones que se plantean en esta solución, se va a usar Hyperledger Fabric. Es una de aplicaciones de Hyperledger que está orientada a la creación de redes permisionadas, y se analizará en más detalle en el siguiente apartado, el diseño de la solución.

CAPÍTULO 5
Diseño de la solución

Durante este apartado, se analizarán las herramientas utilizadas para dar lugar a la solución planteada en el apartado anterior, así como los pasos seguidos para su desarrollo y despliegue.

5.1 Herramientas utilizadas

5.1.1. VMWare Workstation Pro

VMWare [1] es uno de los componentes básicos que ha permitido el desarrollo de este proyecto. Permite la virtualización de sistemas operativos.

VMWare hace uso de un uso de un equipo anfitrión al que se conoce como Host, en él ejecuta máquinas virtuales que permiten la ejecución de otros sistemas operativos asignándole una cantidad de recursos concretos.

Ello permite que un equipo Windows pueda estar ejecutando distintas distribuciones Linux al mismo tiempo sin que exista fricción entre ellas. Permite desplegar una compleja red de servidores en tu propio equipo, recrear entornos complejos para probar ataques informáticos o acercamientos más simples como el de utilizar aplicaciones que no están disponibles para el sistema operativo del Host.

Cuenta con un cómodo menú que permite ejecutar varias máquinas virtuales al mismo tiempo y moverse entre ellas de forma sencilla. Además, incluye distintas funcionalidades que hacen la vida más fácil para cualquier desarrollador como la posibilidad de copiar y pegar archivos y texto directamente en la máquina virtual desde el host y viceversa.

5.1.2. Docker

Docker [2] nace con la idea de crear contenedores ligeros y portables para la ejecución de aplicaciones de software en cualquier dispositivo que cuente con Docker instalado, independientemente del sistema operativo que esté usando. Permite por ello rápidos despliegues de aplicaciones, ya que todo lo necesario para su funcionamiento se encuentra autocontenido en el propio contenedor.

Docker permite a un desarrollador introducir dentro de un contenedor todo complemento requerido para el funcionamiento de su aplicación, de forma que no necesita

[1] https://www.vmware.com/es.html
[2] https://www.docker.com/

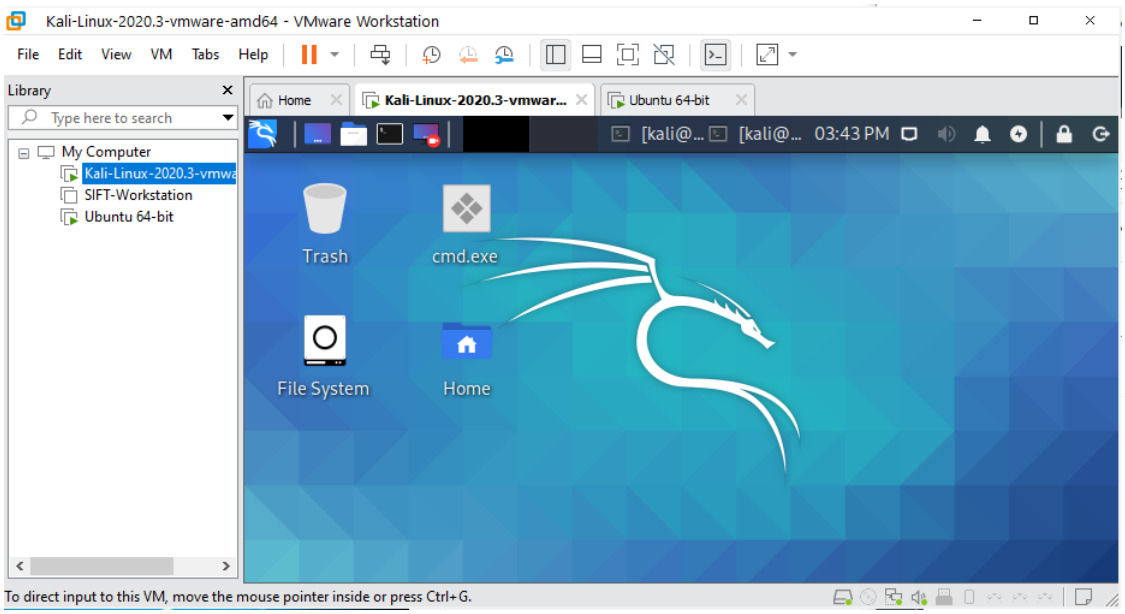

Figura 5.1: VMWare ejecutando varias máquinas virtuales

preocuparse por la infraestructura en la que trabajará ese contenedor, si tendrá los complementos adecuados o si necesitará alguna librería externa con la que no cuenta.

Esto aporta, tanto a desarrolladores como a usuarios, una facilidad de uso inmensa, ya que el usuario tan solo tiene que desplegar el contenedor en caso de querer usar la aplicación y el desarrollador tiene que centrarse menos en el despliegue y puede concentrar sus esfuerzos en la funcionalidad.

Es un concepto distinto al de la máquina virtual introducida en el punto anterior, ya que una máquina virtual requiere de un sistema operativo instalado, siendo más compleja, mientras que Docker solo necesita del sistema operativo del propio equipo, por lo que es un mucho más ligero que el ejemplo anterior.

Se podría entender entonces, que Docker adquiere los componentes que puede usar del Sistema Operativo, aquellos que son comunes a cualquier despliegue, y encapsula dentro del propio contenedor aquellos que son más críticos y que generan más problemas en distintas implementaciones.

5.1.3. Hyperledger Fabric

Hyperledger Fabric [3] es la más conocida de las implementaciones disponibles de Hyperledger. En muchas ocasiones se puede confundir a Fabric con el común de hyperledger, cuando en la realidad este se trata de un proyecto mucho más extenso.

Fabric permite la creación de redes privadas. Es por tanto, en concepto, una Blockchain privada que permite dotar de permisos y privilegios a los distintos agentes de la misma. Tiene una utilidad única a la hora de crear redes que sirvan para fundamentar modelos de negocio entre distintos actores con intereses concretos. Es la base fundamental de este proyecto, dado que constituye un espacio perfecto para la construcción de una red descentralizada con los requisitos que se necesitan en el caso planteado.

Su estructura se basa en varias piezas:

[3] https://www.hyperledger.org/use/fabric

- Organizaciones. Son la pieza básica fundamental para organizar la red, representan actores externos como compañías, que tienen ciertos intereses y particularidades.

- Nodos. Los nodos están asociados con las organizaciones, y se encargan de replicar el libro mayor, así como de firmar y transmitir las transacciones realizadas por cada una de los usuarios de cada organización. Uno de los nodos corresponde al nodo "Orderer", que se encarga de organizar el orden en que las transacciones son escritas y posteriormente transmitidas al resto de nodos.

- Clientes. Representa a los usuarios de cada organización, es la forma que tienen las organizaciones de interactuar con la Blockchain, enviando sus transacciones hasta uno de los nodos a los que tienen acceso.

- MSP (Membership Service Provider). Es el componente que define las reglas en que los usuarios son validados y autenticados. Maneja distintos IDs (MSPid) que permiten reconocer cuando un cliente está realizando una acción para la cual tiene privilegios.

- Canal. Es el componente que conecta las distintas organizaciones de una red. Permite establecer conexiones en función de la información que se quiere transmitir y puede dar lugar a que existan canales privados entre organizaciones que forman parte de otro canal común para transmitir datos sensibles.

- Chaincode. Representa el código de la plataforma, son las acciones y la lógica de la red, e incluye las acciones que se pueden llevar a cabo en la misma. Se puede comparar con los "smart contracts" de Ethereum vistos anteriormente, como piezas de código que usan la red para ejecutarse y asegurar que se cumplen las condiciones especificadas.

Gracias a estos elementos, permite construir redes en las que los participantes son identificables, se puede otorgar permisos concretos a distintas .ºrganizaciones.en función de sus necesidades, existe una latencia baja a la hora de confirmar transacciones y puede darse la privacidad en la comunicación de información sensible entre ciertas organizaciones.

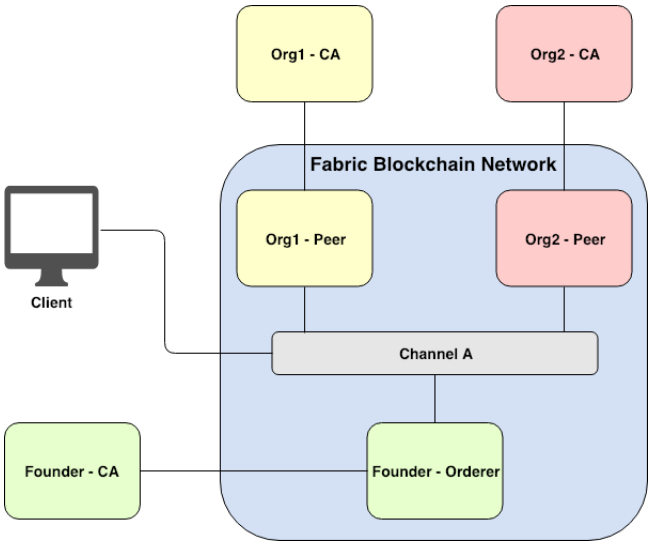

Figura 5.2: Estructura de Hyperledger Fabric

5.1.4. NodeJS

NodeJS [4] es un framework de Javascript que está orientado a la creación de aplicaciones de red escalables. fue creado por los desarrolladores originales de Javascript, y consiguió que un lenguaje orientado meramente al navegador como javascript fuese capaz de ejecutarse como una aplicación tradicional en cualquier ordenador.

Permite construir aplicaciones orientadas al servidor en la que la concurrencia es muy alta. Para ello utiliza un modelo de entrada y salida sin bloqueo, que basa su flujo de ejecución en la utilización de eventos. Esto lo hace ligero, eficiente y escalable ante un número grande de peticiones.

Gracias a este acercamiento, al contrario que otros acercamientos al desarrollo web en los que cada conexión genera un subproceso que consume recursos, Node trabaja en un solo subproceso, utilizando el modelo anteriormente comentado, para conseguir procesar decenas de miles de peticiones al mismo tiempo.

Ante una petición, el sistema, en lugar de bloquearse, trabaja de forma asíncrona, por lo que cuando entra una petición nueva asigna a la petición anterior un callback que devolverá el resultado cuando este esté disponible. De esta forma, devolverá los datos cuando sea posible, pero no bloqueará el funcionamiento del servidor.

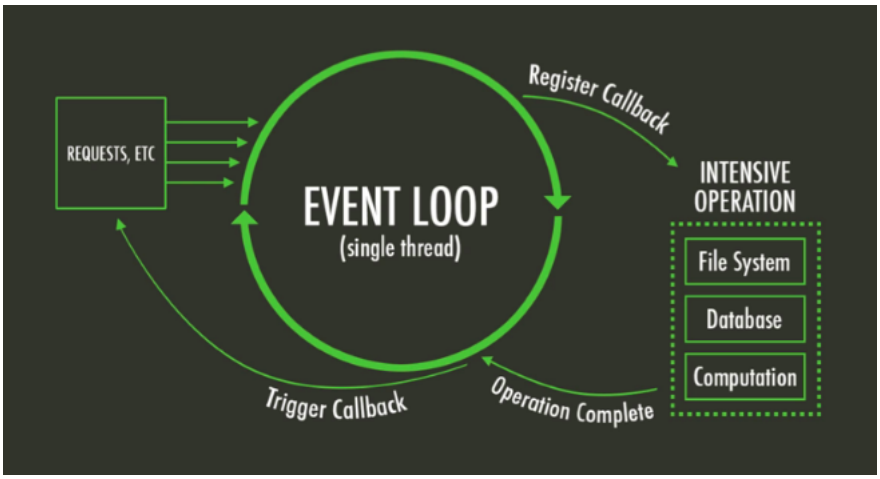

Figura 5.3: Funcionamiento asíncrono de NodeJS

En el caso de Hyperledger Fabric, tiene un SDK disponible que permite el desarrollo de chaincode usando NodeJS como base. Es uno de los frameworks principales en Fabric, junto a Python, y es actualmente el más usado por el groso de desarrolladores de Hyperledger.

También es sencillo su uso para la construcción de APIs que permitan la interacción con la Blockchain a un nivel más alto, facilitando el uso a una gran cantidad de usuarios novatos que tendrán que usar la plataforma.

5.1.5. React

React [5] es un framework de Javascript que está orientado a la creación de aplicaciones web. Permite construir aplicaciones orientadas principalmente a su ejecución del lado del cliente, por lo que resulta un complemento perfecto del framework anteriormente citado NodeJS. Es muy común verlos trabajar de forma conjunta en aplicaciones web complejas.

[4] https://nodejs.org/es/
[5] https://es.reactjs.org/

Está desarrollada por Facebook, es software libre y hasta hace poco era con diferencia el framework de JS más popular en Github.

Está orientado principalmente a la creación de interfaces de usuario dinámicas. React solo renderiza los cambios que se han producido en la interfaz ante datos u órdenes recibidas, es por ello que es muy eficiente y no gasta más recursos de los necesarios.

5.1.6. Visual Studio Code

Visual Studio Code [6] es un editor de texto sofisticado propietario de Microsoft que permite una gran cantidad de opciones a la hora de trabajar con código.

Admite prácticamente todos los lenguajes de programación de forma estándar, por lo que es una herramienta muy versátil que es útil en cualquier tipo de proyecto.

Es multiplataforma, teniendo binarios disponibles para Windows, Linux y Mac OS, por lo que es fácil mantener tus proyectos independientemente del sistema operativo usado.

Cuenta con una extensa cantidad de plugins que añaden funcionalidades al sistema, estos van incluso desde conexiones directas con repositorios de código hasta nuevos lenguajes de programación que no se contemplan en su versión por defecto.

Es open source y su código está disponible en Github, por lo que cualquier usuario puede auditar y verificar los usos del programa y contribuir a su desarrollo.

5.1.7. Git

Git[7] es una herramienta para realizar un control de versiones sobre código de forma distribuida. Permite organizar el desarrollo a un equipo y facilitar la paralelización de tareas.

Fue diseñada por Linus Torvalds y es software libre, permite desplazarse por todas las versiones del código en un momento dado y descargar versiones anteriores a la actual, permitiendo así retomar el desarrollo desde cualquier punto.

Uno de sus mayores exponentes es Github, una plataforma de software libre en la que los usuarios suben sus trozos de código para el beneficio del resto de la plataforma, así como la creación de proyectos de desarrollo conjunto y colaborativo.

5.2 Configuración del entorno

En este punto se plantearán los requisitos básicos que tienen que inicializarse en la máquina para el correcto funcionamiento de la red. Al ser Hyperledger Fabric un entorno muy complejo que tiene muchas dependencias para su despliegue, creo necesario incidir en las necesidades del entorno y en los pasos a seguir para contar con un entorno funcional de cara a realizar las pruebas que se tratarán en esta memoria.

En primer lugar, se deben instalar los requisitos, para ello, se requiere de un equipo ejecutando una versión actualizada de Linux o un equipo que tenga desplegada una máquina virtual con Linux. En este caso, se dispone de un Host con Windows, desplegando Linux Mint mediante VMWare.

[6] https://code.visualstudio.com/
[7] https://git-scm.com/

En primer lugar se instalará Golang, lenguaje sobre el que está escrito Hyperledger y el cual es necesario para su funcionamiento, además, se instanciará el GOPATH para que los binarios instalados por Golang queden referenciados y puedan llamarse directamente en la terminal sin acudir a la ruta completa.

```
sudo apt-get install golang-go

export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin
```

A continuación, se instalarán las dependencias de los "chaincodes", en este caso solo se instalará NodeJS, a pesar de que los "chaincodes" también pueden ser escritos en Python, debido a que en este proyecto solo se ha usado el mentado lenguaje. También se instalarán herramientas tan utilizadas como cURL y git, introducidas en el anterior apartado.

```
sudo apt-get install nodejs
sudo apt-get install npm
sudo apt-get install git
sudo apt-get install curl
```

También se debe instalar docker, ya que la herramienta que usaremos, Hyperledger Fabric, basa su funcionamiento en los contenedores de la citada aplicación. Para ello, se debe ejecutar los siguientes comandos

```
sudo apt-get install docker.io
sudo apt-get install docker-ce
```

Por último, se instala Hyperledger desde su repositorio oficial, junto a todas las dependencias que tiene implícitas:

```
sudo curl -sSL https://goo.gl/6wtTN5 | sudo bash
```

5.3 Creación de la red

En este punto trataremos y definiremos las organizaciones que van a formar parte de la red y cuantos nodos tendrán cada uno de ellos. En este caso se ha introducido nombres estándar para que sea un esquema extrapolable a otras cadenas de suministro y fácilmente replicable.

En la figura 5.2 se pueden ver representadas las organizaciones incorporadas a la red:

Figura 5.4: Organizaciones de la red

- **Proveedor** Es el encargado de suministrar las materias primas al resto de miembros de la cadena. En este caso, es el responsable de obtener la carne de las vacas de la granja de la que dispone.
- **Repartidor** Es el encargado de realizar los envíos entre los distintos miembros de la cadena, en caso de ser necesario. En este caso, realiza los envíos desde el proveedor hasta el fabricante y desde el fabricante a la tienda final.
- **Fabricante** Es el encargado de trabajar las materias primas que obtiene del proveedor. En este caso, convertirá la carne en hamburguesas y las empacará para su distribución.
- **Tienda final** Es el encargado de poner a disposición del cliente las hamburguesas recibidas del fabricante. En este caso, serán las distintas carnicerías que forman parte de la cadena.

Para que cada uno de los elementos sea representado correctamente, es necesario editar los ficheros de configuración crypto-config.yaml, configtx.yaml y docker-compose-cli.yaml. Se pueden encontrar adjuntos en el Anexo.

El fichero crypto-config.yaml es el encargado de definir que organizaciones conforman la red, además del nodo Orderer, que se encarga de ordenar las transacciones realizadas para introducirlas en el libro mayor.

Un ejemplo de configuración para una de las organizaciones es la expuesta en este fragmento del fichero crypto-config.yaml.

```
OrdererOrgs:
  - Name: Orderer
    Domain: hamburguesas.com
    EnableNodeOUs: true
    Specs:
      - Hostname: orderer

PeerOrgs:
  - Name: Fabricante
    Domain: fabricante.hamburguesas.com
    EnableNodeOUs: true
    Template:
      Count: 1
    Users:
      Count: 1
```

Se puede observar como se define el nodo organizador, en el que se especifica el dominio y el nombre asignado. A continuación, se especifica uno a uno todas las organizaciones que forman parte de la red. Se puede ver el valor asignado, el dominio al que pertenece, así como el número de nodos que tendrá el dominio. En este caso, se asigna solo un nodo para simplificar el despliegue por ser un entorno teórico.

El fichero configtx.yaml, por otra parte, define el identificador MSP de cada organización, para que puedan identificarse como parte de la red. Por último, el fichero docker-compose-cli.yaml es el encargado de desplegar los contenedores docker de la red. En este se especifica cuales son los contenedores que se desplegarán y detalles intrínsecos al mismo como su ruta de configuración o el nombre del contenedor.

Un ejemplo de configuración para uno de los nodos de la red sería el siguiente:

```
peer0.proveedor.hamburguesas.com:
    container_name: peer0.proveedor.hamburguesas.com
    extends:
```

```
4       file:   base/docker-compose-base.yaml
5       service: peer0.productor.hamburguesas.com
6     networks:
7       - byfn
```

A continuación, se deben crear los certificados asociados a cada uno de los dominios de nuestra red. Estos dominios que son definidos en el fichero crypto-config.yaml anteriormente mentado, se pasan como configuración al binario cryptogen que el propio Hyperledger provee para la generación de estos certificados. Tenemos el ejemplo de su creación para esta red en la figura 5.3

```
jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-template$ sudo ../bin/cryptogen generate
 --config ./crypto-config.yaml --output="./crypto-config"
fabricante.hamburguesas.com
repartidor.hamburguesas.com
proveedor.hamburguesas.com
tienda_final.hamburguesas.com
```

Figura 5.5: Creación de certificados

Por su estructura de Blockchain, en la que como se ha incidido en previos capítulos, los datos se organizan en bloques, es esencial la creación del bloque génesis (el primer bloque de la cadena), además de la creación de un canal para que las distintas organizaciones puedan comunicarse. Se pueden crear canales para permitir comunicaciones complejas entre distintas organizaciones que tienen distintas necesidades estructurales, en este caso se ha tomado la decisión de introducir un solo canal para simplificar el diseño, ya que no existe ninguna necesidad concreta que deba limitar la comunicación entre ninguno de los agentes.

En la figura 5.4 puede verse ejemplificada la creación del bloque génesis, así como el canal de comunicación entre las organizaciones.

```
jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-template$ sudo ../bin/configtxgen -configPath ./ -profile Sup
plyOrdererGenesis -outputBlock ./channel-artifacts/genesis.block -channelID s
2020-11-27 01:35:43.556 CET [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-11-27 01:35:43.560 CET [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 orderer type: solo
2020-11-27 01:35:43.560 CET [common.tools.configtxgen.localconfig] Load -> INFO 003 Loaded configuration: configtx.yaml
2020-11-27 01:35:43.561 CET [common.tools.configtxgen] doOutputBlock -> INFO 004 Generating genesis block
2020-11-27 01:35:43.561 CET [common.tools.configtxgen] doOutputBlock -> INFO 005 Creating system channel genesis block
2020-11-27 01:35:43.563 CET [common.tools.configtxgen] doOutputBlock -> INFO 006 Writing genesis block
jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-template$ sudo ../bin/configtxgen -configPath ./ -profile Cad
ena_de_Suministro -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel
2020-11-27 01:35:59.168 CET [common.tools.configtxgen] main -> INFO 001 Loading configuration
2020-11-27 01:35:59.172 CET [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: configtx.yaml
2020-11-27 01:35:59.172 CET [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel configtx
2020-11-27 01:35:59.174 CET [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel tx
```

Figura 5.6: Creación de bloque génesis y canal de comunicación

Una vez desplegada la estructura de la organización, se deben desplegar los contenedores que representarán los nodos de cada organización, así como el nodo que se encarga de ordenar las transacciones en el libro mayor.

El último paso que se realizó fue la unión de las distintas organizaciones a un canal común. El canal supone la vía de conexión entre las distintas organizaciones, por tanto, el consorcio entre estas solo será real si estas organizaciones se encuentran en el mismo canal.

Al acabar de desplegar estos contenedores ya se cuenta con la estructura básica de la red planteada al inicio, con 4 organizaciones que representan agentes básicos y estándares en cualquier cadena de suministro y un canal común que permite compartir la información.

```
jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-template$ sudo docker-compose -f docker-compose-cli.yaml
up -d
Creating network "net_byfn" with the default driver
Creating volume "net_orderer.hamburguesas.com" with default driver
Creating volume "net_peer0.proveedor.hamburguesas.com" with default driver
Creating volume "net_peer0.fabricante.hamburguesas.com" with default driver
Creating volume "net_peer0.repartidor.hamburguesas.com" with default driver
Creating volume "net_peer0.tiendafinal.hamburguesas.com" with default driver
Creating volume "net_caFabricante" with default driver
Creating volume "net_caRepartidor" with default driver
Creating volume "net_caProveedor" with default driver
Creating volume "net_caTienda_final" with default driver
WARNING: Found orphan containers (peer0.org1.example.com, peer0.org2.example.com, orderer.example.com) for this project
. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag
to clean it up.
Creating peer0.repartidor.hamburguesas.com  ... done
Creating peer0.proveedor.hamburguesas.com   ... done
Creating orderer.hamburguesas.com           ... done
Creating peer0.tiendafinal.hamburguesas.com ... done
Creating peer0.fabricante.hamburguesas.com  ... done
Creating cli                                ... done
```

Figura 5.7: Despliegue de contenedores

5.4 Lógica de la red

Una de las partes más críticas en cualquier red privada entre empresas es la lógica que sustenta las operaciones entre las distintas organizaciones. Supone un punto fundamental, porque establece de forma explícita las reglas y controles que estas organizaciones van a introducir. Estas reglas permiten que las organizaciones preestablezcan que condiciones se van a seguir para asegurar la continuidad futura de la cadena de suministro, permitiendo que se omita en gran parte la confianza que debe depositar cada actor en el resto de miembros de la cadena, ya que será la propia red la que se encargará de verificar y asegurar que las condiciones acordadas se cumplan.

En esta cadena de suministro se han establecido varias funciones básicas que pueden ser extendidas en el futuro gracias a la API que permitirá la conexión de las empresas con la red de forma sencilla:

- **addAsset(asset)**: Función que permite añadir activos a la red. Es una función básica que en este caso concreto será extendida mediante la API para que sea la base de la función que añade suministros de carne en el Proveedor, así como la que convierte la carne en hamburguesas si la carne cumple los gramos necesarios. Se han establecido ciertas medidas de control para que esta función solo pueda ser llamada por dos organizaciones, proveedor y fabricante, ya que carecería de sentido que la tienda final o el distribuidor creasen nuevos activos en esta cadena.

- **queryAsset(Id)**: Función que, dado un ID, devuelve el estado actual del activo, sin tener en cuenta sus estados pasados.

- **setOwner(Id, Owner)**: Función que, dado un ID, cambia el dueño del activo en la red. Es una función que se usará por todos los agentes de la cadena a fin de reflejar la entrega del activo al siguiente agente de la cadena.

- **setPosition(Id, latitud, longitud)**: Función que, dado un ID, cambia la posición del activo. Permite reflejar con exactitud la posición de un activo en la cadena, hecho que permitirá que la información que tengan las demás organizaciones sea más efectiva.

- **getHistory(Id)**: Función que, dado un ID, recopila la información del activo y los estados por los que ha pasado. Es especialmente útil porque permite a cualquier participante en la cadena conocer perfectamente de donde viene el producto a ciencia cierta.

Se puede ver un ejemplo de la implementación de una de las funciones en el siguiente fragmento de código:

```
async addAsset(ctx, asset) {
    await ctx.stub.putState(JSON.parse(asset).id.toString(), Buffer.from(asset)
        );
    return ctx.stub.getTxID();
}
```

El parámetro ctx representa el contexto de la transacción realizada, o la llamada a la función. Este parámetro permite la conexión con la API de Hyperledger Fabric, pudiendo acceder así a variables globales de la red. Permite, entre otras cosas, cambiar el estado de un activo, generar transacciones y obtener valores como el MSPid del creador de la transacción, permitiendo así relacionar y establecer restricciones sobre quien puede realizar ciertas acciones. [8]

Para desplegar el código que se ejecutará en todos los nodos de la red, se debe instalar el código en cada uno de ellos.

Un ejemplo de la instanciación de cada uno de los contenedores mediante el contenedor CLI previamente creado sería el siguiente:

```
docker exec -it cli ./scripts/install-cc/install-peer.sh peer0 Proveedor
    ProveedorMSP 7051 1.0
```

En él se indica:

- La ruta del script install-peer.sh.
- El nombre del peer en el que se instalará el chaincode.
- El nombre de la organización.
- El MSPid asociado a esta organización.
- El puerto en el que escucha el contenedor en el que se ejecuta el nodo.

Cuando se ha instalado, se debe instanciar en solo 1 de los nodos, de forma que en ese momento se creará un contenedor que permitirá que todas las organizaciones llamen a este a la hora de intentar realizar una operación.

El código que permite la instanciación es el siguiente:

```
docker exec -it cli ./scripts/install-cc/instanciate.sh
```

Automatizando el proceso, podemos ver desplegados todos los contratos en la red en la figura 5.8.

Una vez realizado el proceso, el contrato estará completamente desplegado en la red, de forma que ya está creada una red con los requisitos necesarios para su funcionamiento, así como la lógica que permite relacionar las operaciones en la misma.

5.5 Creación de usuarios

Se ha conseguido desplegar una red con todos los requisitos para que se plantee una cadena de suministro en un entorno real, pero para poder interactuar con ella se han de incorporar los usuarios que participarán en ella.

[8] https://hyperledger-fabric.readthedocs.io/en/release-2.2/developapps/transactioncontext.html

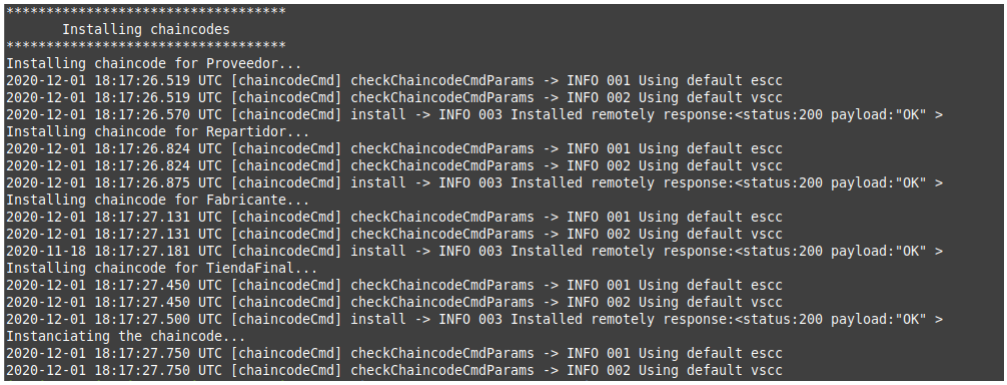

Figura 5.8: Despliegue del contrato

Durante el transcurso de este punto se hará referencia al concepto de *wallet* o monedero. Es necesario no confundir este concepto con el planteado por otras Blockchain como Ethereum o Bitcoin, el monedero en Hyperledger Fabric es usado para guardar las identidades de los usuarios. De ningún modo almacena ningún tipo de moneda o activo. Un usuario puede tener distintas identidades en su monedero, cada cual con unos permisos diferentes en cada canal. En este caso, por simplificar el despliegue, se asumirá que un usuario cuenta con una sola identidad y existirá solo un usuario común por organización, además del administrador.

Una de los requisitos básicos para interactuar de forma efectiva con la red que se ha desplegado es el SDK (Software Development Kit) de Hyperledger. Este permite:

- Interactuar con la red enviando transacciones y hacer consultas específicas sobre objetos en el libro mayor distribuido.
- Interactuar con la red como un cliente, interactuando con los contratos haciendo peticiones a funciones, monitorizar eventos...
- Funciones propias de la entidad certificadora, como registrar usuarios y los certificados asociados a los mismos, eliminar o quitar privilegios a usuarios...

Para incorporar al primero de los usuarios, el administrador, se ha usado un script llamado newAdmin.js que toma inspiración en el código planteado en el repositorio oficial de github de Hyperledger Fabric[9] en el que hay distintos ejemplos de implementaciones exitosas. Este script al ser llamado con el nombre de la organización como parámetro realiza las siguientes acciones:

- Llama al fichero connection-org.json que se ha creado previamente cuando se han desplegado los contenedores y que incluye los datos de conexión de la organización.
- Crea una nueva Autoridad Certificadora asociada a la organización, cuyo fin será la creación de los certificados que autorizan a los nuevos usuarios de esta.
- Crea un nuevo monedero en el que almacenar las identidades correspondientes a la organización.
- Crea la identidad del usuario administrador dados un nombre de usuario y una contraseña, guardándolo a posteriori en el monedero.

[9] https://github.com/hyperledger/fabric-samples/blob/release/fabcar/enrollAdmin.js

También se ha integrado un script para la creación de usuarios comunes, de nuevo, planteando como inspiración el código en el repositorio antes mentado. Este código repite los pasos anteriores pero no crea una nueva autoridad certificadora ni un nuevo monedero, sino que simplemente recupera la conexión de la organización y se conecta a la unidad certificadora (usando las credenciales del administrador) para la creación de un nuevo usuario en el monedero.

5.6 Despliegue de la API

Por último, ya con todas las piezas dispuestas sobre el tablero, queda crear una API que permita a usuarios externos interactuar con la red de forma más intuitiva, intentando llevar las conexiones con los distintos nodos a un lenguaje de más alto nivel.

Esto servirá para integrar más fácilmente la red en aplicaciones ya existentes en distintas empresas, así como generar aplicaciones nuevas e interfaces de usuario accesibles para soportar la carga del usuario final, cuyo especialización no suele ser técnica.

Primero se creará una aplicación javascript que se encarga de realizar la conexión con la red. Se aislará esta función de la API en sí para establecer una estructura modular, de forma que los cambios realizados en la API no afecten a la conexión con el nodo.

Este aplicación cuenta con un solo método que será llamado por la API, y conectará con un nodo pasado por parámetro, además de la ruta en la que se encuentra el monedero con las identidades de usuario de la organización. Para simplificar, se usará el usuario estándar en todas las conexiones con la red.

Esta función devuelve un objeto que constituye el contrato desplegado en la red, de forma que sirve para interactuar directamente con él usando los permisos del usuario que alberga el monedero. Puede verse reflejado en el fragmento de código inmediatamente posterior.

```
1  //Obtiene el canal en el que est  desplegado el contrato.
2  const canal = await gateway.getNetwork(CHANNEL);
3  //Obtiene el contrato desplegado en la red.
4  const contrato = canal.getContract(CONTRACT);
5  return contrato;
```

Ya automatizada la conexión con la red, queda generar los endpoints de la API en los cuales las distintas aplicaciones realizará sus peticiones.

Se trata, evidentemente, de un enfoque teórico, ya que en un despliegue real cada organización contaría con su API y las empresas no contarían en su sistema con las identidades de los usuarios de todas las organizaciones. Para simplificar el despliegue, se ha automatizado la conexión con la red de forma que cada llamada conecta al nodo de una organización con el usuario de la misma en función de la petición y su relación con las necesidades de la organización.

Para ejemplificar, se explicará en detalle uno de los endpoints que, por concepto, tiene relación directa con la organización Repartidor"de la red.

```
1 app.post('/api/nuevaLocalizacion', async function (req, res) {
2   try {
3     const contract = await
4     fabricNetwork.connectNetwork('connection-repartidor.json', 'wallet/wallet-
          repartidor');
```

El endpoint llama a la función connectNetwork() que hemos definido en la aplicación anterior, pasando como parámetros el fichero de conexión de la organización Repartidor", así como la referencia al directorio en el que se encuentra su monedero de identidades.

```
    let tx = await contract.submitTransaction('setPosition', req.body.id.
        toString(), req.body.latitude.toString(), req.body.longitude.toString()
        );
    res.json({
      status: 'OK - Transaction has been submitted',
      txid: tx.toString()
    });
  } catch (error) {
    console.error(`Failed to evaluate transaction: ${error}`);
    res.status(500).json({
      error: error
    });
  }
});
```

El endpoint contacta de forma directa con la función "setPosition()"que se encuentra en el contrato de la red. Se analiza el contenido del JSON en la petición POST y se cambian los parámetros latitud y longitud de un objeto con una ID dada.

Siguiendo esta aproximación se crean el resto de endpoints de la API, quedando con la siguiente estructura:

- **POST /api/nuevaCarne/** Endpoint que llama a la función addAsset() enviando un JSON que representa un activo de tipo carne. Este activo tiene como dueño el proveedor (ya que es el único que dispone de granjas y puede crear materia prima) e indica la cantidad de gramos pasándole el parámetro cantidad. Devuelve el txID de la carne generada.
- **POST /api/nuevaLocalizacion**: Endpoint que llama a la función setLocation() y cambia la latitud y longitud de un objeto dado un ID concreto.
- **POST /api/nuevoDueno**: Endpoint que llama a la función setOwner() y cambia el dueño de un activo dado un ID.
- **GET /api/historial/<id>**: Endpoint que, dado un id, llama a la funcion getHistory() y devuelve el historial de un activo, incluyendo todos los activos que tienen relación con él.
- **POST /api/nuevaHambuguesa**: Función que, dado un ID, llama a la función addAsset() y produce una hamburguesa con la carne suministrada a la que se asocia el ID, devolviendo el ID del nuevo producto.
- **GET /api/datoProducto/<id>**: Función que, dado un ID, llama a la función queryAsset() y obtiene el estado actual del activo asociado, sin incluir el historial completo.

CAPÍTULO 6
Pruebas

En este capítulo se interactuará con la red creada mediante la API que se ha dispuesto en el anterior capítulo, por medio de la aplicación curl. Se trazará el flujo habitual previsto en esta cadena logística y se comprobará la trazabilidad de los datos.

El primer agente en esta cadena es el Proveedor, es el encargado de crear y poner a disposición del fabricante la carne para que se produzcan las hamburguesas.

Se realizará una petición POST usando al endpoint nuevaCarne de la API, pasando como parámetro el id, la latitud, la longitud, el peso y el dueño:

```
curl --request POST
 --url http://localhost:3000/api/nuevaCarne
 --header 'content-type: application/json'
 --data '{
       "id":1,
       "latitude":"43.3623",
       "longitude":"8.4115",
       "peso":500,
       "owner": "Proveedor"
        }'
```

```
jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-supply-chain/src/server$ curl --request POST --url http:/
/localhost:3000/api/nuevaCarne --header 'content-type: application/json' --data '{"id": 1,"latitude":"43.3623","longitu
de":"8.4115","peso":500,"owner":"Proveedor"}'
{"status":"OK - Transaction has been submitted","txid":"6f23bf54f154f461850483340e605e8c2ca7fcf00b2a78b83bd9b8644c19355
```

Figura 6.1: El Proveedor añade carne a su stock

Al acabar, nos devuelve el id de la transacción, certificando que esta ya ha sido enviada.

Tras recibir una orden de compra por parte del fabricante, el proveedor decide enviarle la cantidad de carne que este ha pedido. Para ello debe contactar con el repartidor y contratar un envío. Cuando esto se ha realizado, se establece al repartidor como nuevo dueño del producto.

Se realiza una petición POST al endpoint nuevoDueno pasando como parámetro el id de la carne, así como su nuevo dueño:

```
curl --request POST
 --url http://localhost:3000/api/nuevoDueno
 --header 'content-type: application/json'
 --data '{
       "id":1,
       "owner: "Repartidor"
        }'
```

```
jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-supply-chain/src/server$ curl --request POST --url http://lo
calhost:3000/api/nuevoDueno --header 'content-type: application/json' --data '{"id": 1,"owner":"Repartidor"}'
{"status":"OK - Transaction has been submitted","txid":"4c579b071452558049490a95b5d0326757b229d58feccc22dfc4e81045ebb656"}
```

Figura 6.2: El Proveedor entrega la carne al repartidor

El repartidor comienza entonces el proceso logístico para enviar la carne, y cambia la localización del producto para establecer que esta se encuentra ya en sus almacenes esperando ser trasladada.

Se realiza una petición POST al endpoint nuevaLocalizacion, indicando como parámetros el id, la latitud y la longitud:

```
curl --request POST
 --url http://localhost:3000/api/nuevaLocalizacion
 --header 'content-type: application/json'
 --data '{
      "id":1,
      "latitude: "12.323232",
      "longitude: "34.4234"
      }'
```

```
jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-supply-chain/src/server$ curl --request POST --url http://lo
calhost:3000/api/nuevaLocalizacion --header 'content-type: application/json' --data '{"id": 1,"latitude": "12.323232", "lo
ngitude": "34.4232"}'
{"status":"OK - Transaction has been submitted","txid":"30a4839eba21364ddf4f76c91a687e462d30b2f2fa61ef0ff5356fe647558e39"}
```

Figura 6.3: El repartidor indica que el producto ya está en sus almacenes

En el transcurso, el repartidor, que cuenta con un perfil alejado del mundo técnico, quiere comprobar si, efectivamente, los datos que ha introducido en la aplicación se han actualizado y el producto figura ya en sus almacenes. Para ello, consulta el estado de este producto:

Se realiza una petición GET al endpoint datoProducto/<id>, reemplazando el valor ID en la petición:

```
curl --request GET
 --url http://localhost:3000/api/datoProducto/1
 --header 'content-type: application/json'
```

```
}}jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-supply-chain/src/server$ curl
--request GET --url http://localhost:3000/api/datoProducto/1 --header 'application/json'
{"result":{"id":1,"latitude":"12.323232","longitude":"34.4232","peso":500,"owner":"Repartidor"
}}jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-supply-chain/src/server$
```

Figura 6.4: El repartidor consulta el estado del producto

El repartidor, ya tranquilo por haber actualizado correctamente los datos, decide seguir con la entrega y llevar el producto a su lugar de destino, se vale de los mismos mecanismos antes mentados para cambiar la localización a la posición de la fábrica de destino, así como el dueño al Fabricante.

Cuando el Fabricante recibe la carne, la procesa con urgencia porque tiene un pedido pendiente que requiere de una hamburguesa.

En este caso, se realiza una petición POST al endpoint nuevaHamburguesa que incluye los parámetros habituales en la creación de carne, pero se le indica el Id previo asociado a la carne de la que proviene. De esta forma, los dos productos están asociados:

```
curl --request POST
 --url http://localhost:3000/api/nuevaHamburguesa
 --header 'content-type: application/json'
 --data '{
```

```
        "id":2,
        "latitude":"43.3623",
        "longitude":"8.4115",
        "peso":50,
        "carneId": 1,
        "owner": "Proveedor"
         }'
```

```
jrrgimenez@jrrgimenez-virtual-machine:~/fabric-samples/hlf1.4-supply-chain/src/server$ curl --request POST --url http://l
ocalhost:3000/api/nuevaHamburguesa --header 'content-type: application/json' --data '{ "id":2, "latitude": "12.323232", "
longitude": "34.4232", "peso":"50", "carneId":1, "owner": "Fabricante" }'
{"status":"OK - Transaction has been submitted","txid":"7fbb99425d95ef5a98d5fc81f7f99d854fac85e088c97ec1722ce153f5179f9c"
```

Figura 6.5: El fabricante procesa las hamburguesas

Ante un pedido de la tienda final, el fabricante contacta con el repartidor para realizar el envío y se repite el mismo proceso que se ha podido observar en la conexión entre el proveedor y el fabricante. No se incidirá en las llamadas que se han realizado por ser idénticas salvando el cambio de variables.

Si se quiere consultar el historial de estados en cualquier punto de la cadena, la llamada al endpoint historial/<id>iterará a lo largo de los estados que se encuentran en la cadena para ese id dado, dándoles formato y presentándolos:

```
curl --request GET
 --url http://localhost:3000/api/historial/2
 --header 'content-type: application/json'
```

Finalizadas las pruebas, se cuenta con una red totalmente funcional que permite desarrollar todas las tareas básicas a la hora de trazar alimentos, con una API que expone de forma sencilla todas las funciones en la red para que cualquier empresa que entre al consorcio sea capaz de integrarla dentro de su infraestructura.

CAPÍTULO 7
Conclusiones

Durante el recorrido de esta memoria, se ha intentado plantear una visión global de los problemas generales que enfrentan las cadenas de suministro en su concepción, incluyendo aquellos que se han tratado de enfrentar en el pasado y aquellos que, por concepto, se han instaurado como base del funcionamiento de las mismas.

Hyperledger, como solución, plantea una alternativa real a problemas cotidianos de empresas que sustentan nuestro día a día. Es comprensible entender que se pierdan por el camino algunos componentes que tienen las Blockchain públicas, al permitir la existencia de permisos especiales y privilegios o censurar la entrada al consorcio a solo aquellos actores que contribuyen dentro del propio negocio. Son ciertas condiciones que se sienten necesarias ante la obligatoriedad de cumplimiento normativo al que se someten las empresas propias de las cadenas de suministro. Pese a todo, siento que como consumidor adquiero un gran poder al ser capaz de trazar de forma completa y veraz la procedencia de los productos de consumo.

No solo como consumidor existen ventajas; plantear un modelo basado en la integridad y disponibilidad de los datos permite que los agentes que se encargan de auditar el debido cumplimiento de la normativa tengan más herramientas para evitar fraudes y estafas.

Finalmente, es objetivo decir que se le conceden más herramientas tanto al consumidor que intenta analizar los objetos que consume, pudiendo trazar estos de forma efectiva, como aquel que no vive preocupado en exceso, dado que los controles serán más exhaustivos y veraces.

Creo que la solución expuesta en este trabajo supone un buen punto de partida para la construcción de una red descentralizada en la que se puedan integrar agentes con otros intereses. Como estudio futuro sería interesante el integrar nuevos canales entre organizaciones que deben compartir información confidencial, la integración con dispositivos IoT (Internet of Things) para la automatización de las funciones que permiten la actualización del estado de los alimentos, y el desarrollo de los contratos de compra/venta integrados en la propia plataforma para integrar la mayor cantidad posible de información.

Es complicado analizar un sector tan complejo y crítico como el de las cadenas de suministro, pero este trabajo me ha permitido obtener una visión profunda sobre un problema global que afecta a todos los sectores por igual. Es un hecho afirmar que avanzamos hacia un mundo digitalizado, el auge del IoT como gran revolución terminará de transformar los procesos físicos tal y como los conocemos.

Es por ello que se muestra un punto relevante hacia el que dirigirse, creo que la tecnología Blockchain puede suponer un cambio profundo a la hora de plantear no solo

modelos económicos, sino también modelos organizativos. Bitcoin ya ha cambiado la forma que tenemos de entender la economía, nos ha permitido abstraer y entender que el dinero es un ente más complejo del que creíamos o nos había acostumbrado. Ha cambiado la forma en la que muchas personas ahorran o guardan valor. Ethereum, por su parte, ha cambiado la forma en que se entienden las economías descentralizadas, ha planteado modelos de negocio que antes eran impensables y que suponen una amenaza directa para conglomerados internacionales. Está en juego la privacidad y la integridad de los datos que ofrecemos, por ello creo que es necesario seguir caminando y construyendo en la dirección hacia la que plantea este trabajo.

Bibliografía

[1] Committee on Payments and Market. *Distributed Ledger technology on Payments and Infrastructures*, Bank for International Settlements, Febrero, 2017. Consultado en https://www.bis.org/cpmi/publ/d157.pdf

[2] Bitcoin: A Peer-to-Peer Electronic Cash System Consultado en https://bitcoin.org/bitcoin.pdf

[3] Bitcoin, preguntas y respuestas frecuentes Consultado en https://bitcoin.org/es/faq#transacciones

[4] Ethereum Smart Contracts Consultado en https://ethereum.org/es/learn/#smart-contracts

[5] Cuantos tipos de Blockchain hay Consultado en https://academy.bit2me.com/cuantos-tipos-de-blockchain-hay/

[6] Utility Tokens vs Security Tokens: What's the Difference? Consultado en https://blog.bcas.io/utility-tokens-security-tokens-difference

[7] Hyperledger Consultado en https://www.hyperledger.org/

[8] How Walmart brought unprecedented transparency to the food supply chain with Hyperledger Fabric Consultado en https://www.hyperledger.org/wp-content/uploads/2019/02/Hyperledger_CaseStudy_Walmart_Printable_V4.pdf

[9] Change Healthcare using Hyperledger Fabric to improve claims lifecycle throughput and transparency Consultado en https://www.hyperledger.org/wp-content/uploads/2019/06/Hyperledger_CaseStudy_ChangeHealthcare_Printable_6.19.pdf

[10] JOINT FAO/WHO FOOD STANDARDS PROGRAMME Consultado en http://www.fao.org/input/download/report/728/al32REPe.pdf

[11] Are Distributed Ledger Technologies the panacea for food traceability? Consultado en https://www.sciencedirect.com/science/article/pii/S2211912418301408

[12] The Untold Story of NotPetya, the Most Devastating Cyberattack in History Consultado en https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/

APÉNDICE A
Ficheros de configuración

A.1 crypto-config.yaml

```
OrdererOrgs:
  - Name: Orderer
    Domain: hamburguesas.com
    EnableNodeOUs: true
    Specs:
      - Hostname: orderer

PeerOrgs:
  - Name: Fabricante
    Domain: fabricante.hamburguesas.com
    EnableNodeOUs: true
    Template:
      Count: 1
    Users:
      Count: 1
  - Name: Repartidor
    Domain: repartidor.hamburguesas.com
    EnableNodeOUs: true
    Template:
      Count: 1
    Users:
      Count: 1
  - Name: Proveedor
    Domain: proveedor.hamburguesas.com
    EnableNodeOUs: true
    Template:
      Count: 2
    Users:
      Count: 1
  - Name: Tienda final
    Domain: tienda_final.example.com
    EnableNodeOUs: true
    Template:
      Count: 1
    Users:
      Count: 1
```

A.2 configtx.yaml

```
---
Organizations:
    - &OrdererOrg
```

```
        Name: OrdererOrg
        ID: OrdererMSP
        MSPDir: crypto-config/ordererOrganizations/example.com/msp
        Policies:
            Readers:
                Type: Signature
                Rule: "OR('OrdererMSP.member')"
            Writers:
                Type: Signature
                Rule: "OR('OrdererMSP.member')"
            Admins:
                Type: Signature
                Rule: "OR('OrdererMSP.admin')"

    - &Proveedor
        Name: ProveedorMSP
        ID: ProveedorMSP
        MSPDir: crypto-config/peerOrganizations/proveedor.hamburguesas.com/msp
        Policies:
            Readers:
                Type: Signature
                Rule: "OR('ProveedorMSP.admin', 'ProveedorMSP.peer', '
                    ProveedorMSP.client')"
            Writers:
                Type: Signature
                Rule: "OR('ProveedorMSP.admin', 'ProveedorMSP.client')"
            Admins:
                Type: Signature
                Rule: "OR('ProveedorMSP.admin')"

    - &Fabricante
        Name: FabricanteMSP
        ID: FabricanteMSP
        MSPDir: crypto-config/peerOrganizations/fabricante.example.com/msp
        Policies:
            Readers:
                Type: Signature
                Rule: "OR('FabricanteMSP.admin', 'FabricanteMSP.peer', '
                    FabricanteMSP.client')"
            Writers:
                Type: Signature
                Rule: "OR('FabricanteMSP.admin', 'FabricanteMSP.client')"
            Admins:
                Type: Signature
                Rule: "OR('FabricanteMSP.admin')"

    - &Repartidor
        Name: RepartidorMSP
        ID: RepartidorMSP
        MSPDir: crypto-config/peerOrganizations/repartidor.example.com/msp
        Policies:
            Readers:
                Type: Signature
                Rule: "OR('RepartidorMSP.admin', 'RepartidorMSP.peer', '
                    RepartidorMSP.client')"
            Writers:
                Type: Signature
                Rule: "OR('RepartidorMSP.admin', 'RepartidorMSP.client')"
            Admins:
                Type: Signature
                Rule: "OR('RepartidorMSP.admin')"

    - &Tienda_Final
        Name: Tienda_FinalMSP
```

```
65          ID: Tienda_FinalMSP
66          MSPDir: crypto-config/peerOrganizations/tienda_final.example.com/msp
67          Policies:
68              Readers:
69                  Type: Signature
70                  Rule: "OR('Tienda_FinalMSP.admin', 'Tienda_FinalMSP.peer', '
                        Tienda_FinalMSP.client')"
71              Writers:
72                  Type: Signature
73                  Rule: "OR('Tienda_FinalMSP.admin', 'Tienda_FinalMSP.client')"
74              Admins:
75                  Type: Signature
76                  Rule: "OR('Tienda_FinalMSP.admin')"
77
78  Capabilities:
79      Channel: &ChannelCapabilities
80          V1_4_3: true
81          V1_3: false
82          V1_1: false
83      Orderer: &OrdererCapabilities
84          V1_4_2: true
85          V1_1: false
86      Application: &ApplicationCapabilities
87          V1_4_2: true
88          V1_3: false
89          V1_2: false
90          V1_1: false
91
92  Application: &ApplicationDefaults
93      Organizations:
94      Policies:
95          Readers:
96              Type: ImplicitMeta
97              Rule: "ANY Readers"
98          Writers:
99              Type: ImplicitMeta
100             Rule: "ANY Writers"
101         Admins:
102             Type: ImplicitMeta
103             Rule: "MAJORITY Admins"
104
105     Capabilities:
106         <<: *ApplicationCapabilities
107
108 Orderer: &OrdererDefaults
109     OrdererType: solo
110     Addresses:
111         - orderer.example.com:7050
112     BatchTimeout: 2s
113     BatchSize:
114         MaxMessageCount: 10
115         AbsoluteMaxBytes: 99 MB
116         PreferredMaxBytes: 512 KB
117
118     Kafka:
119         Brokers:
120             - 127.0.0.1:9092
121
122     Organizations:
123
124     Policies:
125         Readers:
126             Type: ImplicitMeta
127             Rule: "ANY Readers"
```

```
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
        BlockValidation:
            Type: ImplicitMeta
            Rule: "ANY Writers"

Channel: &ChannelDefaults
    Policies:
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
    Capabilities:
        <<: *ChannelCapabilities

Profiles:

    SupplyOrdererGenesis:
        <<: *ChannelDefaults
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *Proveedor
                    - *Fabricante
                    - *Repartidor
                    - *Tienda_final
    Cadena_de_Suministro:
        Consortium: SuministroConsortium
        <<: *ChannelDefaults
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Proveedor
                - *Fabricante
                - *Repartidor
                - *Tienda_final
            Capabilities:
                <<: *ApplicationCapabilities
```

APÉNDICE B
Lógica aplicación

B.1 Chaincode

```
'use strict';

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

class CadenaSuministro extends Contract {


  async addAsset(ctx, asset) {
    console.info('============= START : Add asset ===========');
    await ctx.stub.putState(JSON.parse(asset).id.toString(), Buffer.from(asset)
        );
    console.info('============= END : Add asset ===========');
    return ctx.stub.getTxID()
  }

  async queryAsset(ctx, assetId) {
    const assetAsBytes = await ctx.stub.getState(assetId);
    if (!assetAsBytes || assetAsBytes.length === 0) {
      throw new Error(`${assetId} does not exist`);
    }
    console.log(sassetAsBytes.toString());
    console.info('============= END : Query asset ===========');
    return assetAsBytes.toString();
  }

  async setPosition(ctx, id, latitude, longitude) {
    console.info('============= START : Set position ===========');
    const keyAsBytes = await ctx.stub.getState(id);
    if (!keyAsBytes || keyAsBytes.length === 0) {
      throw new Error(`${id} does not exist`);
    }
    let key = JSON.parse(keyAsBytes.toString());
    key.latitude = latitude;
    key.longitude = longitude;
    await ctx.stub.putState(id, Buffer.from(JSON.stringify(key)));
    console.info('============= END : Set position ===========');
    return ctx.stub.getTxID();
  }

  async setOwner(ctx, id, owner) {
    console.info('============= START : Set Owner ===========');
    const keyAsBytes = await ctx.stub.getState(id);
    if (!keyAsBytes || keyAsBytes.length === 0) {
      throw new Error(`${id} does not exist`);
```

```
    }
    let asset = JSON.parse(keyAsBytes.toString());
    asset.owner = owner;
    await ctx.stub.putState(id, Buffer.from(JSON.stringify(asset)));
    console.info('============= END : Set Owner ===========');
    return ctx.stub.getTxID();
  }

  async getHistory(ctx, id) {
    console.info('============= START : Query History ===========');
    let iterator = await ctx.stub.getHistoryForKey(id);
    let result = [];
    let res = await iterator.next();
    while (!res.done) {
      if (res.value) {
        console.info(`found state update with value: ${res.value.value.toString
            ('utf8')}`);
        const obj = JSON.parse(res.value.value.toString('utf8'));
        result.push(obj);
      }
      res = await iterator.next();
    }
    await iterator.close();
    console.info('============= END : Query History ===========');
    return result;
  }


}

module.exports = CadenaSuministro;
```

B.2 API

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
const fabricNetwork = require('./fabricNetwork')
app.set('view engine', 'ejs');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
  extended: true
}));
app.post('/api/nuevaCarne', async function (req, res) {
  try {
    const contract = await fabricNetwork.connectNetwork('connection-producer.
        json', 'wallet/wallet-proveedor');
    let carne = {
      id: req.body.id,
      latitude: req.body.latitude,
      longitude: req.body.longitude,
      peso: req.body.peso,
      owner: req.body.owner
    }
    let tx = await contract.submitTransaction('addAsset', JSON.stringify(carne)
        );
    res.json({
      status: 'OK - Transaction has been submitted',
      txid: tx.toString()
    });
  } catch (error) {
    console.error(`Failed to evaluate transaction: ${error}`);
```

```
27      res.status(500).json({
28        error: error
29      });
30    }
31  });
32
33  app.get('/api/datoProducto/:id', async function (req, res) {
34    try {
35      const contract = await fabricNetwork.connectNetwork('connection-retailer.
            json', 'wallet/wallet-tiendaFinal');
36      const result = await contract.evaluateTransaction('queryAsset', req.params.
            id.toString());
37      let response = JSON.parse(result.toString());
38      res.json({result:response});
39    } catch (error) {
40      console.error(`Failed to evaluate transaction: ${error}`);
41      res.status(500).json({
42        error: error
43      });
44    }
45  })
46
47  app.post('/api/nuevaLocalizacion', async function (req, res) {
48    try {
49      const contract = await fabricNetwork.connectNetwork('connection-deliverer.
            json', 'wallet/wallet-repartidor');
50      let tx = await contract.submitTransaction('setPosition', req.body.id.
            toString(), req.body.latitude.toString(), req.body.longitude.toString()
            );
51      res.json({
52        status: 'OK - Transaction has been submitted',
53        txid: tx.toString()
54      });
55    } catch (error) {
56      console.error(`Failed to evaluate transaction: ${error}`);
57      res.status(500).json({
58        error: error
59      });
60    }
61
62  });
63
64  app.post('/api/nuevoDueno', async function (req, res) {
65    try {
66      const contract = await fabricNetwork.connectNetwork('connection-deliverer.
            json', 'wallet/wallet-repartidor');
67      let tx = await contract.submitTransaction('setOwner', req.body.id.toString
            (), req.body.owner.toString());
68      res.json({
69        status: 'OK - Transaction has been submitted',
70        txid: tx.toString()
71      });
72    } catch (error) {
73      console.error(`Failed to evaluate transaction: ${error}`);
74      res.status(500).json({
75        error: error
76      });
77    }
78  });
79
80
81  app.get('/api/historial/:id', async function (req, res) {
82    try {
```

```
83      const contract = await fabricNetwork.connectNetwork('connection-producer.
            json', 'wallet/wallet-Proveedor');
84      const historySushi = JSON.parse((await contract.evaluateTransaction('
            getHistory', req.params.id.toString())).toString());
85      const actualSushi = JSON.parse((await contract.evaluateTransaction('
            queryAsset', req.params.id.toString())).toString());
86      historySushi.unshift(actualSushi);
87      const historyTuna = JSON.parse((await contract.evaluateTransaction('
            getHistory', actualSushi.carneId.toString())).toString());
88      const actualTuna = JSON.parse((await contract.evaluateTransaction('
            queryAsset', actualSushi.carneId.toString())).toString());
89      historyTuna.unshift(actualTuna);
90      historialCarne = historyTuna
91      historialBurguer = historySushi
92      res.json({
93        historialCarne: historialCarne,
94        historialBurguer: historialBurguer
95      });
96    } catch (error) {
97      console.error(`Failed to evaluate transaction: ${error}`);
98      res.status(500).json({
99        error: error
100     });
101   }
102 })
103
104 app.post('/api/nuevaHamburguesa', async function (req, res) {
105   try {
106     const contract = await fabricNetwork.connectNetwork('connection-
            manufacturer.json', 'wallet/wallet-fabricante');
107     let sushi = {
108       id: req.body.id,
109       latitude: req.body.latitude,
110       longitude: req.body.longitude,
111       peso: req.body.peso,
112       owner: req.body.owner,
113       carneId: req.body.carneId
114     }
115     let tx = await contract.submitTransaction('addAsset', JSON.stringify(sushi)
            );
116     res.json({
117       status: 'OK - Transaction has been submitted',
118       txid: tx.toString()
119     });
120   } catch (error) {
121     console.error(`Failed to evaluate transaction: ${error}`);
122     res.status(500).json({
123       error: error
124     });
125   }
126 })
127
128 app.listen(3000, ()=>{
129   console.log("**********************************");
130   console.log("API server listening at localhost:3000");
131   console.log("**********************************");
132 });
```